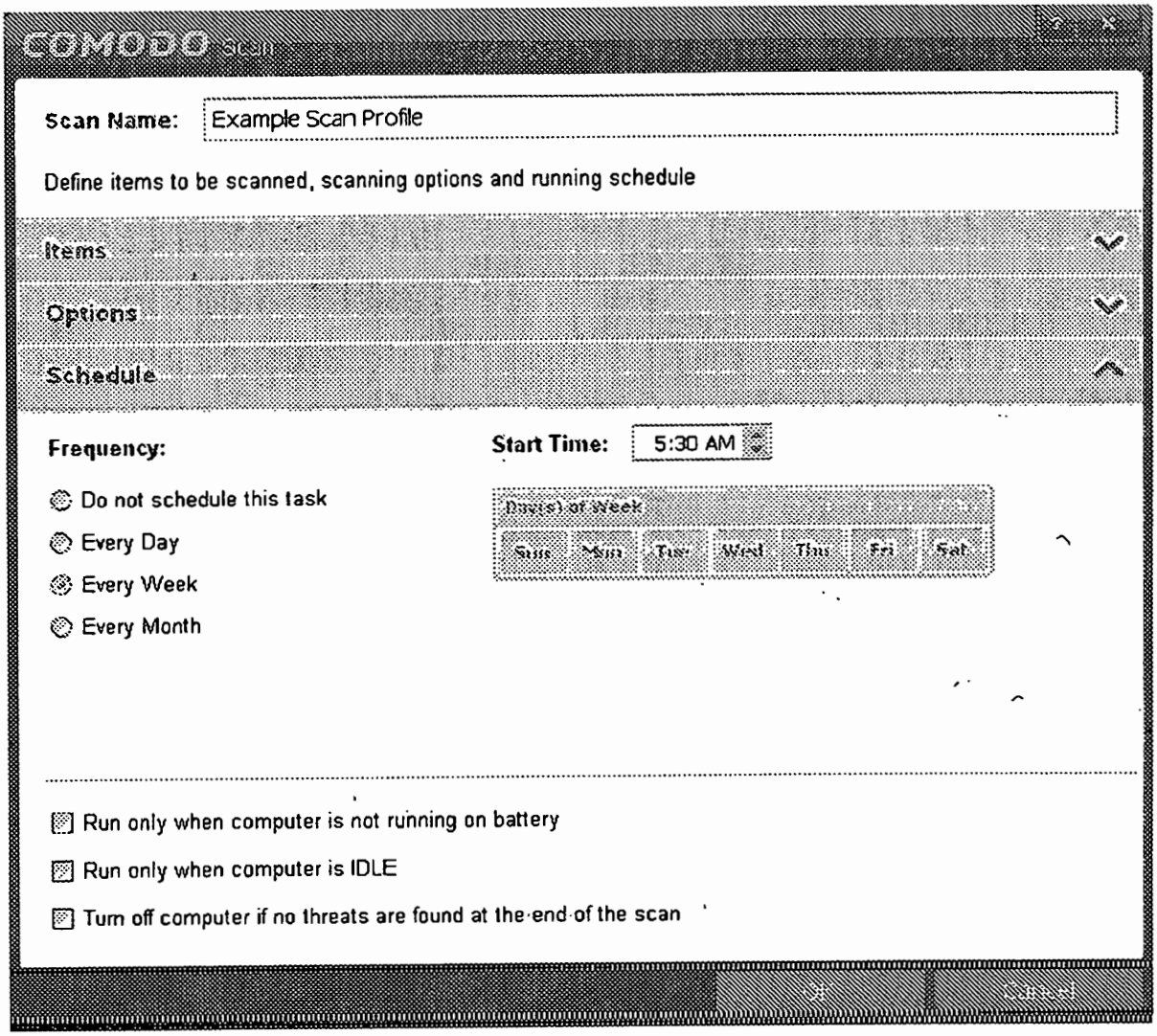


- **Do not schedule this task** - The scan profile will be created but will not be run automatically. The profile will be available for manual on-demand scanning
- **Every Day** - The Antivurus starts scanning the areas defined in the scan profile every day at the time specified in the Start Time field
- **Every Week** - The Antivurus starts scans the areas defined in the scan profile on the day(s) of the week specified in 'Days of the Week' field and the time specified in the 'Start Time' field. You can select the days of the week by directly clicking on them.
- **Every Month** - The Antivurus starts scans the areas defined in the scan profile on the day(s) of the month specified in 'Days of the month' field and the time specified in the 'Start Time' field. You can select the days of the month by directly clicking on them.
- **Run only when computer is not running on battery** - This option is useful when you are using a laptop or any other battery driven portable computer. Selecting this option runs the scan only if the computer runs with the adopter connected to mains supply and not on battery.
- **Run only when computer id IDLE** - Select this option if you do not want to disturbed when involved in computer related activities. The scheduled can will run only if the computer is in idle state
- **Turn off computer if no threats are found at the end of the scan** - Selecting this option turns your computer off, if no threats are found during the scan. This is useful when you are scheduling the scans to run at nights.

- Click OK to save the profile.

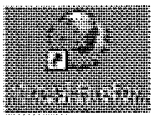
The profile will be saved and the selected areas will be scanned repeatedly as per the set schedule.

Note: The schedule scan will run only if it is enabled. Click the button under the Active column beside the respective profile row

to toggle between on and off status.

Run Untrusted Programs In the Sandbox

Comodo Internet Security allows you to run programs inside the Sandbox on a 'one-off' basis. This is helpful to test the behavior of new executables that you have downloaded or for applications that you are not sure that you trust. You can also create a desktop shortcut to run the application inside the sandbox on future occasions. The following image shows hows a 'virtual' shortcut will appear on your desktop:



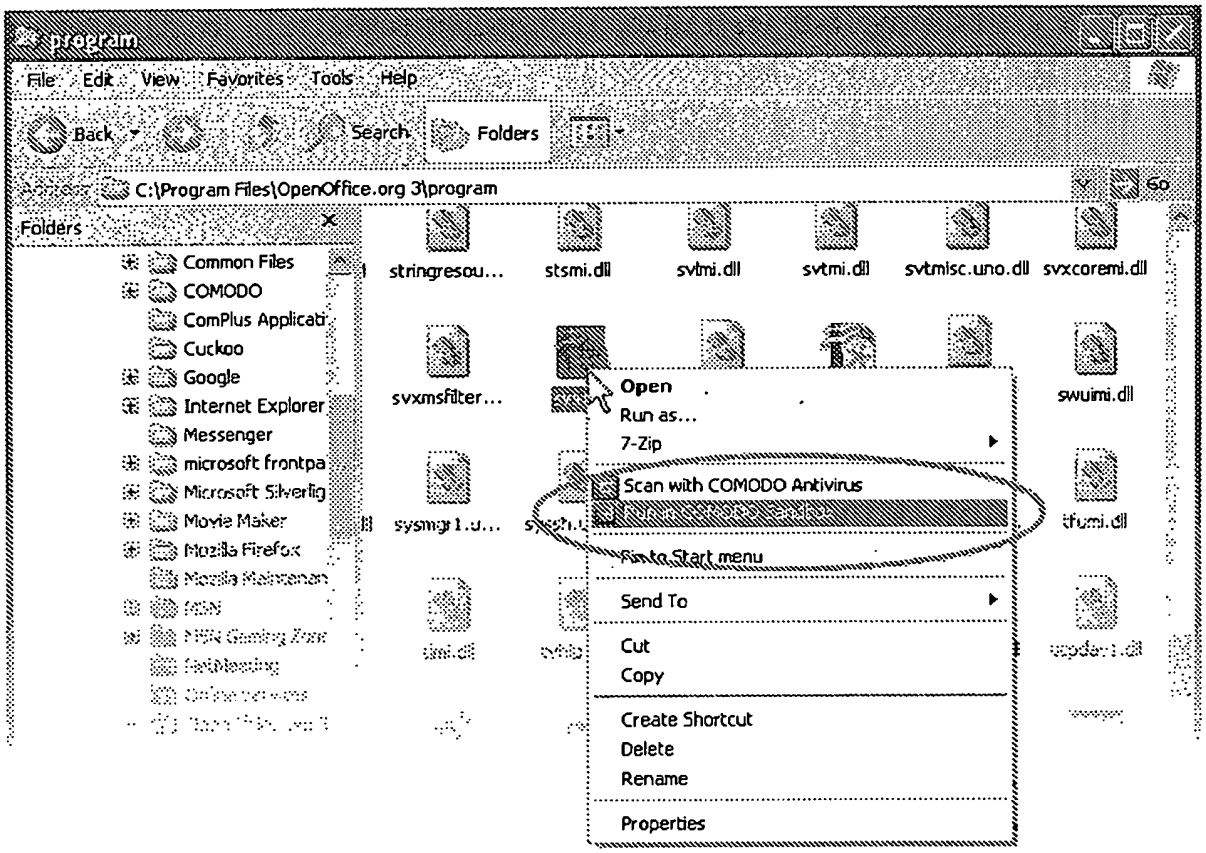
Comodo Internet Security allows you to run a program in the sandbox:

- From the right click options
- From the Sandbox Tasks interface

Note: If you wish to run an application in the sandbox on a long-term/permanent basis then add the file to the Sandbox.

Run a program inside the sandbox through right click options

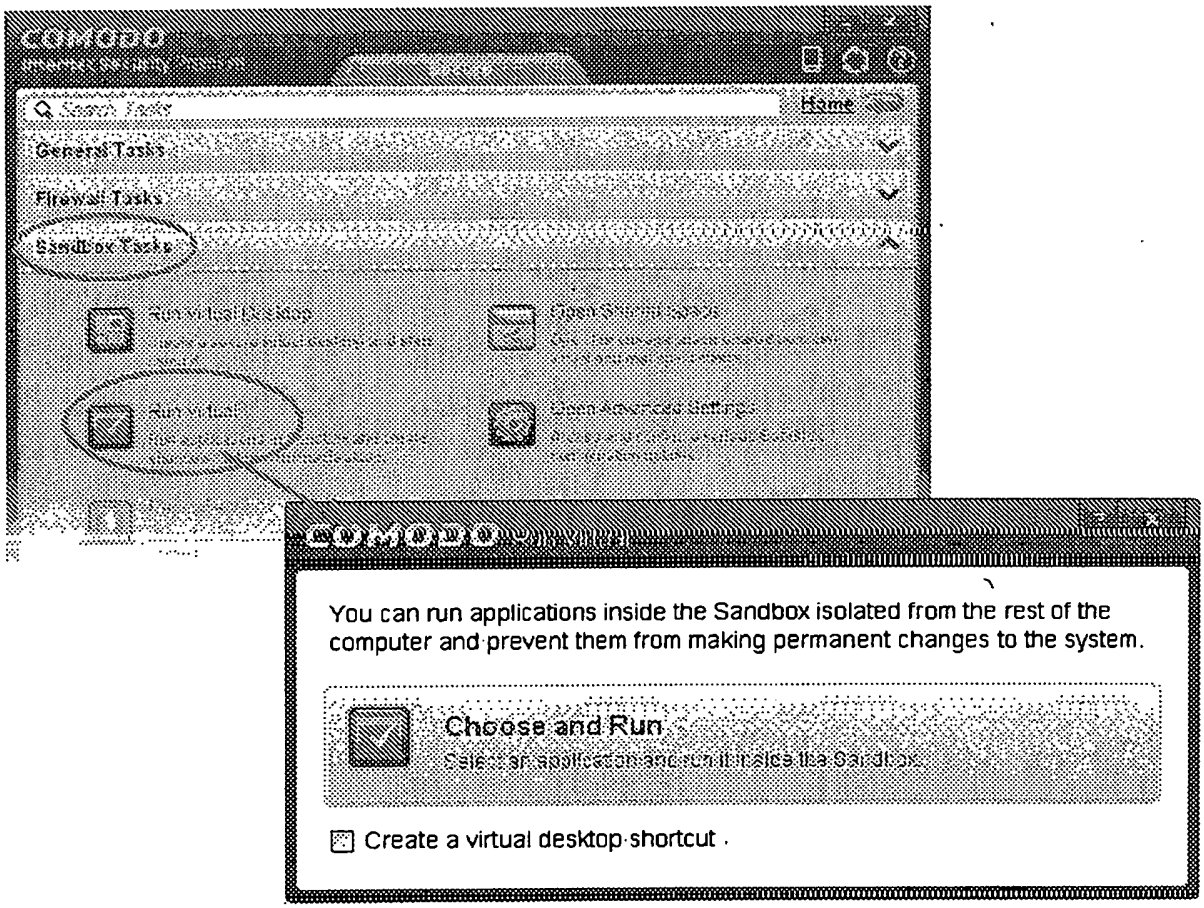
1. Browse to the installation folder of the .exe file through Windows Explorer
2. Right click on the program that you want to run inside the sandbox



3. Choose 'Run in Comodo Sandbox' from the context sensitive menu

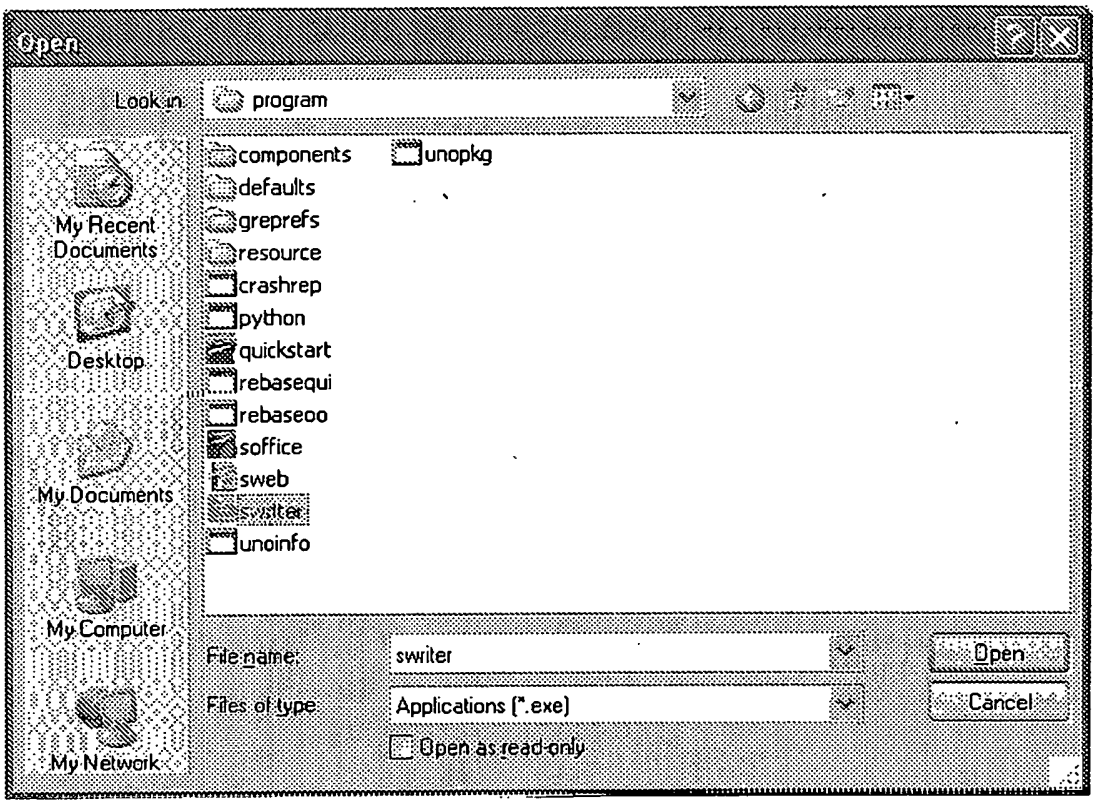
Run a program in sandbox from Sandbox Tasks interface

1. Click the 'Tasks' arrow on the home screen to open the main Tasks menu
2. Click 'Sandbox Tasks' and click 'Run Virtual' from the 'Sandbox Tasks' interface



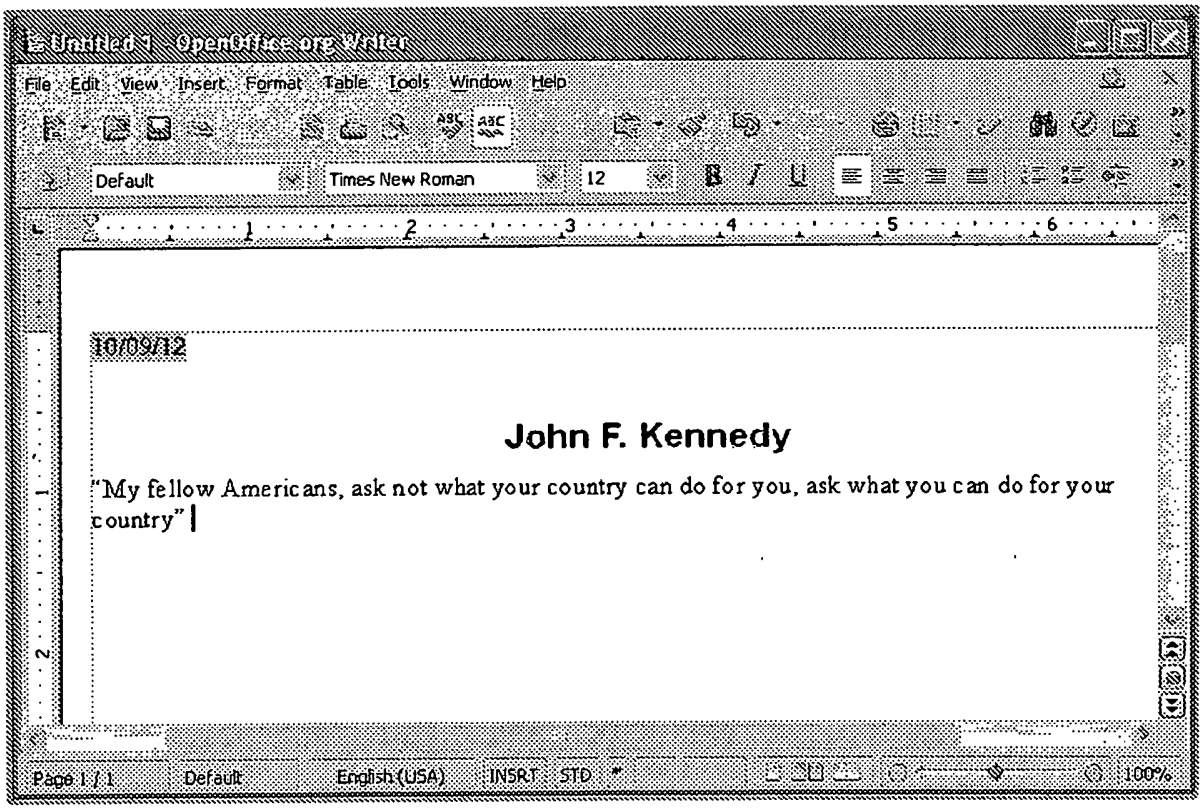
The 'Run Virtual' dialog will be displayed.

3. To run an application inside the sandbox, click 'Choose and Run' then browse to the application. The application will run with a green border indicating that it is sandboxed. If you wish to run the application in the sandbox in future, then select 'Create a virtual desktop shortcut'.



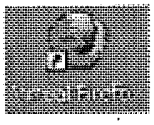
4. Browse to the application and click 'Open'. In the example above, Open Office Writer is chosen.

The application will run in the Sandbox on this occasion only. If you often want the browser to run sandboxed then create a 'virtual shortcut' for the application by selecting the check-box 'Create a virtual desktop shortcut' in step 2. If you wish to run an application in the sandbox on a long-term/permanent basis then add the file to the Sandbox.



Run Browsers Inside Sandbox

This page explains how to run your Internet browser inside the sandbox. Surfing the Internet with a sandboxed browser is the same as normal, with the benefit that any malicious files you inadvertently download cannot do damage your real computer. You can also create a desktop shortcut to run the browser inside the sandbox on future occasions. The following image shows how a 'virtual' shortcut will appear on your desktop:



Comodo Internet Security allows you to run a browser in the sandbox:

- From the desktop widget
- From the Sandbox Tasks interface

Starting a browser from the desktop widget

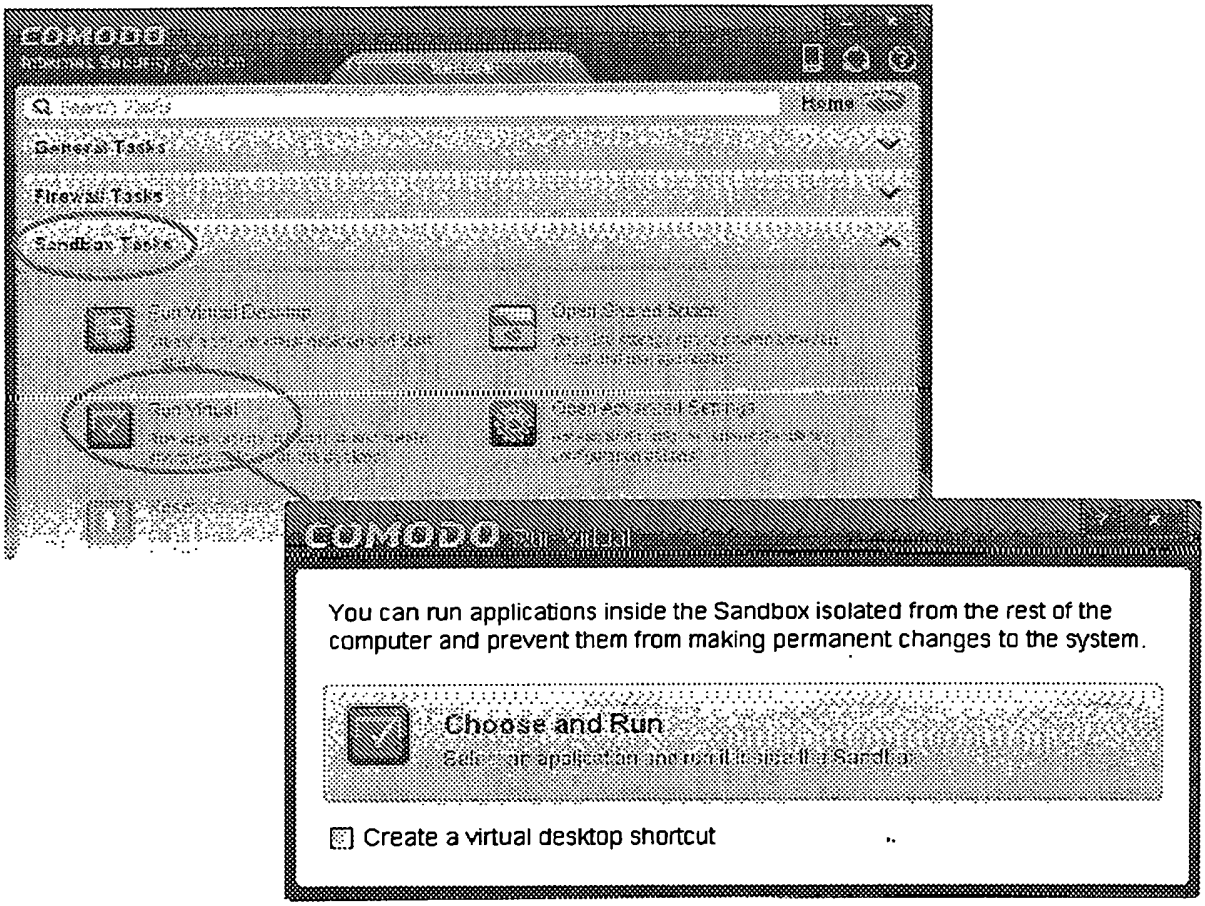
The CIS Desktop Widget displays shortcut icons of the browsers installed in your computer.



- To start a browser inside the sandbox, click on the browser icon.

Starting a browser from the Sandbox Tasks interface

1. Click the 'Tasks' arrow on the home screen to open the main Tasks menu
2. Click 'Sandbox Tasks' and click 'Run Virtual' from the 'Sandbox Tasks' interface



The 'Run Virtual' dialog will be displayed.

3. To run a browser inside the sandbox, click 'Choose and Run', navigate to the installation location of the browser and select the exe file of the browser. If you wish to create a desktop shortcut to run the browser in the sandbox in future, then select 'Create a virtual desktop shortcut'.

The browser will run with a green border indicating that it is sandboxed.

Run Untrusted Programs Inside Virtual Desktop

This page explains how to run untrusted programs inside the Virtual Desktop. Applications running in the virtual desktop also leave no cookies or history behind on your real system, making it ideal for testing out beta/unstable software.

Applications installed or the files stored in your system can be opened inside the Virtual Desktop by the following methods:

- Opening the applications/files from the desktop shortcuts
- Sharing the application/files through the Shared Space folder

Desktop Shortcuts

You can copy the files or create shortcuts for the applications/files to be opened in Virtual Desktop, in the desktop of your real system. The shortcuts of your real desktop will be available in the Virtual Desktop. You can double click on the icon to open the respective application of file inside the Virtual Desktop.

Note: The real computer desktop icons will be available only in the Classic Windows Mode and Tablet + Classic Mode.

Shared Space

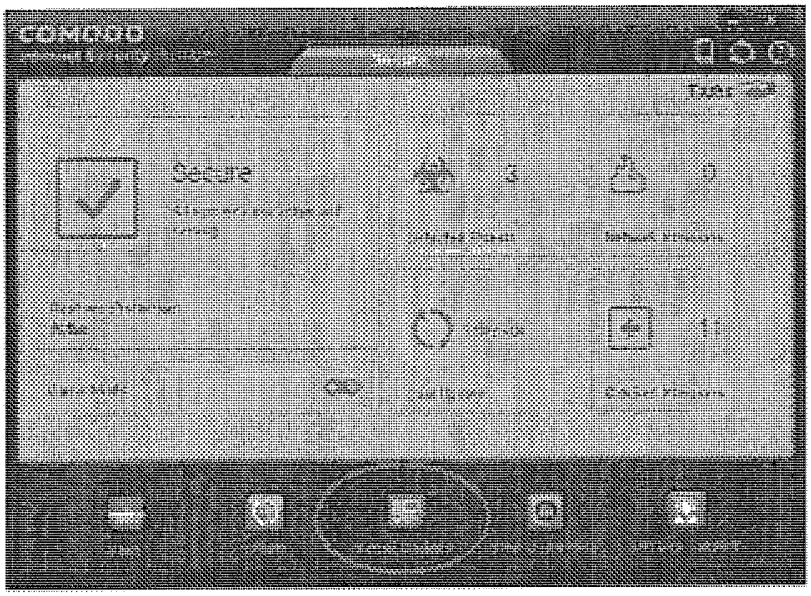
The Virtual Desktop creates a folder Shared Space in the location "C:\Documents and Settings\All Users\Application Data\Shared Space", which can be shared by your host operating system and the Virtual Desktop.

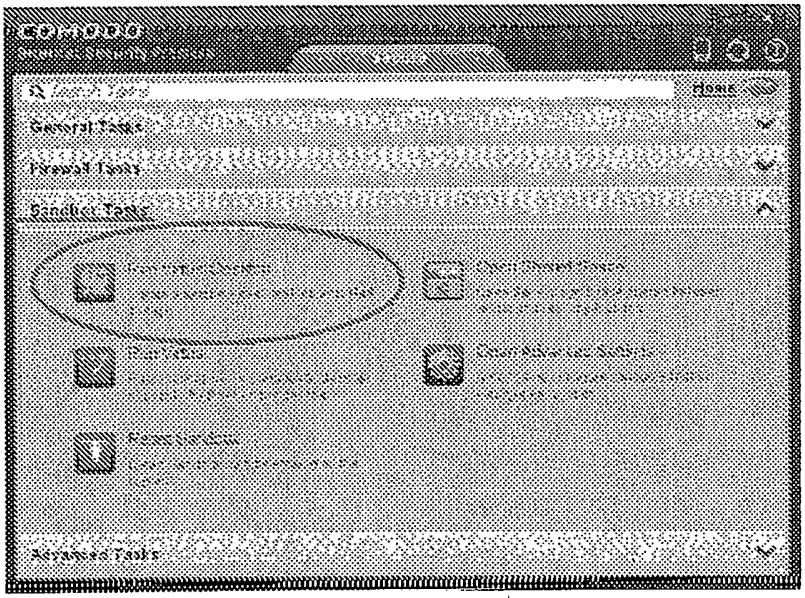
The Shared Space can be accessed in the following ways:

- Click 'Open Shared Space' under 'Sandbox' Tasks in the Tasks interface
- Click the 'Shared Space' shortcut icon from the home screen of CIS
- Click the 'Shared Space' shortcut icon from the CIS widget
- Click the 'Shared Space' desktop shortcut icon

To open an application or file from your host system in the Virtual Desktop

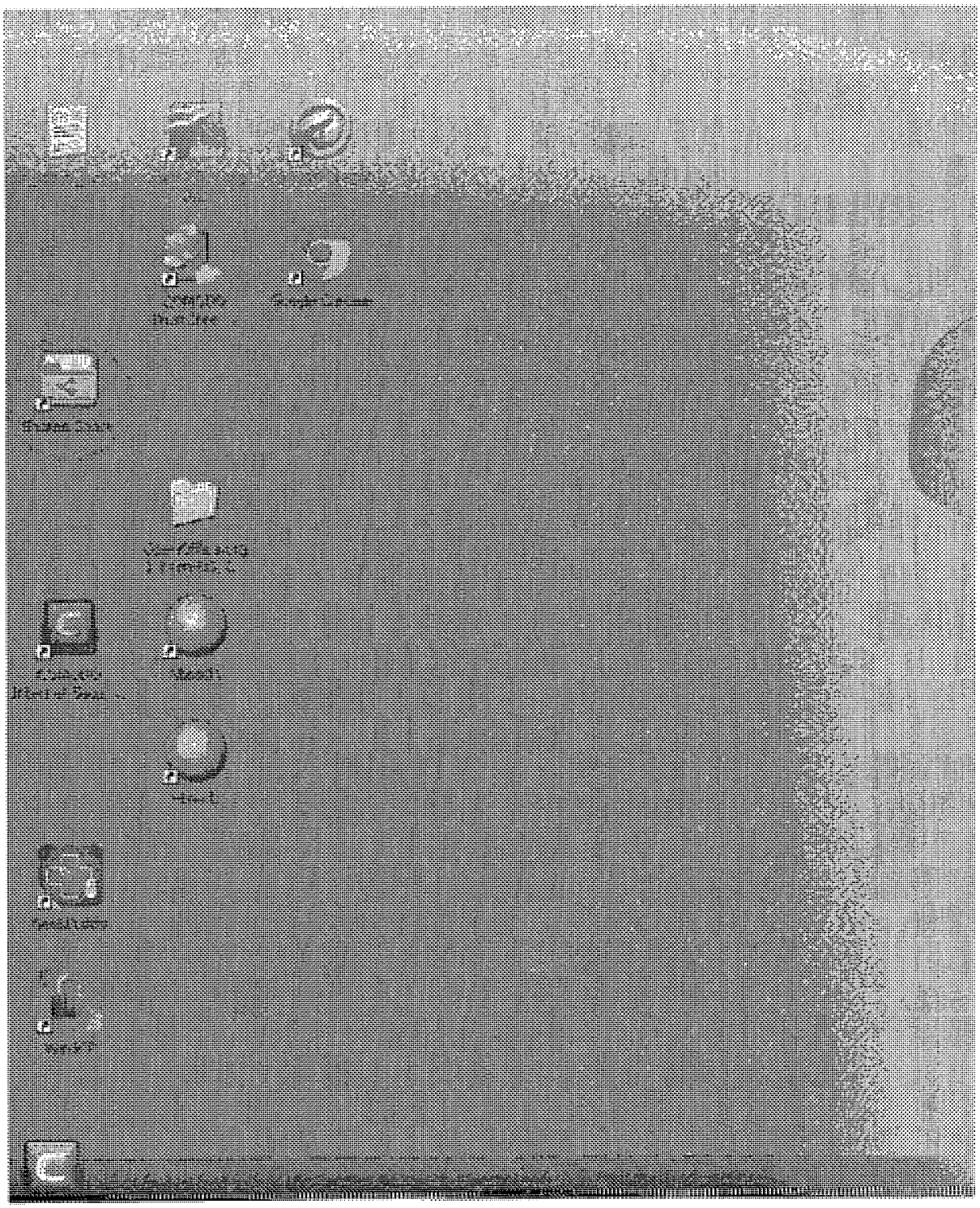
1. Open the 'Shared Space' as mentioned above
2. Copy/Move the application or the file to be opened into the Shared Space
3. Start 'Virtual Desktop' by clicking the Virtual Desktop shortcut in the CIS home screen or by clicking Sandbox Tasks > Run Virtual Desktop from the Tasks interface of CIS.





4. Open Shared Space inside the Virtual Desktop by clicking the 'Shared Space' icon in the home screen.

Note: The Shared space home screen icon will be available only in the Classic Windows Mode and Tablet + Classic Mode.



5. Double click on the application/file in the shared space to open it inside the virtual Desktop.

The changes you make to the file will be stored to the file only inside the Virtual Desktop and not in the real computer system.

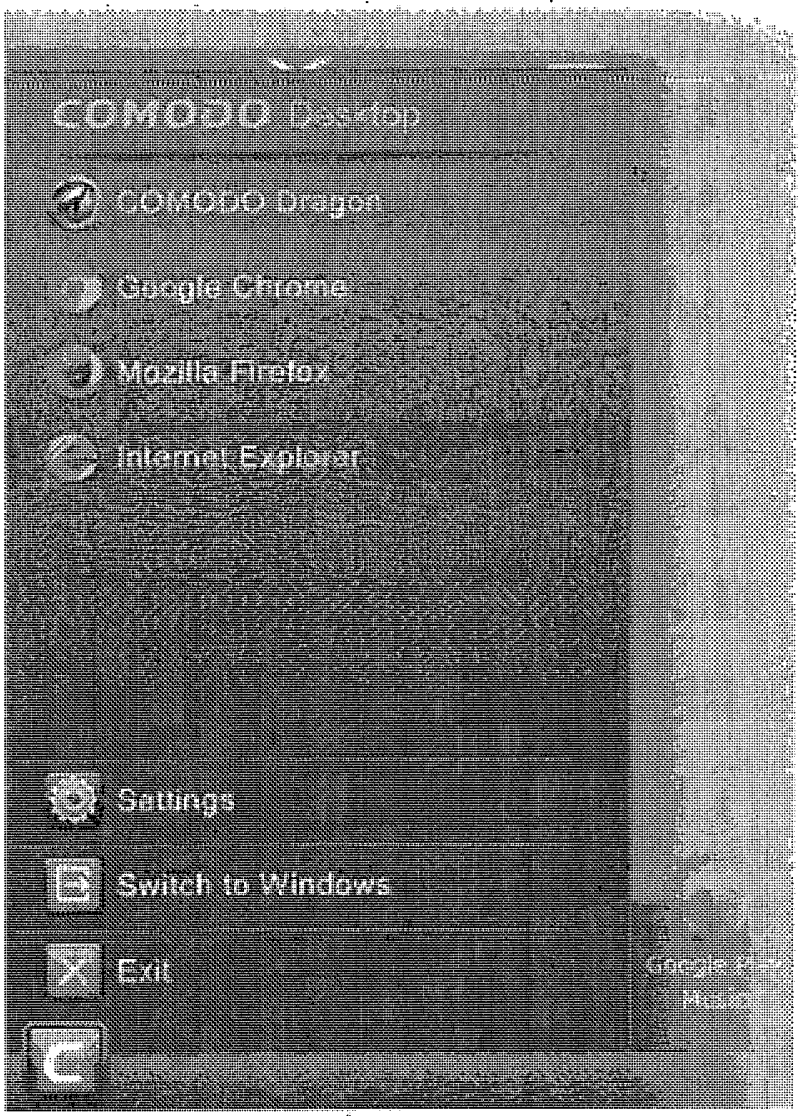
Run Browsers Inside the Virtual Desktop

The Virtual Desktop provides an extremely secure environment for internet related activities because it isolates your browser

from the rest of your computer. Just by visiting them, malicious websites can install viruses malware, rootkits and spyware onto your computer that can allow hackers to steal confidential information. Surfing the 'net inside the virtual desktop removes this threat to your machine by preventing any website from installing anything on your real computer. Furthermore, the Virtual Keyboard allows you to securely enter user-names, credit card numbers and passwords without fear of key-logging software recording your physical keystrokes.

To run a browser inside the Virtual Desktop

1. Click the 'C' button
2. Select the browser you want to run from the browsers displayed.



The browser will open inside the Virtual Desktop.



On completion of your browsing session, no trails like your browsing history will be stored in your computer.

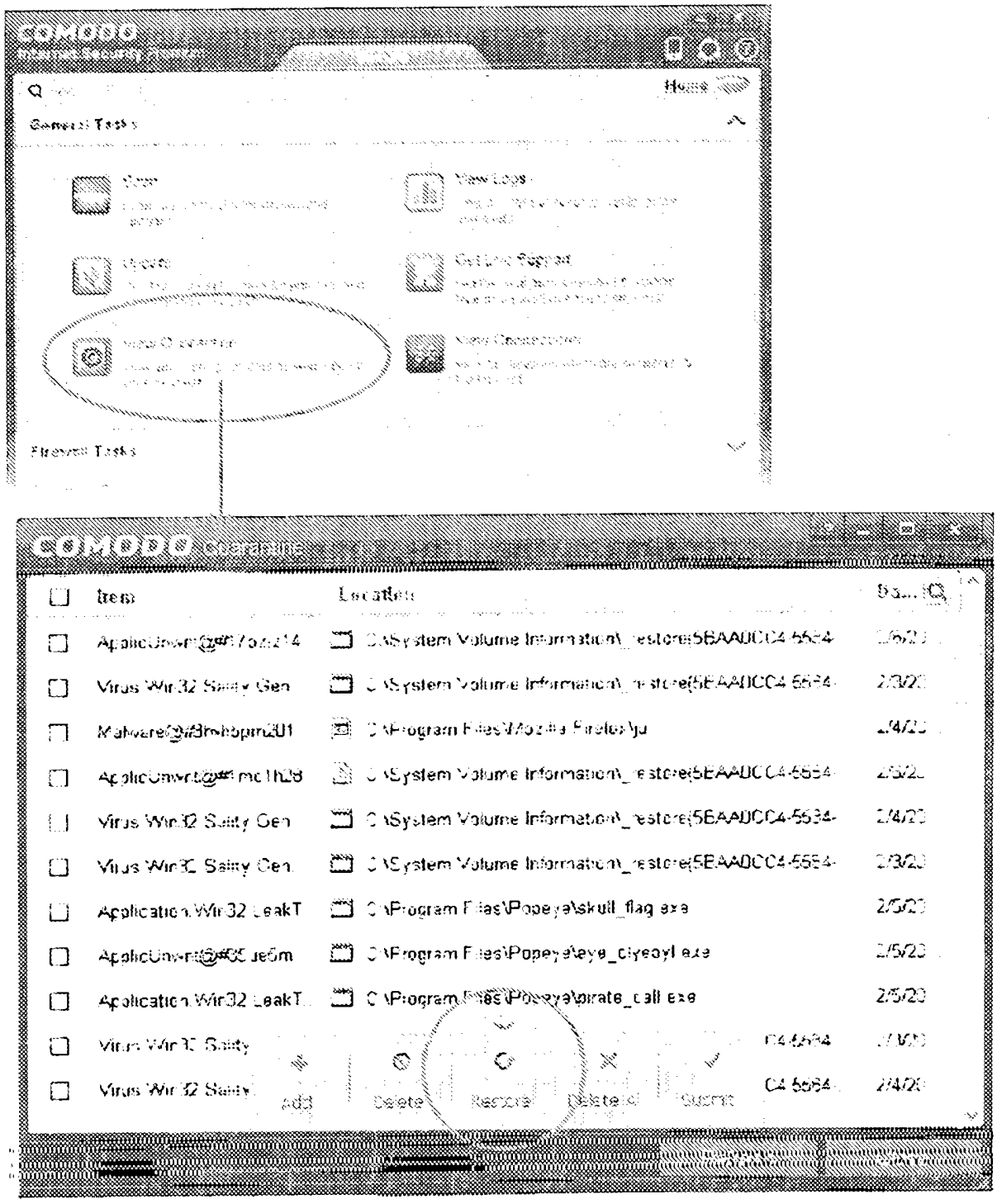
Restore Incorrectly Quarantined Item(s)

If you have incorrectly quarantined item(s) or you feel an item has been incorrectly quarantined by the application (a false positive) then you can restore it/them using the following procedure:

To submit Quarantined items

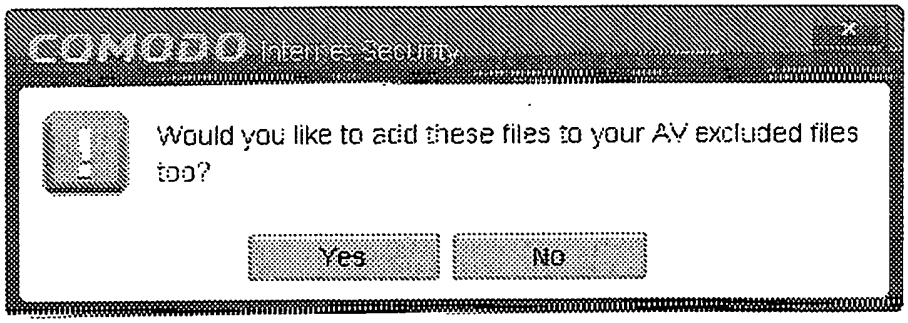
1. Click the 'Tasks' arrow on the home screen to open the main Tasks menu
2. In 'General Tasks', click 'View Quarantine'

The 'Quarantine' interface will open. The interface displays a list of items moved to Quarantine manually, from the results of real-time scanning, on-demand scanning and scheduled scans.



3. Choose the items to be restored by selecting the checkboxes beside them.
4. Click the handle from the bottom and choose 'Restore'.

An option to add the selected items to AV excluded files will open.



If you select 'Yes', these items will not be included for AV scans. If you select 'No', these items will be included for AV scans and quarantined during the next scanning.

All the selected files will be restored to their original locations immediately.

5. Click 'Close' button to exit.

Click here for more details on the Quarantined Items.

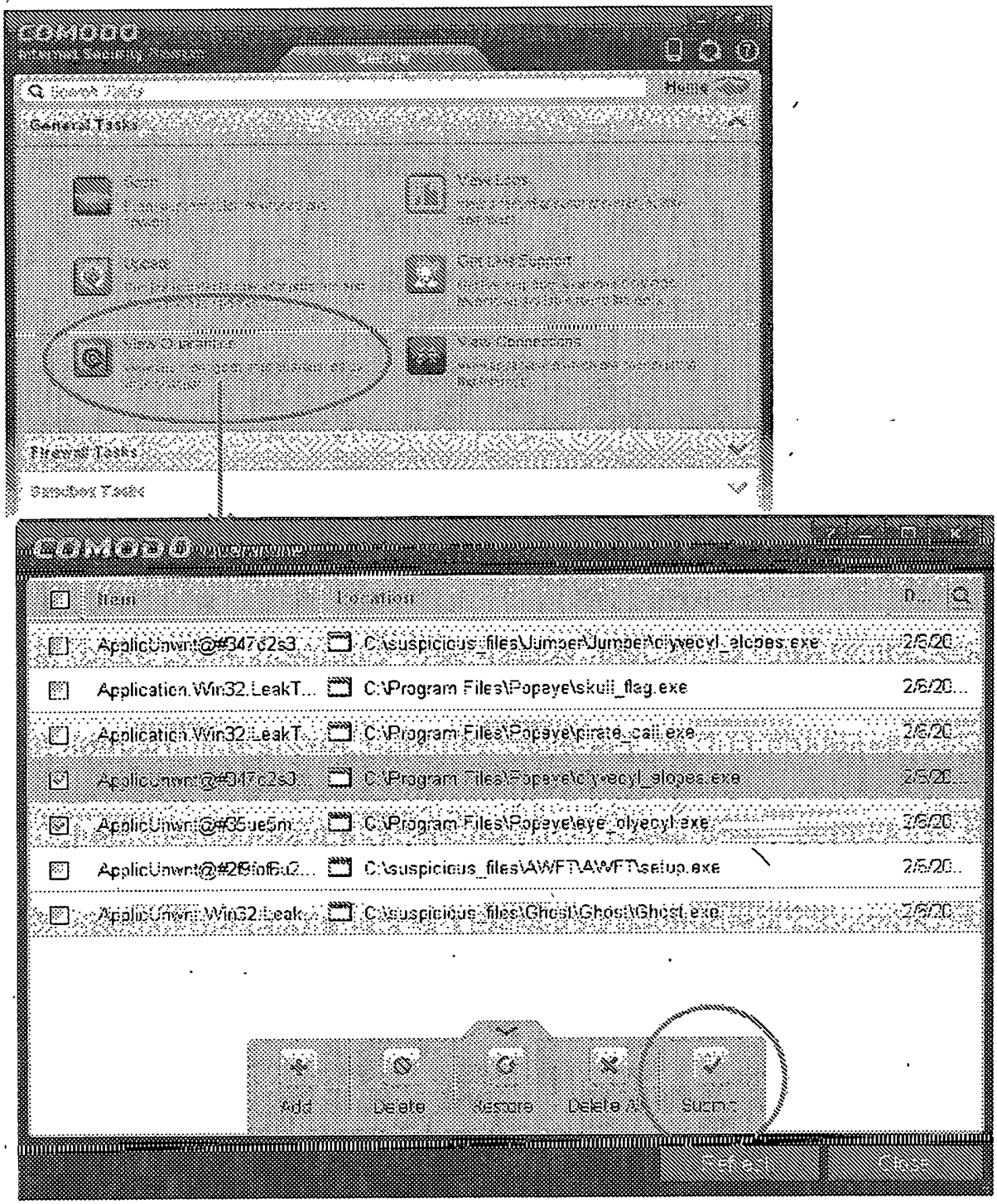
Submit Quarantined Items to Comodo for Analysis

Items which have been quarantined as a result of an On Access, On Demand or Scheduled Scans, can be sent to Comodo for Analysis. After the analysis, if the submitted item is found to be a False Positive, it will be added to Comodo Safe List. Conversely, if it is found to be a malware, it will be added to the anti-malware Black list. This helps Comodo to enhance its virus signature database and helps millions of other CIS users to benefit out of it. Click here for more details on Quarantined Items.

To submit Quarantined items

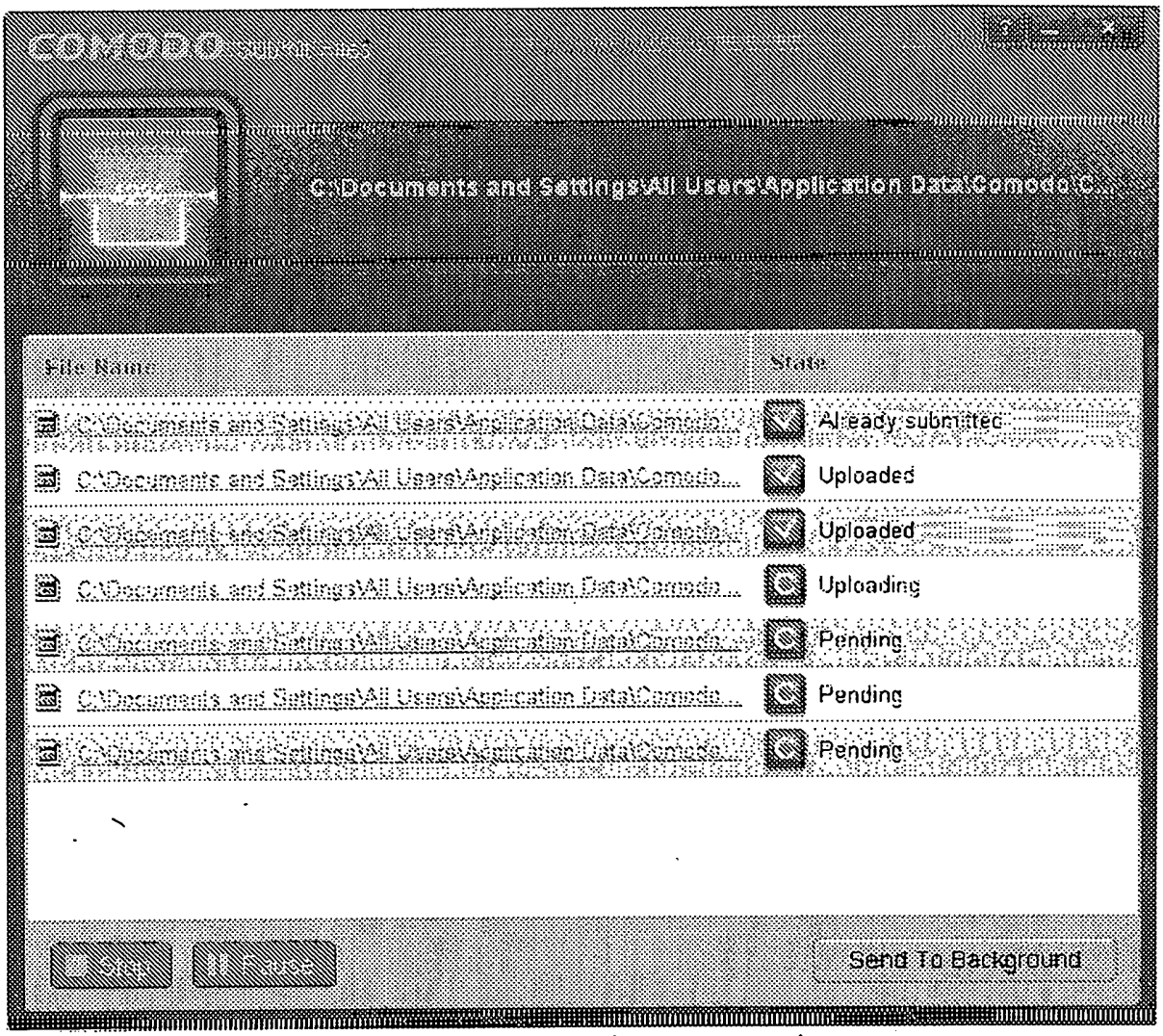
1. Click the 'Tasks' arrow on the home screen to open the main Tasks menu
2. In 'General Tasks', click 'View Quarantine'

The 'Quarantine' interface will open. The interface displays a list of items moved to Quarantine manually, from the results of real-time scanning, on-demand scanning and scheduled scans.



3. Choose the items to be submitted to Comodo for analysis by selecting the checkboxes beside them.
4. Click the handle from the bottom and choose 'Submit'.

The submission progress will be indicated.



On completion, the submission results will be displayed, indicating whether the file is successfully submitted or already submitted by other users and is pending for analysis.



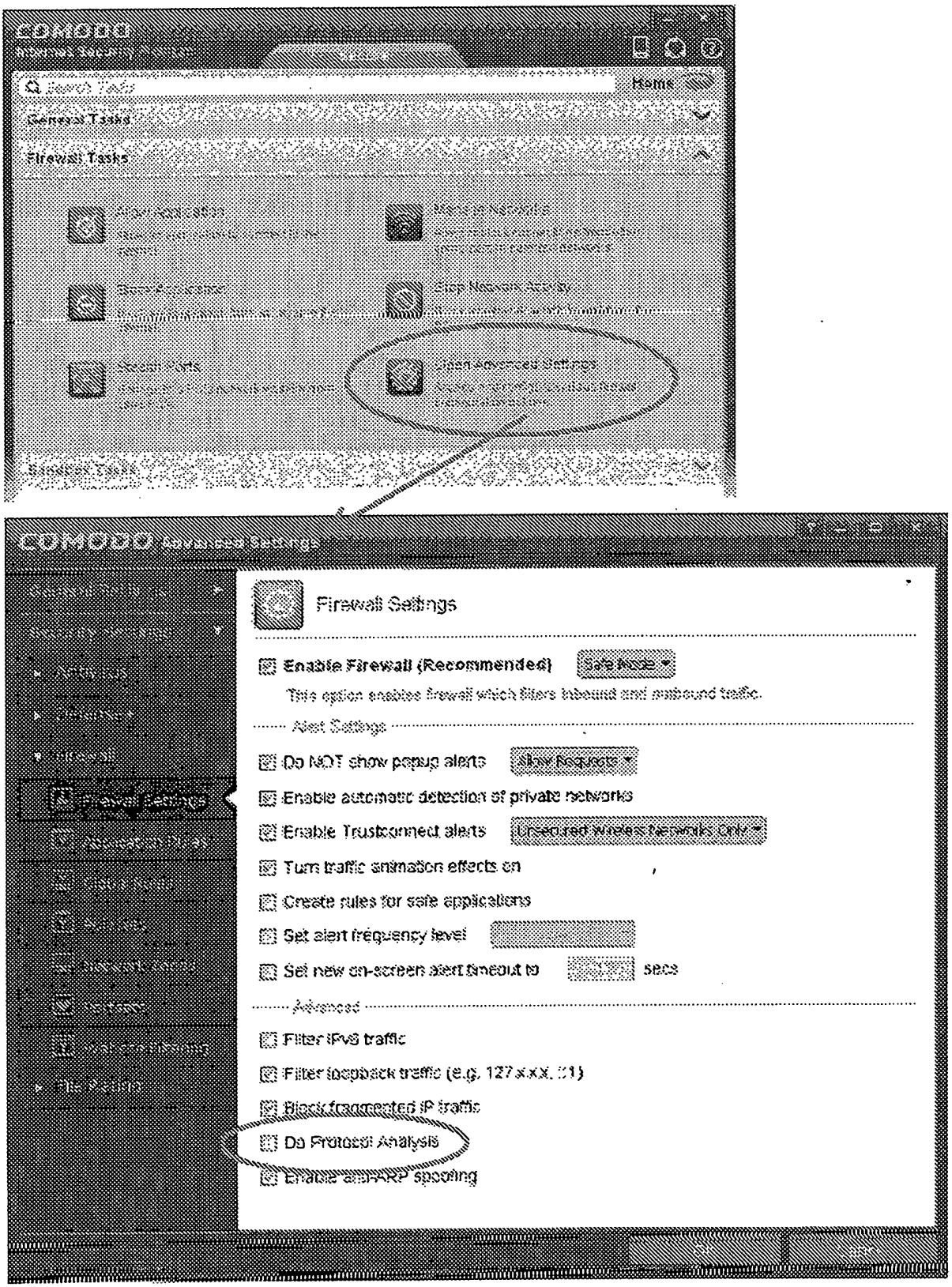
Enable File Sharing Applications like BitTorrent and Emule

This page explains how to configure Comodo Firewall for file sharing applications like Shareaza/Emule and BitTorrent/UTorrent. To allow these file sharing applications, you must:

- Disable 'Do Protocol analysis' *(disabled, by default)*
- Create a 'Predefined Firewall Ruleset' for Shareaza/Emule
- Create a 'Predefined Firewall Ruleset' for BitTorrent/Utorrent

To Disable 'Do Protocol analysis'

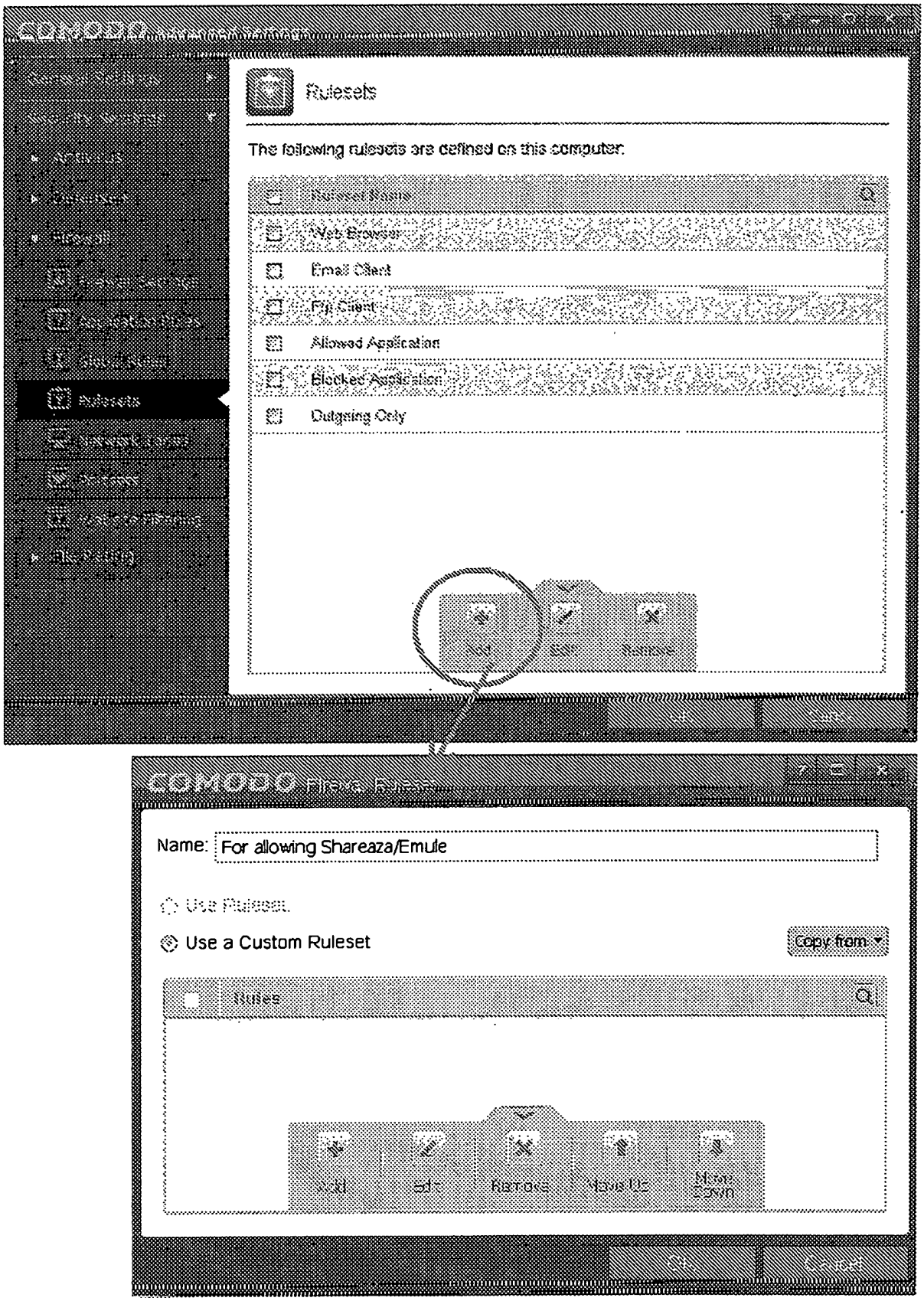
1. Open 'Tasks' interface by clicking the green curved arrow at top right of the 'Home' screen
2. Open 'Firewall Tasks' by clicking 'Firewall Tasks' from the Tasks interface and click 'Open Advanced Settings'.



3. Ensure that 'Do Protocol Analysis' checkbox is not selected.

To Create a 'Predefined Firewall Ruleset' for Shareaza/Emule

1. Click 'Rulesets' under 'Firewall' from the LHS navigation pane of 'Advanced Settings' interface to open 'Rulesets' panel
2. Click the handle from the bottom of the panel and choose 'Add'

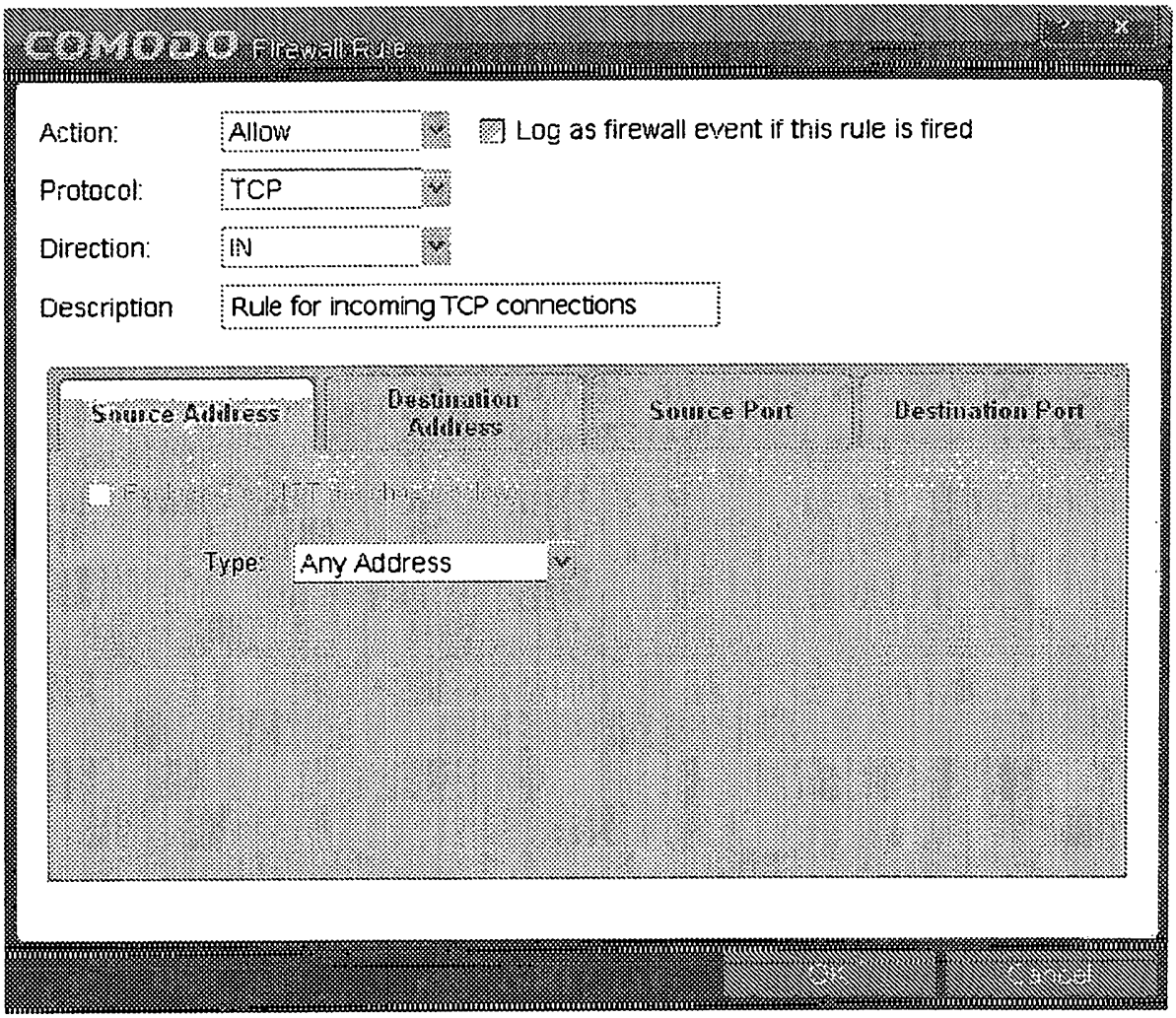


The 'Firewall Ruleset' interface will open for creating a new set of rules.

3. Click the handle from the bottom and choose 'Add'

4. Enter a descriptive name for the new ruleset to be created in the 'Description' text box (for example: For allowing Shareaza/Emule).

5. Now you need to create six rules for the newly created ruleset. To do so, click 'Add'. The 'Firewall Rule' interface will appear. For creating each rule, select the check box and choose the drop-down options under each tab as given below. After creating each rule, click 'OK' for the rule to be added. Click handle in the 'Firewall Ruleset' interface and choose 'Add' to create the next rule.



Rule 1

- Action : Allow
- Protocol : TCP
- Direction : In
- Description : Rule for incoming TCP connections
- Source Address : Any Address
- Destination Address : Any Address
- Source port : A Port Range : (Start Port = 1025 / End Port = 65535)
- Destination port : A Single Port : (Port : Your TCP port of Shareaza/Emule)

Rule 2

- Action : Allow
- Protocol : UDP
- Direction : In
- Description : Rule for incoming UDP connections

- Source Address : Any Address
- Destination Address : Any Address
- Source port : A Port Range : (Start Port = 1025 / End Port = 65535)
- Destination port : A Single Port : (Port : Your UDP port of Shareaza/Emule)

Rule 3

- Action : Allow
- Protocol : TCP or UDP
- Direction : Out
- Description : Rule for outgoing TCP and UDP connections
- Source Address : Any Address
- Destination Address : Any Address
- Source port : A port range : (start port = 1025 / end port = 65535)
- Destination port : A port range : (start port = 1025 / end port = 65535)

Rule 4

- Action : Allow
- Protocol : ICMP
- Direction : Out
- Description : Ping the server (edk network)
- Source Address : Any Address
- Destination Address : Any Address
- ICMP Details : Message : ICMP Echo Request

Rule 5

- Action : Ask (Also select the check box 'Log as a firewall event if this rule is fired')
- Protocol : TCP
- Direction : Out
- Description : Rule for HTTP requests
- Source Address : Any Address
- Destination Address : Any Address
- Source port : A port range : (start port = 1025 / end port = 65535)
- Destination port : Type : Single Port; (Port : 80)

Rule 6

- Action : Block (Also select the check box 'Log as a firewall event if this rule is fired')
- Protocol : IP
- Direction : In/Out
- Description : Block and Log All Unmatching Requests
- Source Address : Any Address
- Destination Address : Any Address
- IP Details : IP Protocol : Any

6. Click 'OK' in the 'Firewall Ruleset' interface.

The new ruleset will be created and added as a Predefined ruleset. Start Shareaza or Emule. When Comodo raises a pop-up alert, choose 'Treat this application as', select the descriptive name you gave for this rule (e.g. For allowing Shareaza/Emule) from the options and select 'Remember my answer'.

To create a 'Predefined Firewall Ruleset' for BitTorrent/Utorrent'

1. Click 'Rulesets' under 'Firewall' from the LHS navigation pane of 'Advanced Settings' interface to open 'Rulesets' panel
2. Click the handle from the bottom of the panel and choose 'Add'

The 'Firewall Ruleset' interface will open for creating a new set of rules.

3. Click the handle from the bottom and choose 'Add'
4. Enter a descriptive name for the new ruleset to be created in the 'Description' text box (for example: For allowing BitTorrent/Utorrent).
5. Now you need to create six rules for the newly created ruleset. To do so, click 'Add'. The 'Firewall Rule' interface will appear. For creating each rule, select the check box and choose the drop-down options under each tab as given below. After creating each rule, click 'OK' for the rule to be added. Click handle in the 'Firewall Ruleset' interface and choose 'Add' to create the next rule.

Rule 1

- Action : Allow
- Protocol : TCP or UDP
- Direction : In
- Description : Rule for incoming TCP and UDP connections
- Source Address : Any Address
- Destination Address : Any Address
- Source port : A Port Range : (Start port = 1025 / End port = 65535)
- Destination port : A Single Port (Port: The port of BitTorrent/Utorrent)

Rule 2

- Action : Allow
- Protocol : TCP
- Direction : Out
- Description : Rule for outgoing TCP connections
- Source Address : Any Address
- Destination Address : Any Address
- Source port : A Port Range : (Start port = 1025 / End port = 65535)
- Destination port : A Port Range : (Start port = 1025 / End port = 65535)

Rule 3

- Action : Allow
- Protocol : UDP
- Direction : Out
- Description : Rule for outgoing UDP connections
- Source Address : Any Address
- Destination Address : Any Address
- Source port : A Single Port: Port: the port of utorrent
- Destination port : A Port Range : (Start port = 1025 / End port = 65535)

Rule 4

- Action : Ask (Also select the check box 'Log as a firewall event if this rule is fired')
- Protocol : TCP
- Direction : Out
- Description : Rule for HTTP requests
- Source Address : Any Address
- Destination Address : Any Address
- Source port : A Port Range : (Start port = 1025 / End port = 65535)

- Destination port : A Single Port (Port = 80)

Rule 5

- Action : Block (Also select the check box 'Log as a firewall event if this rule is fired')
- Protocol : IP
- Direction : In/Out
- Description : Block and Log All Unmatching Requests
- Source Address : Any Address
- Destination Address : Any Address
- IP Details : IP Protocol : Any

6. Click 'OK' in the 'Firewall Ruleset' interface.

The new ruleset will be created and added as a Predefined Firewall ruleset. Start BitTorrent or Utorrent. When Comodo raises a pop-up alert, choose 'Treat this application as', select the descriptive name you gave for this rule (e.g. For allowing BitTorrent/Utorrent) from the options and select 'Remember my answer'.

Block any Downloads of a Specific File Type

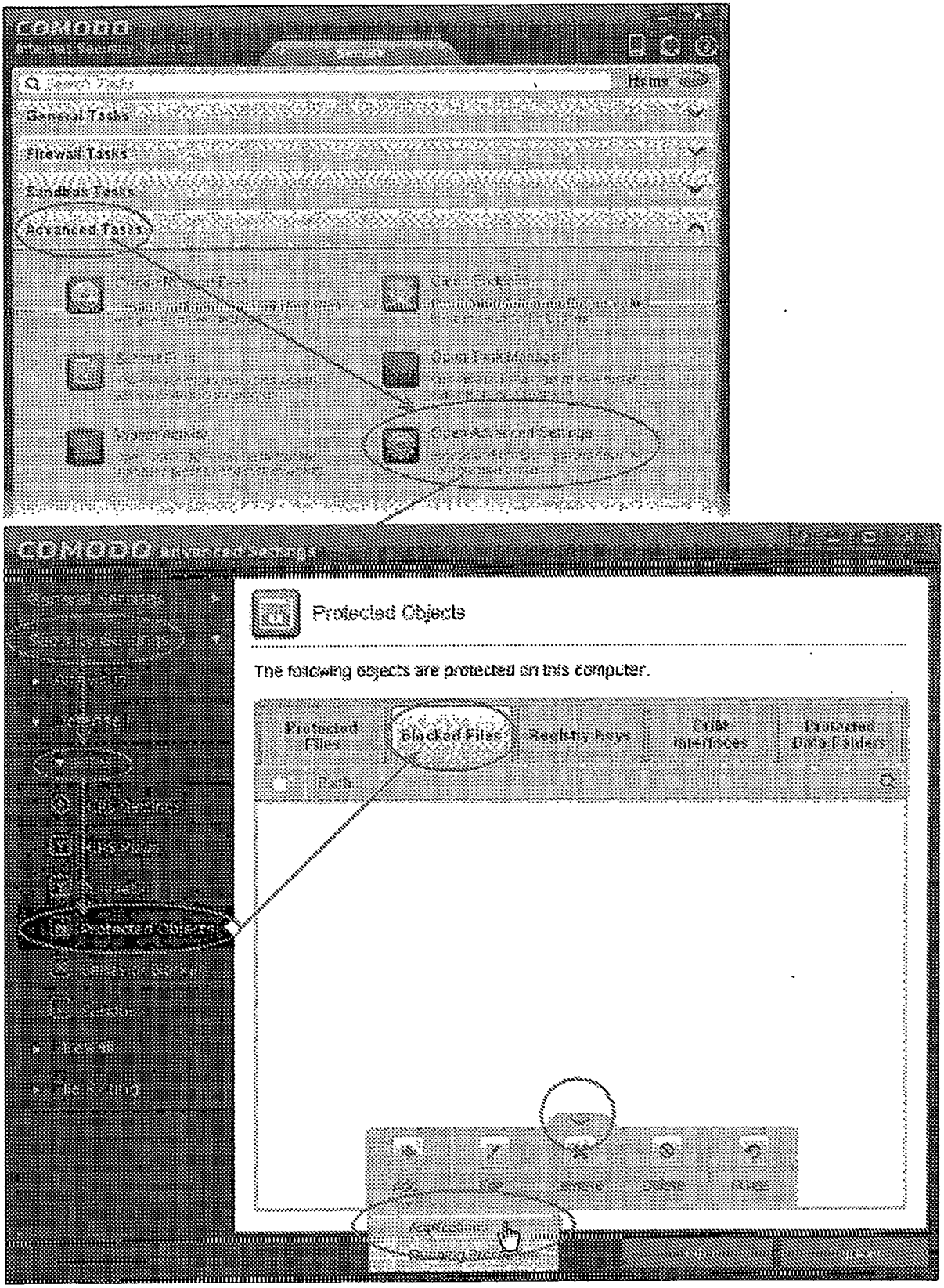
Comodo Internet Security can be configured to block downloads of specific types of file.

Example scenarios:

- Some malicious websites try to push downloads of malware in .exe file format. .exe files are programs which can execute commands on your computer. If the .exe is malicious in intent then these commands could include the installation of key logging programs, initiation of buffer overflow attacks or code to turn your PC into a zombie. For this reason, you may wish to block all downloads of files with a .exe file extension.
- You want to avoid downloading media files like audio files (e.g. files with extensions .wma, .mp3, .wav, .midi), video files (e.g. files with extensions .wmv, .avi, .mpeg, .swf) or image files (e.g. files with extensions .bmp, .jpg, .png) for your disk space restrictions.

To selectively block downloading of specific file type, you need to configure Defense+ component of CIS to block the specific file type from the default download folder of your browser.

1. Open 'Tasks' interface by clicking the green curved arrow at top right of the 'Home' screen
2. Open 'Advanced Tasks' by clicking ' Advanced Tasks' from the Tasks interface and click 'Open Advanced Settings'.
3. Click 'Security Settings' > 'Defense+ ' > 'HIPS' > 'Protected Objects' from the left hand side pane
4. Click 'Blocked Files' tab

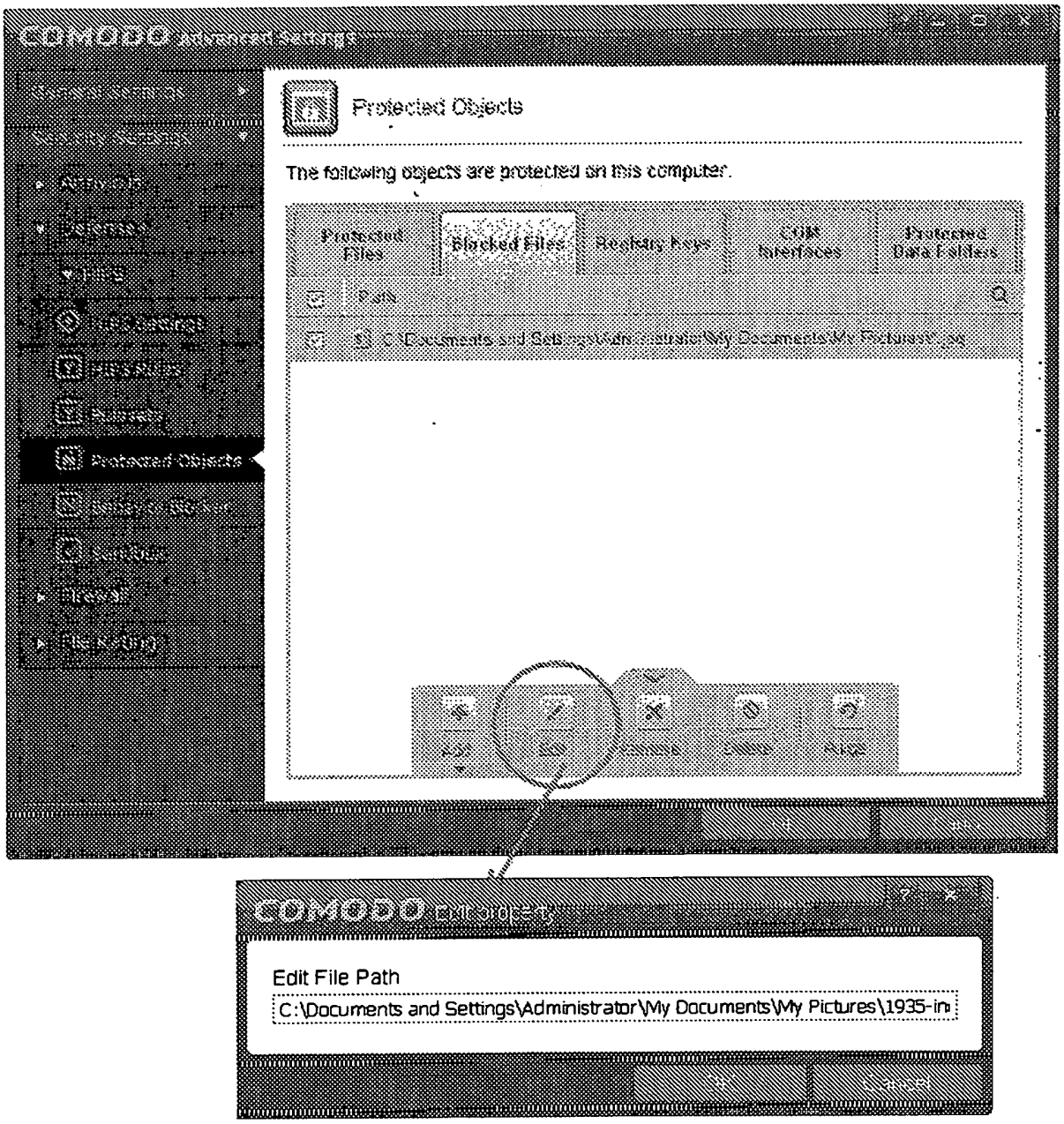


5. Click the handle from the bottom and choose 'Add' > 'Applications'.
6. Browse to the default download folder for that particular file type of your Internet Browser from the Open dialog
- For example, the default download locations for some file types in Internet Explorer are given below:
    - Executable files - C:\Documents and Settings\user name\Local Settings\Temporary Internet Files\

- Document files - C:\Documents and Settings\user name\My Documents\
- Image files - C:\Documents and Settings\user name\My Documents\My Pictures\
- Music files - C:\Documents and Settings\user name\My Documents\My Music\
- Video files - C:\Documents and Settings\user name\My Documents\My Videos\

7. Select file from the folder and click 'Open'

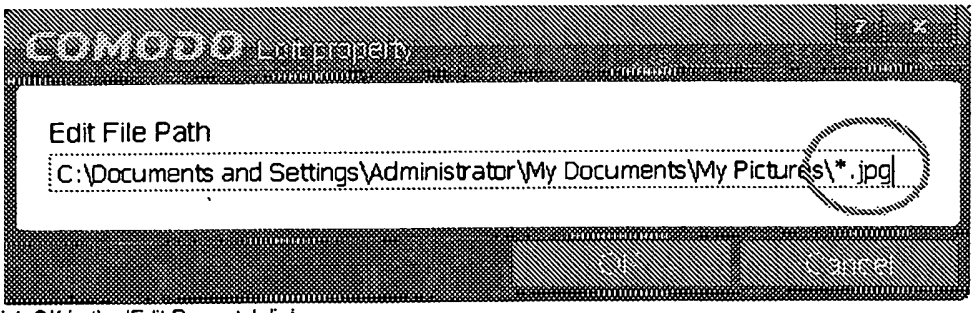
The file will be added to blocked files list.



8. Select the entry from the Blocked Files interface, click the handle from the bottom and choose Edit

The Edit Property dialog will appear.

9. Change the file name at the end of the file path to *.file_extension* (e.g. *.exe, *.jpg)



10. Click OK in the 'Edit Property' dialog
11. Click OK in the Advanced Settings interface to save your settings

The download of the specific file type to the specified folder through the browser will be blocked. If you have more than one browser, repeat the same for the other browsers too.

Note: Blocking files in this way will only block the downloads of the specific file types in the specified folders. If you change the download destination while downloading a file through your web browser, the download will be allowed.

Tip: To unblock the download, Advanced Settings > Defense+ > HIPS > File Protection > Blocked Files, select the file path, click the handle from the bottom and choose 'Remove'.

Disable Behavior Blocker and Auto-Sandboxing on a Per-application Basis

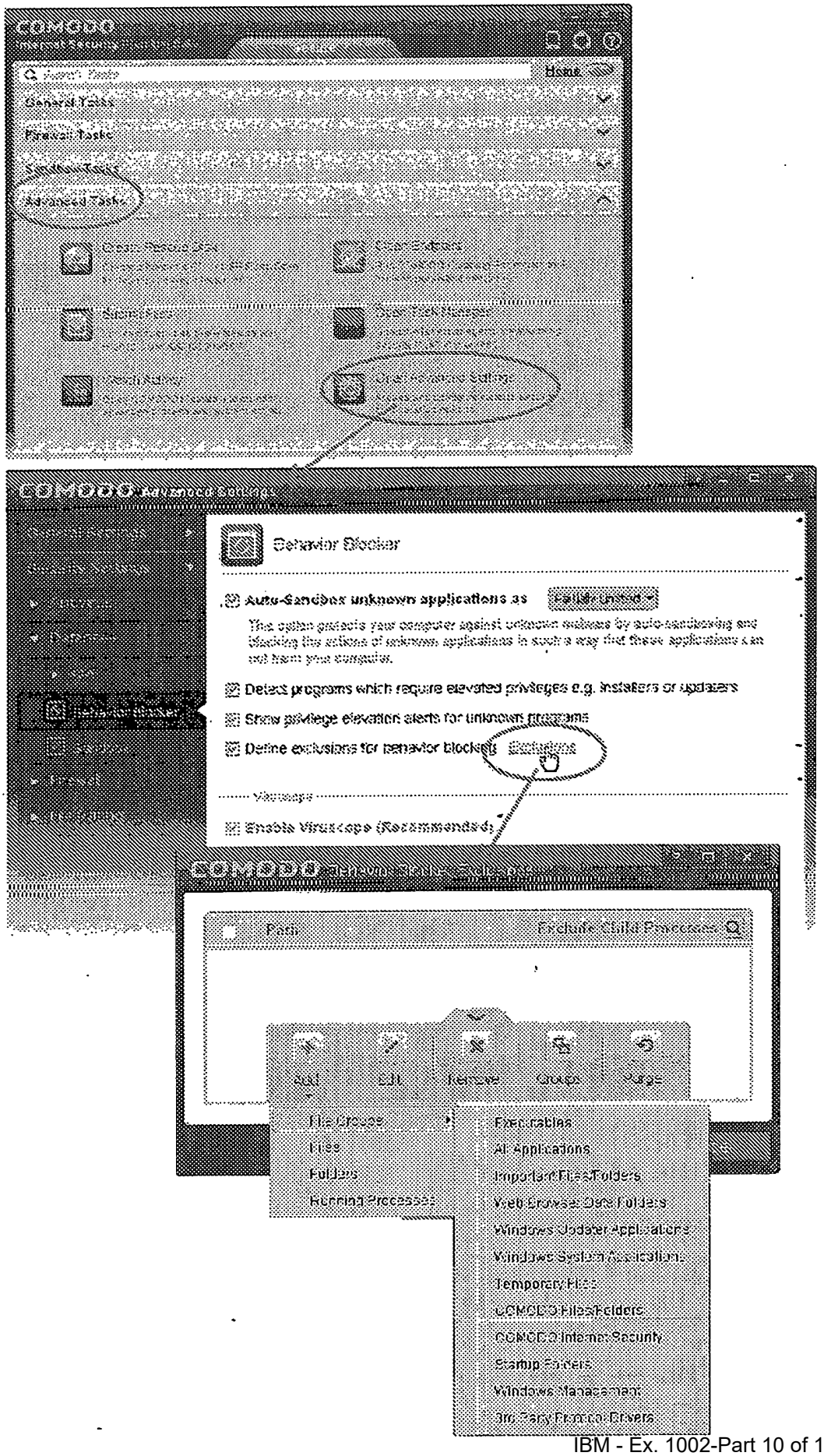
Behavior Blocker will auto-sandbox run unknown executables and queue them for submission to Comodo Cloud scanners for behavior analysis. Users do, however, have the option to exclude specific files or file types from this auto-sandboxing process. This is particularly useful for developers that are creating new applications which, by their nature, are as yet unknown to the Comodo safe list.

To disable the auto-sandboxing selectively

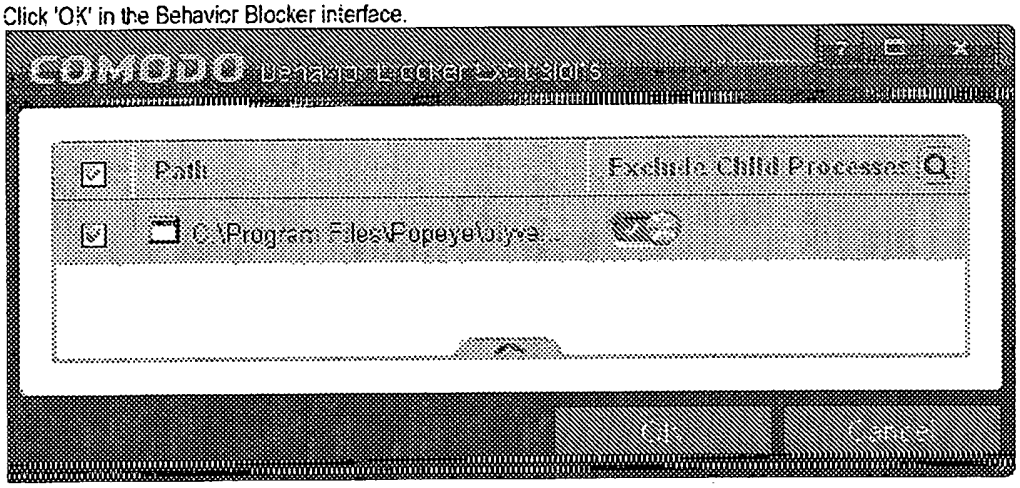
1. Open 'Tasks' interface by clicking the green curved arrow at top right of the 'Home' screen
2. Open Advanced Settings panel by clicking Advanced Tasks > Advanced Settings from the Tasks interface
3. Click 'Security Settings' > 'Defense+' > 'Behavior Blocker' in the left hand side navigation pane of Advanced Settings pane
4. Select the check box 'Define exceptions for behavior blocking and then click on the words 'Exceptions'

The Behavior Blocker Exceptions interface will open.

5. Click the handle from the bottom and choose 'Add' from the options to add the files/executables to be excluded from monitoring by Behavior Blocker and Auto-Sandboxing.
    - You can add items by selecting the required option from the drop-down:
        - **File Groups** - Enables you to select a category of pre-set files or folders. For example, selecting 'Executables' would enable you to create a ruleset for all files with the extensions .exe .dll .sys .ocx .bat .pif .scr .cpl . Other such categories available include 'Windows System Applications' , 'Windows Updater Applications' , 'Start Up Folders' etc. For more details on file groups, refer to the section File Groups.
        - **Files** - Opens the 'Open' window and enables you to navigate to the application or file you wish to add.
        - **Folder** - Opens the 'Browse for Folders' window and enables you to navigate to the folder you wish to add.
        - **Running Processes** - As the name suggests, this option allows you to select an application or executable from the processes that are currently running on your PC.


Home
General Tasks
Firewall Tasks
Advanced Tasks
Behavior Blocker
Auto-Sandbox unknown applications as
Detect programs which require elevated privileges e.g. installers or updaters
Show privilege elevation alerts for unknown programs
Define exclusions for behavior blocker
Exclusions
Enable Virus Logs (Recommended)
Path
Exclude Child Processes
Add
Edit
Remove
Quick
Purge
File Groups
Files
Folders
Running Processes
Executables
All Applications
Important Files/Folders
Web Browser Data Folders
Windows Updater Applications
Windows System Applications
Temporary Files
COMODO Files/Folders
COMODO Internet Security
Startup Folders
Windows Management
3rd Party Program Drivers

6. Click 'OK' in the Behavior Blocker interface.



7. Select whether the child processes of the application also needs to be excluded using the 'Exclude Child Processes' toggle switch.
8. Click OK in the Advanced Settings panel for your settings to take effect.

The selected application will not be auto-sandboxed on its execution.

Alternatively...

1. Add the files to the list of Trusted Files
2. Digitally sign your files with a code signing certificate from a trusted CA then manually add your organization to the Trusted Software Vendors list
3. Disable Behavior Blocker by de-selecting the 'Enable Auto-Sandbox unknown applications as' check box in the Behavior Blocker settings panel. *Not recommended.*

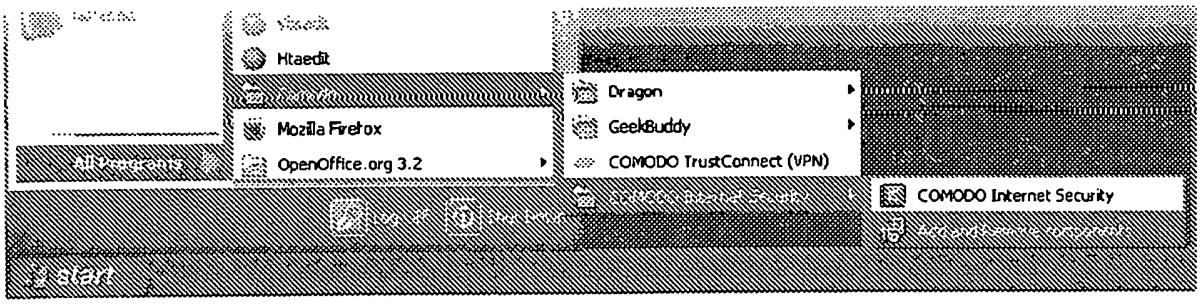
For more details on Auto-Sandboxing process, refer to the section Unknown Files: The Auto-Sandboxing and Scanning Processes.

Switch Between Complete CIS Suite and Individual Components (just AV or FW)

Comodo Internet Security provides the flexibility of installation as a complete security suite or as individual components. You can choose the installation type during the installation itself. Even after the installation, you can switch the installation type without the requirement of uninstallation of the software, retaining your configuration settings. You might have installed the CIS as complete suite initially but may want to retain the AV part or FW part and uninstall the other or vice versa. The inbuilt uninstaller of CIS enables to switch your installation type at any point of time.

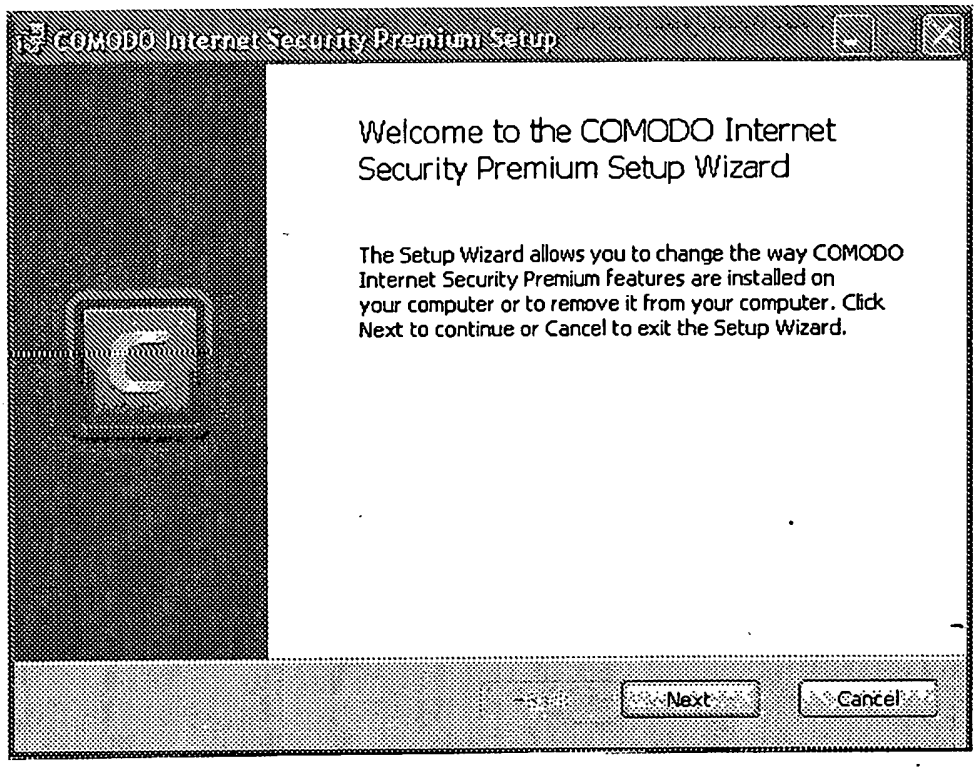
To switch the installation type

1. Click Start > All Programs > COMODO > COMODO Internet Security > Add and Remove Components

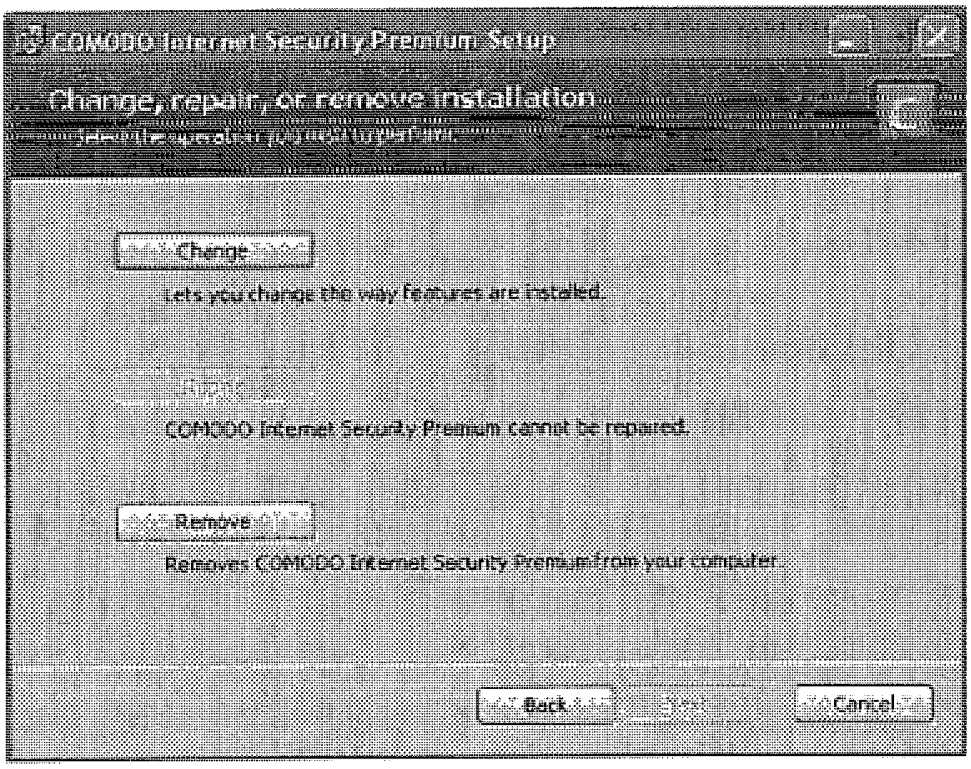


or click Start > Control Panel > Add/Remove Programs > Comodo Internet Security > Change.

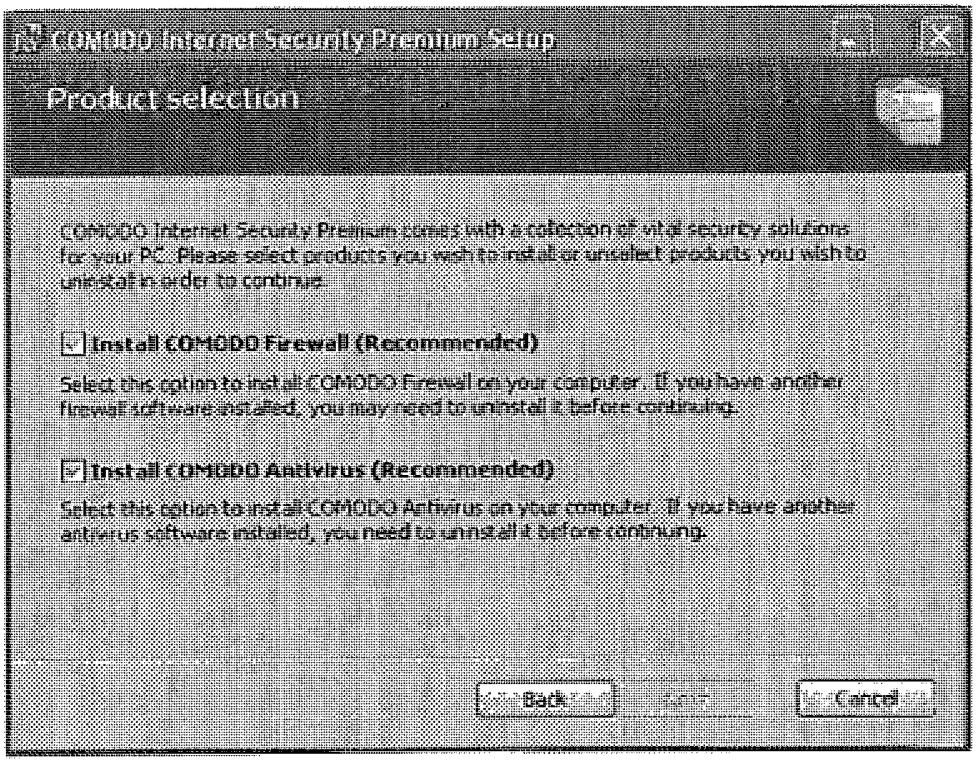
The Configuration Wizard will start.



2. Click 'Next'. The configuration selection screen will appear.



3. Select 'Change' button to change the installed features and click 'Next.' The Product Selection screen will appear.

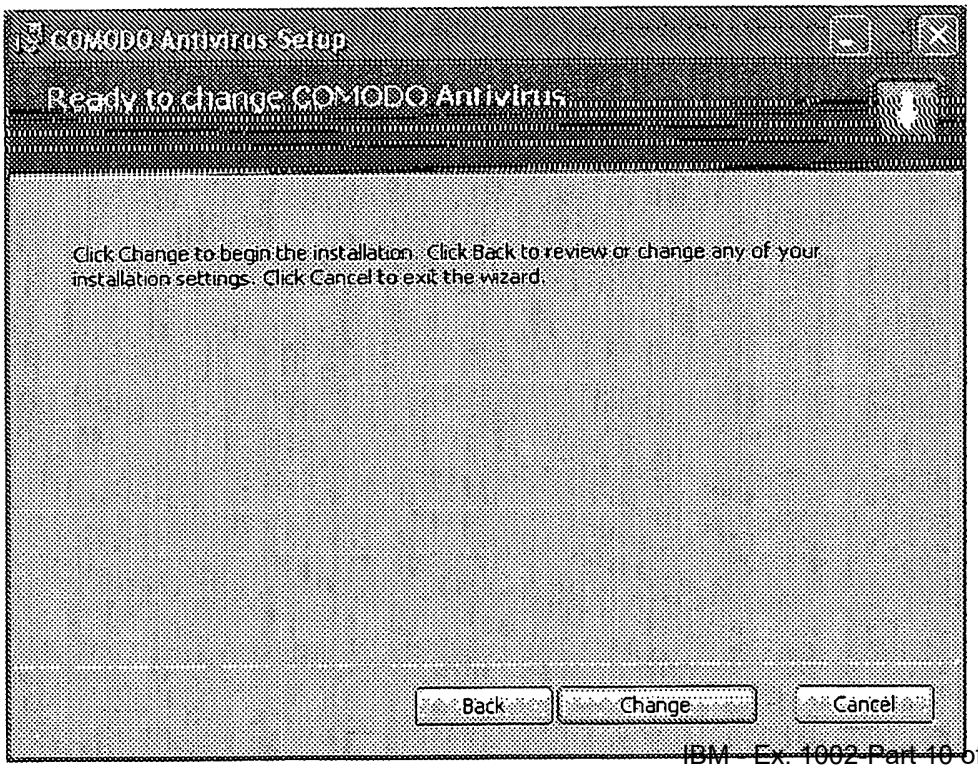


Select the installation type.

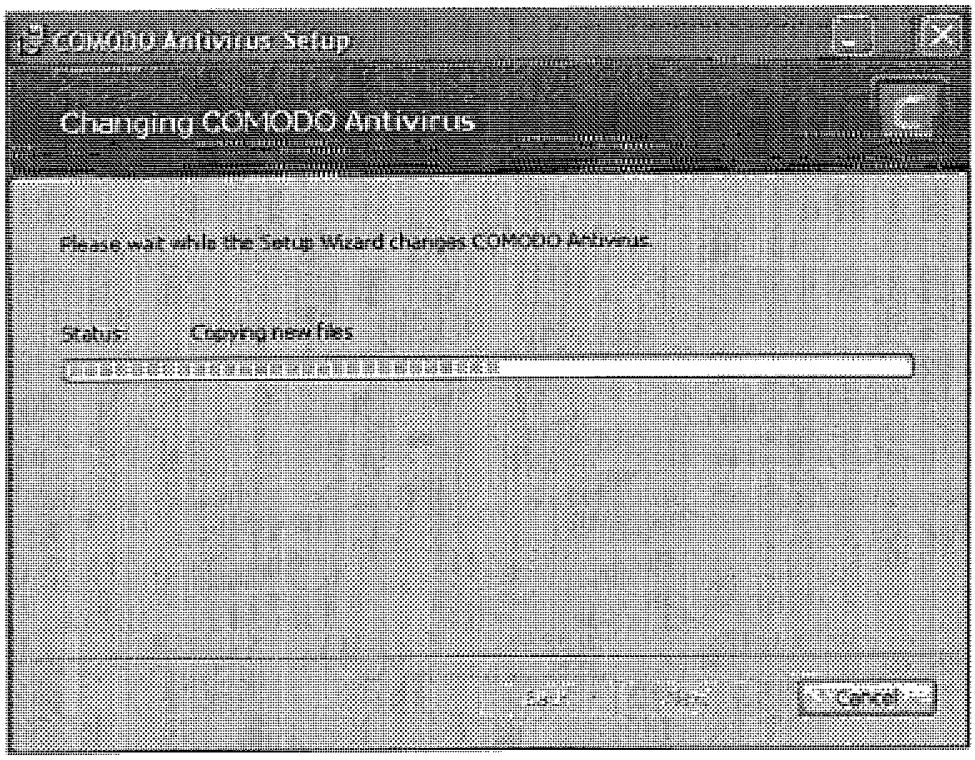
- If you want the complete installation, select both Install COMODO Antivirus and Install COMODO Firewall.
- If you want only the Antivirus part and not the Firewall part, select only the Install COMODO Antivirus and uncheck Install COMODO Firewall.
- If you want only the Firewall part and not the Antivirus part, select only the Install COMODO Firewall and uncheck Install COMODO Antivirus.

Click here for more details on the installation of individual components.

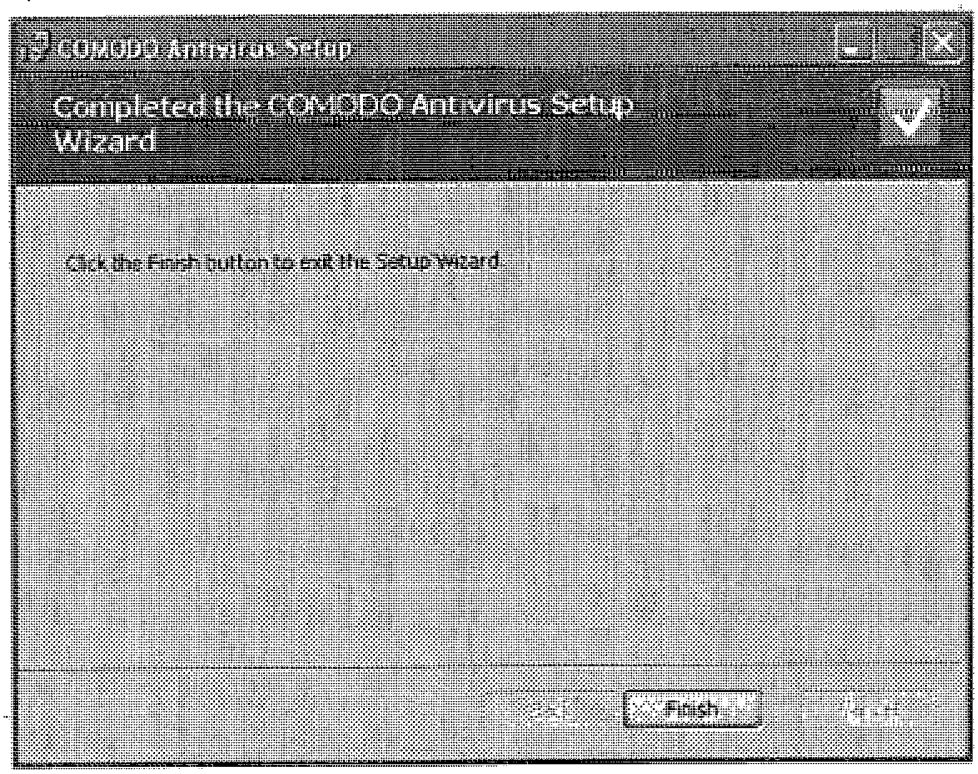
- Click 'Next'. Wait till the CIS is configured and is ready for the change.



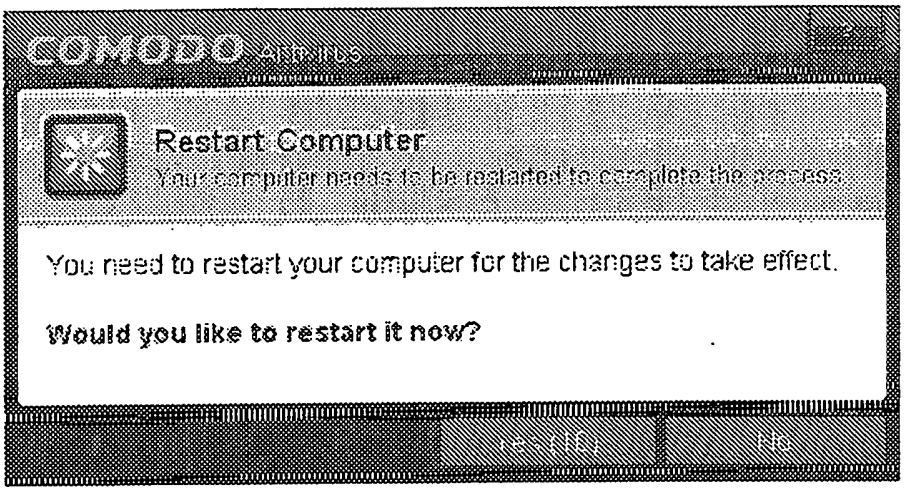
4. Click 'Change' in the next screen. The change progress will be indicated...



... and on completion, Click the Finish button to exit the wizard.



In order for the configuration change to take effect, your computer needs to be restarted. A 'Restart Computer' dialog with a countdown timer will be displayed.



- If you want to restart the system at a later time, click 'No'.
- If you want to restart the system immediately, please save any unsaved data and click 'Yes' or leave the dialog for the system to restart at the end of countdown timer.

Note: The change will take effect only on the next restart of the computer.

Switch Off Automatic Antivirus and Software Updates

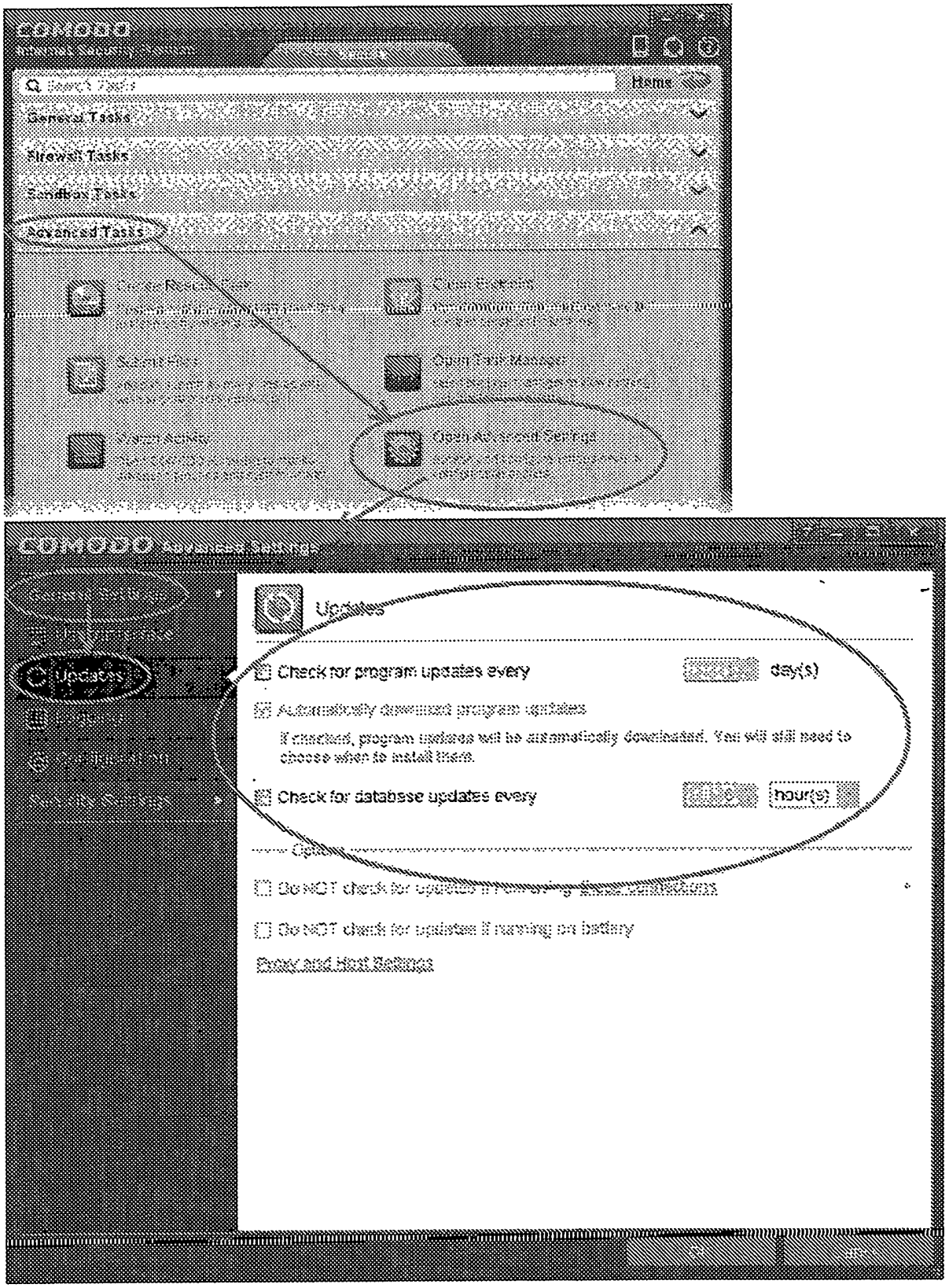
By default, Comodo Internet Security will automatically check for software and Antivirus database updates. However, some users like to have control over what gets downloaded and when it gets downloaded. For example, network administrators may not wish to automatically download because it will take up to much bandwidth during the day. Similarly, users that have particularly heavy traffic loads may not want automatic updates because they conflict with their other download/upload activity.

CIS provides full control over virus and software updates. Click the appropriate link below to find out more:

- Switch off automatic software and virus signature database updates entirely
- Switch off automatic software and virus signature database selectively
- Switch off automatic virus signature database updates prior to Antivirus Scans

To switch off automatic updates entirely:

1. Open 'Tasks' interface by clicking the green curved arrow at top right of the 'Home' screen
2. Open Advanced Settings panel by clicking Advanced Tasks > Advanced Settings from the Tasks interface
3. Click 'Updates' under 'General Settings' from the left hand side navigation pane
4. Deselect the check boxes 'Check for program updates every xxx day(s)' and 'Check for database updates every xxx hour(s)'

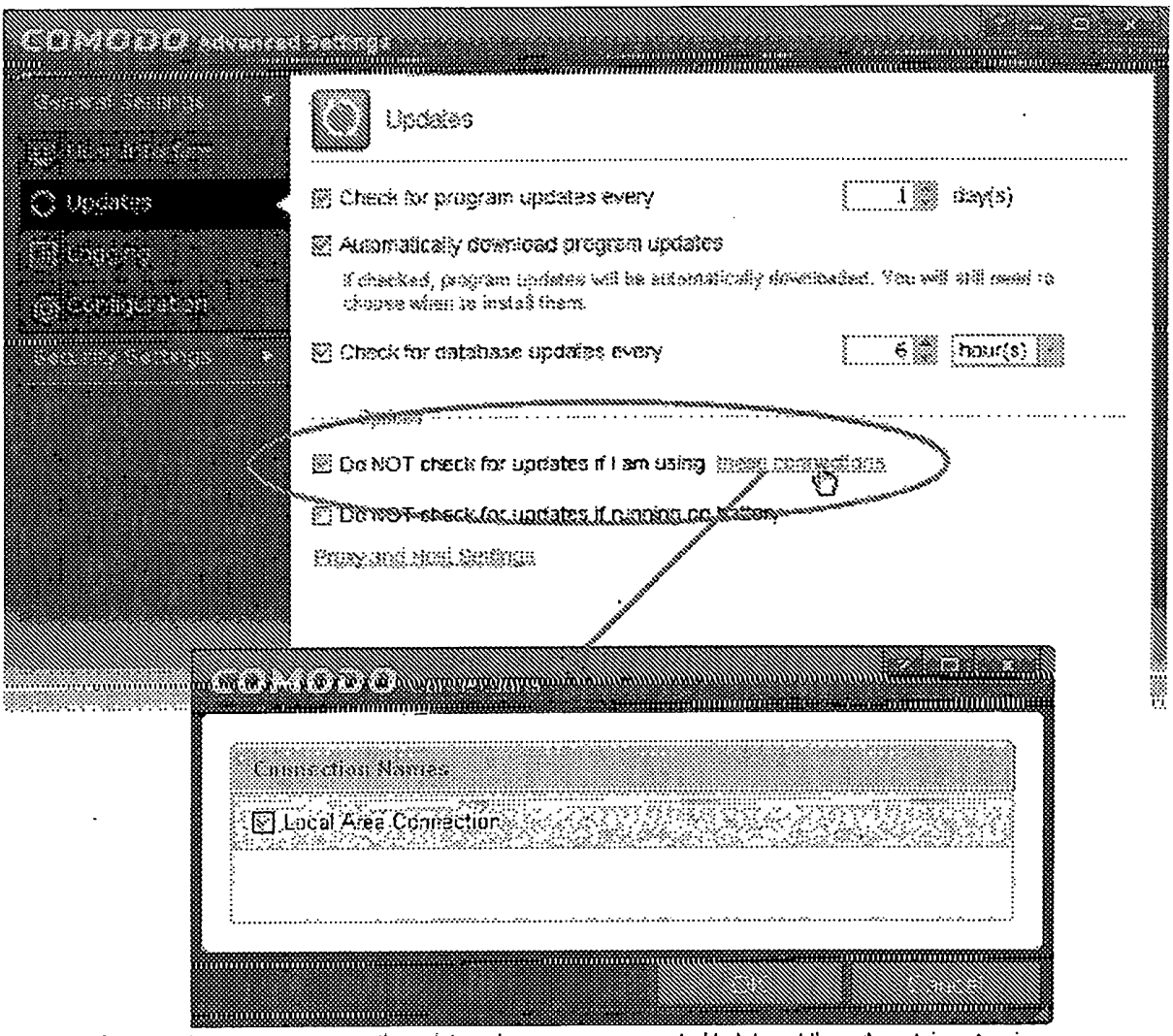


5. Click 'OK' in the Advanced Settings panel.

To switch off automatic updates selectively:

1. Open 'Tasks' interface by clicking the green curved arrow at top right of the 'Home' screen
2. Open Advanced Settings panel by clicking Advanced Tasks > Advanced Settings from the Tasks interface

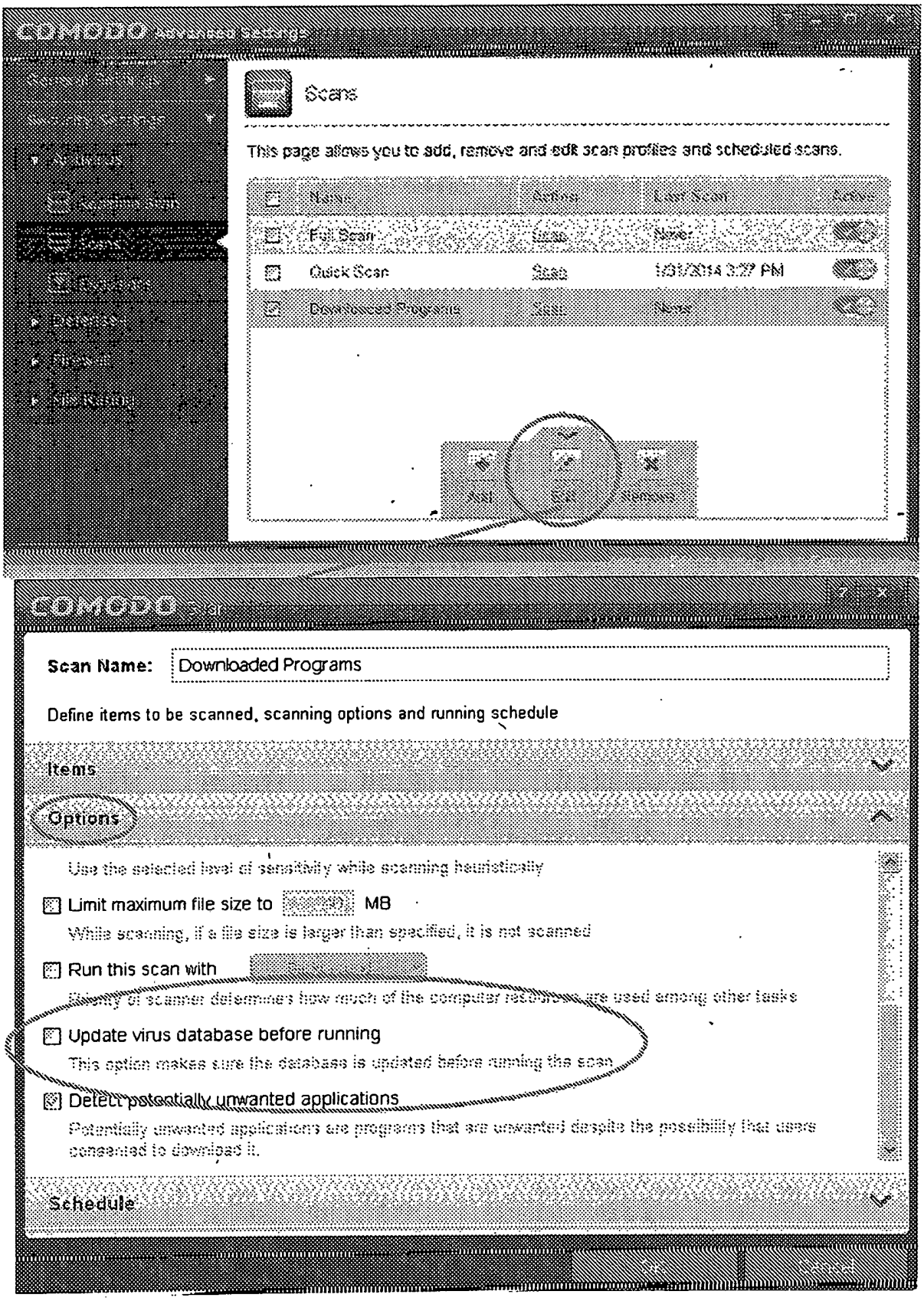
3. Click 'Updates' under 'General Settings' from the left hand side navigation pane



- If you want to suppress automatic updates when you are connected to Internet through certain networks
  - Select the 'Do NOT check updates if am using these connections' check-box
  - Then click the 'these connections'. The 'Connections' dialog will appear with the list of connections you use.
  - Select the connection through which you do not want CIS to check for updates and click OK.
- If you want to suppress automatic updates when your computer is running on battery
  - Select the 'Do NOT check for updates if running on battery' checkbox

To switch off automatic virus signature database updates prior to AV Scans:

1. Open 'Tasks' interface by clicking the green curved arrow at top right of the 'Home' screen
2. Open Advanced Settings panel by clicking Advanced Tasks > Advanced Settings from the Tasks interface
3. Click 'Security Settings' > Antivirus' > 'Scans'. A list defined scan profiles will be displayed.
4. Select the scan profile for which you do want the automatic virus database updates prior to the scan



5. Click the handle from the bottom and select 'Edit'
6. Click 'Options' to open the Options pane and deselect 'Update virus database before running' checkbox

7. Click 'OK' on the 'Scan' interface.
8. Click 'OK' in the 'Advanced Settings' interface for your changes to take effect.

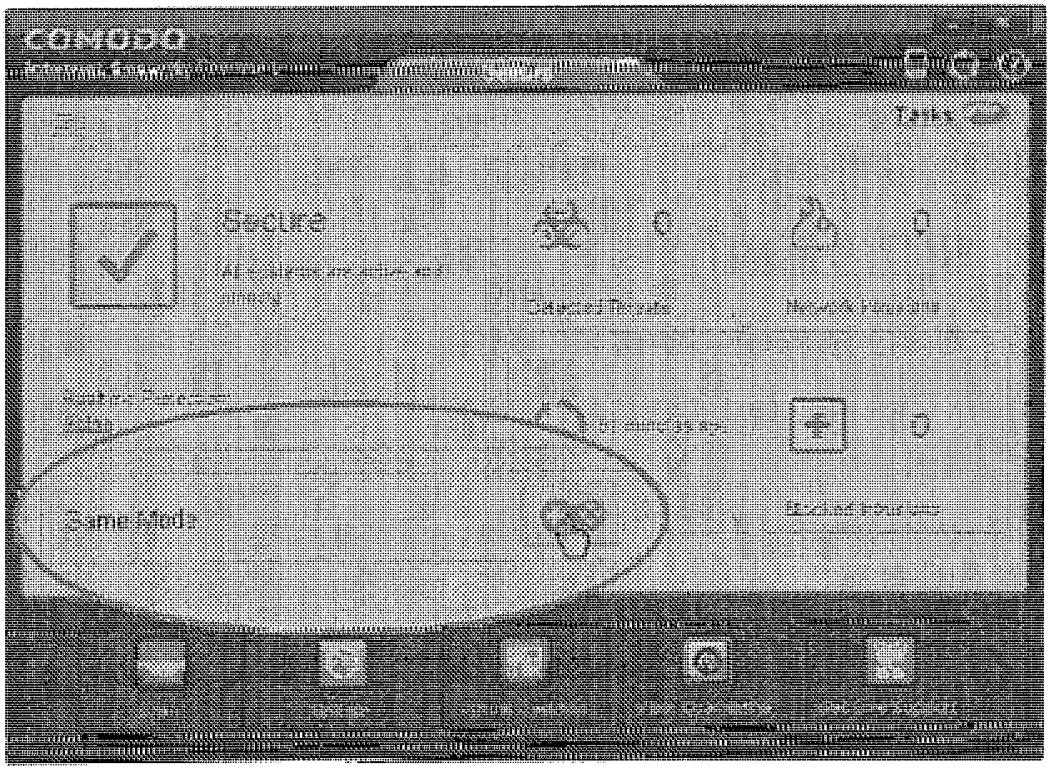
Suppress CIS Alerts Temporarily while Playing Games

Because of continuous monitoring of all your system activities in granular level for implementing Default-Deny Protection, Comodo Internet Security generates pop-up alerts whenever it identifies any event appearing to be a malicious activity or execution of programs that require privileges like Internet access and file access rights. Each alert provides information and options that enable you to make an informed decision on whether you want to allow or block a request or activity. Alerts also to allow you to instruct Comodo Internet Security on how it should behave in future when it encounters activities of the same type.

But at times when you are involved in activities like playing computer games, where you do not want to be interfered with such alerts, you can temporarily stop them from being displayed. require undisturbed environment. During this time, the operations that can interfere with users' gaming experience are either suppressed or postponed.

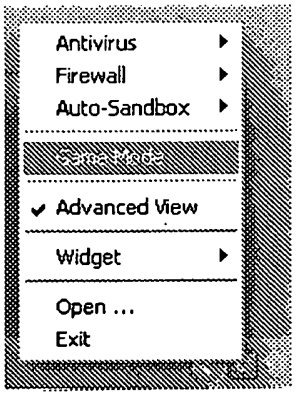
To temporarily stop pop-up alerts

- Click 'Game Mode' button from CIS Home screen



or

- Right click on the CIS System Tray icon and select 'Game Mode' from the options.

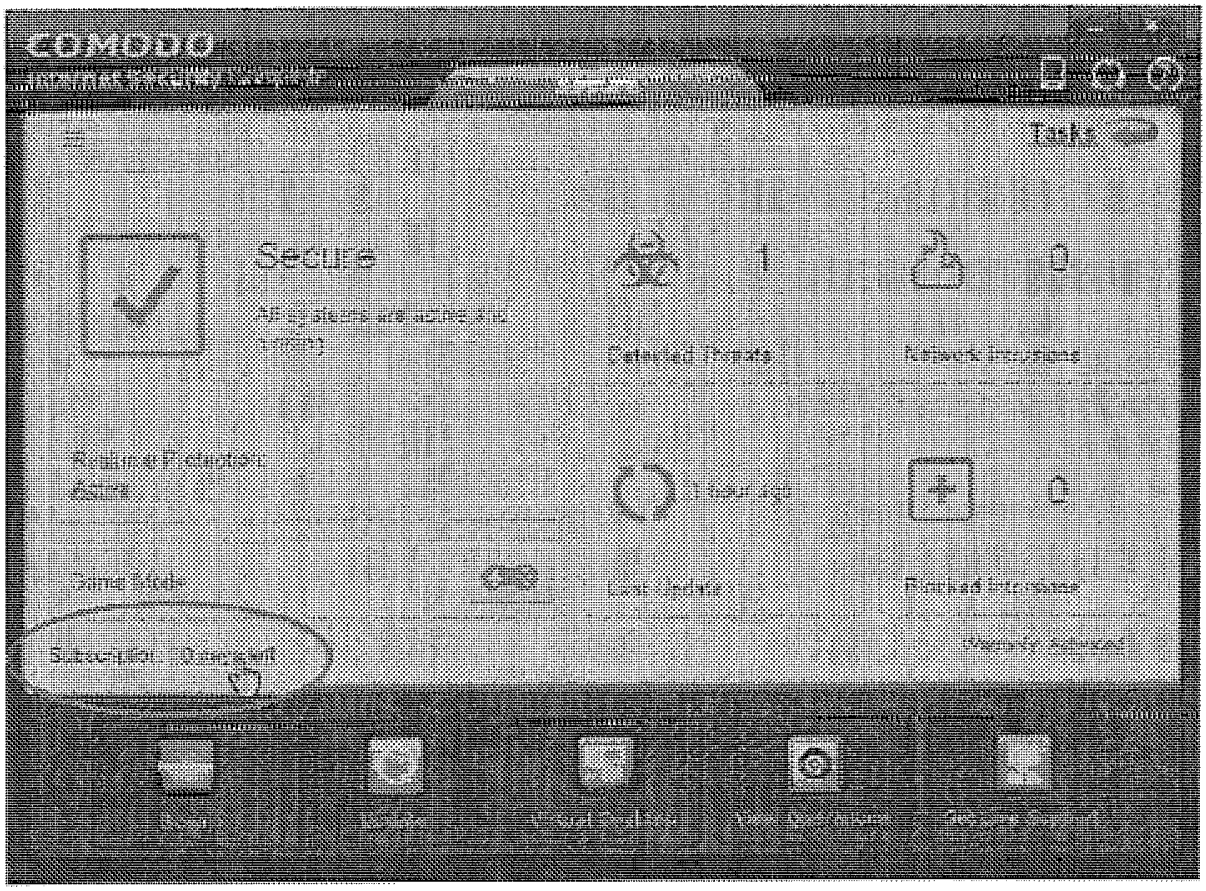


The alerts are now suppressed. To resume alerts and scheduled scans, just de-activate Game Mode from the home screen or the system tray icon right click options.

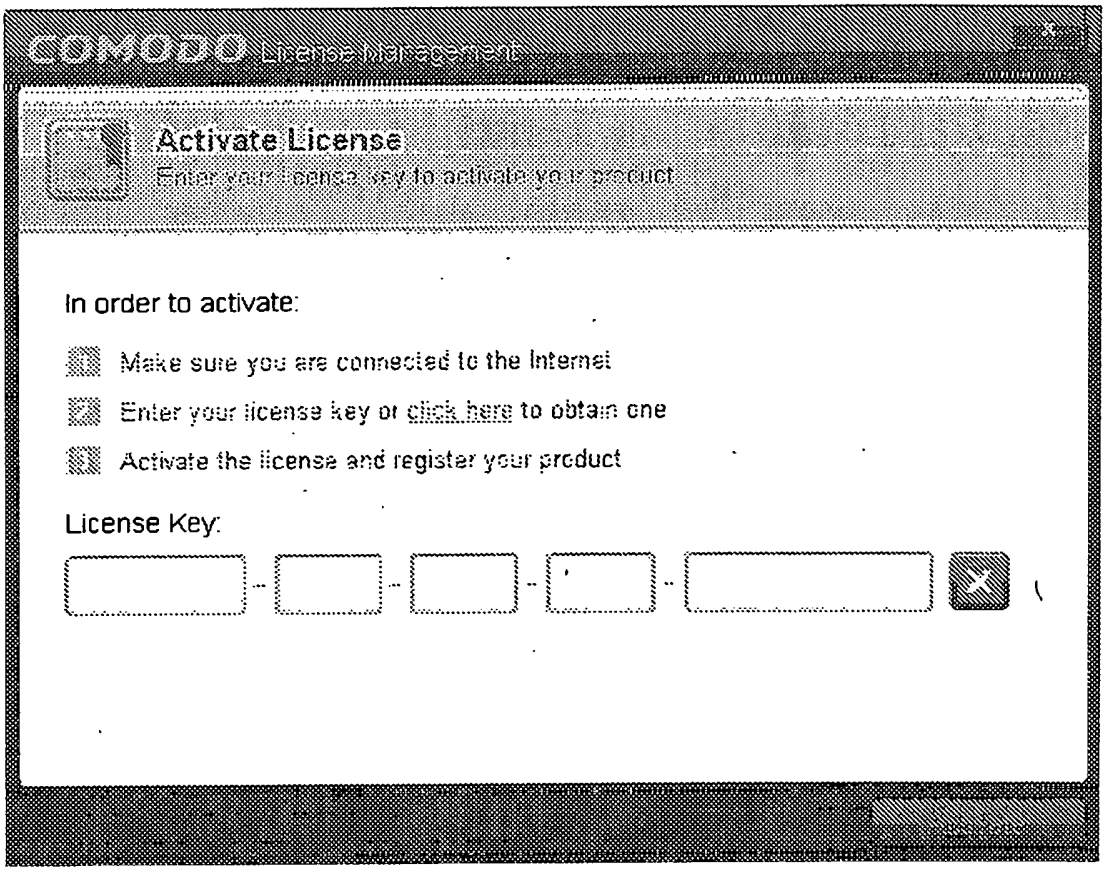
Renew or Upgrade your License

In order to enjoy continued protection from Comodo Internet Security, you will need to renew your license when it is due to expire.

To renew or upgrade your license, click the 'Activate Now' link beside 'Subscription' on the CIS home screen (alternatively, click 'No. of days left')



The Product Activation Wizard will start.



- Click the 'click here' link. You will be taken to the purchase page at https://secure.comodo.com/home/purchase.php?afi=Comodo&rs=7&pid=9&cid=RkJEMUZENjMzQUM4RDIDNDE4MzBDQjc1NDIENUIzRkY&lid=&
- Select your CIS Package.
- Select 'Existing Comodo User' checkbox in 'Enter Customer Details' area, enter your login and password and complete the payment procedure.
- The License key will be sent to you by email. Enter the license key and click the 'Activate' button.
- After successful validation, your subscription will be activated and a confirmation screen will be displayed.

If you are renewing a license for the same CIS product then entering the license key will upgrade the license without requiring re-installation. If you are upgrading license types, then installation of the new product type will begin automatically. You may need to restart your computer to finalize the upgrade.

If you are using any of the trial versions of CIS, you have to purchase the license at the end of trial period in order to continue using the product. An alert will be displayed after the expiry of trial period.

- Click the the 'Renew Now' button in the alert screen and follow the same purchase and activation procedure explained above.

How to use CIS Protocol Handlers

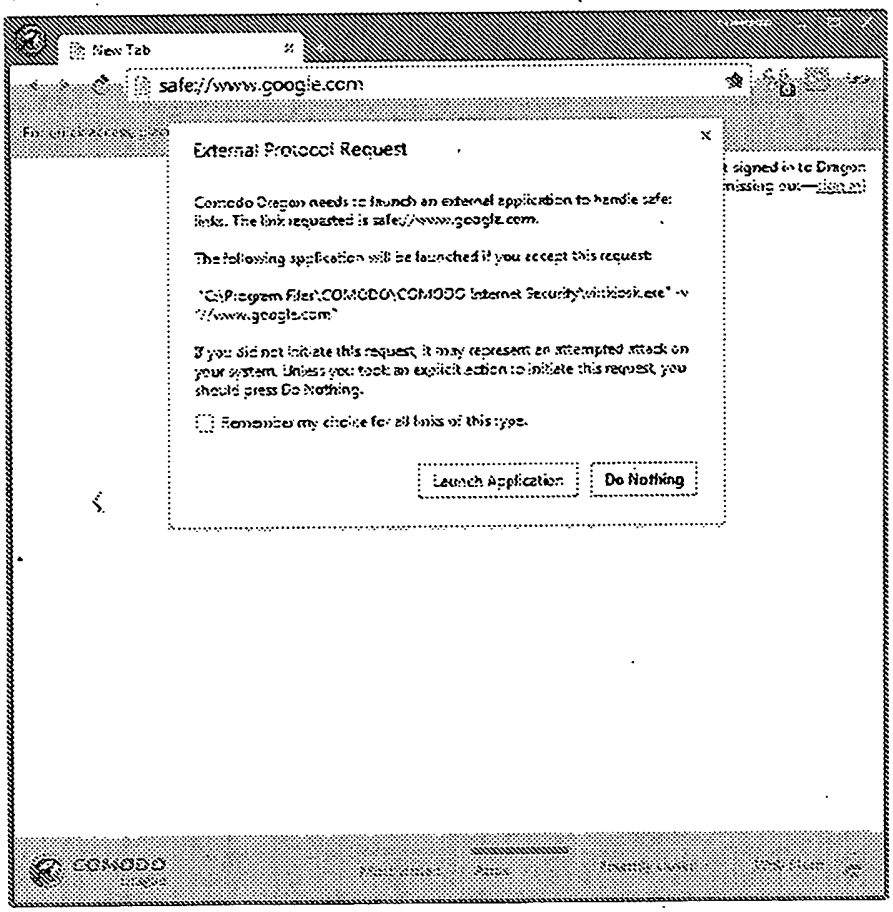
COMODO Internet Security has its own protocol handlers that allow you to perform certain tasks from a web page. This includes tasks like opening a web page from a sandboxed browser, or starting a virus database update etc. CIS 2013 supports several protocol handlers listed below.

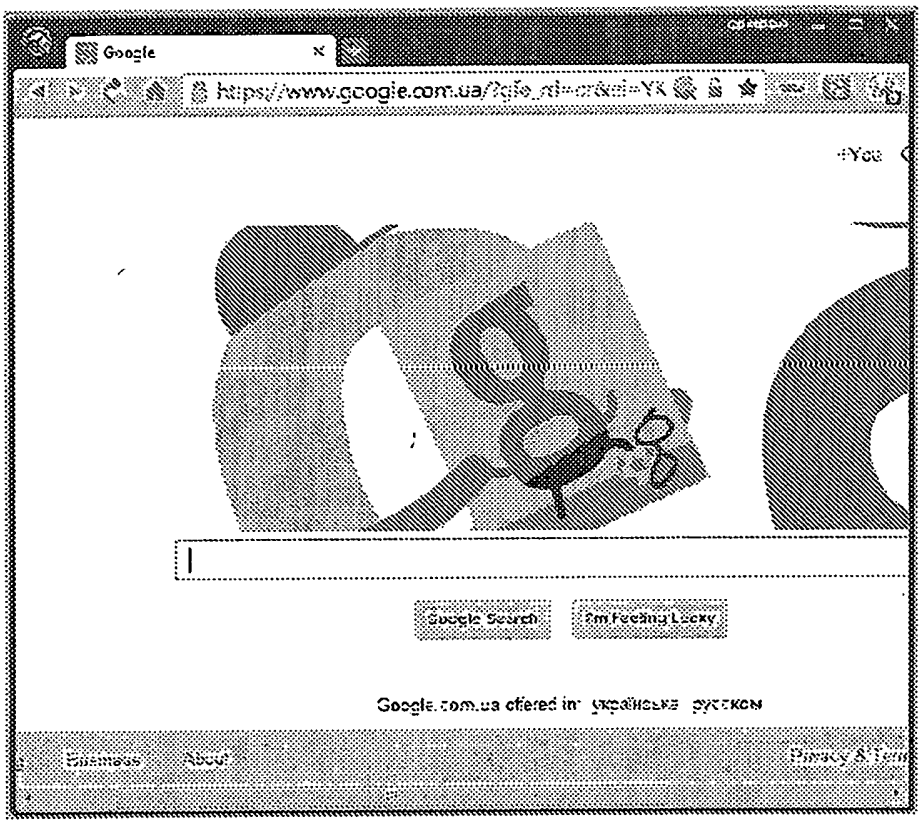
1 – safe://

This protocol is used to open any URL with a sandboxed browser.

For example: Try safe://www.google.com

The URL will be open in a sandboxed browser. Note the green border:

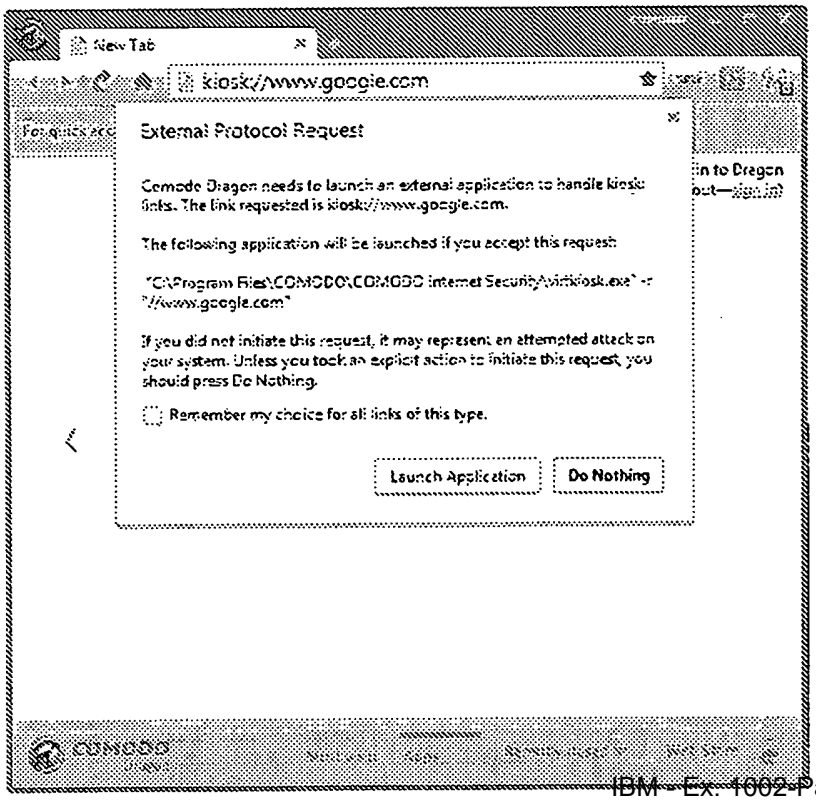




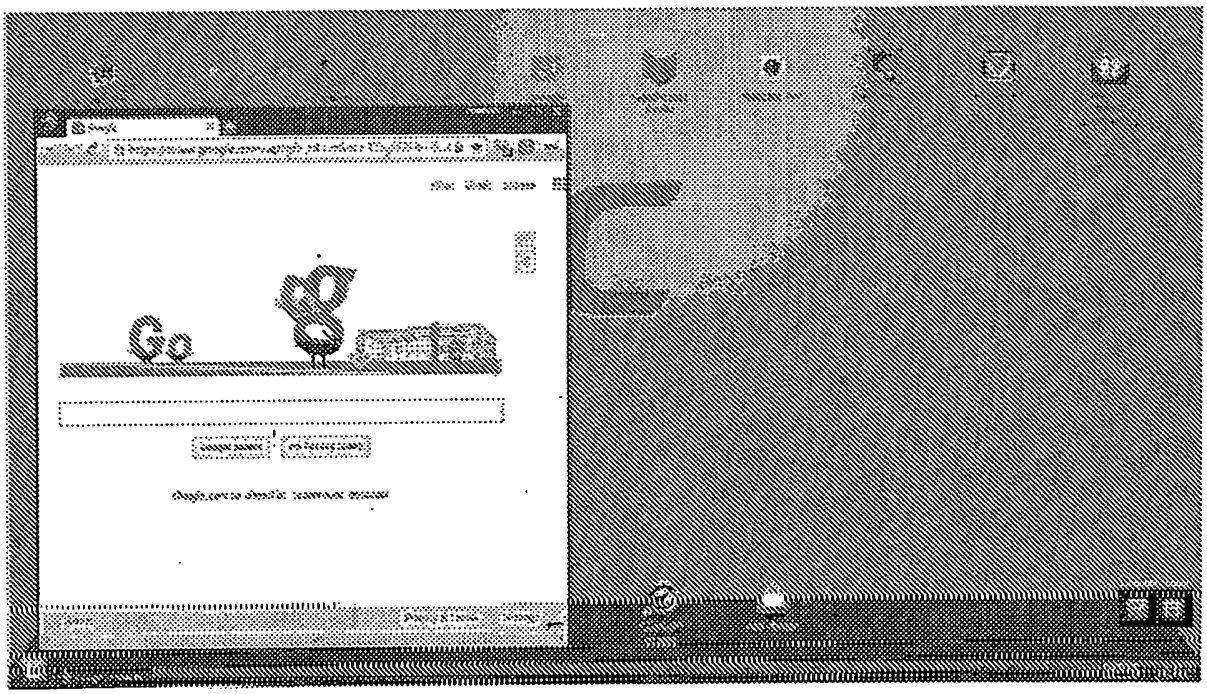
2 – kiosk://

Like the previous one, opens the URL from COMODO Virtual Kiosk.

E.g. Try kiosk://www.google.com



URL will display in a browser from the kiosk:



Appendix 2 - Comodo Secure DNS Service

Introduction

Comodo Secure DNS service replaces your existing Recursive DNS Servers and resolves all your DNS requests exclusively through Comodo's proprietary Directory Services Platform. Most of the networks use recursive DNS services that are provided by their ISP or that reside on their own set of small DNS servers but it becomes essential to have a secure and broadly distributed DNS service to have a faster and safe DNS resolution.

Background Note: Every device on the Internet is uniquely identified by a 32-bit number (IPv4) or a 128-bit number (Ipv6). While this is perfectly satisfactory for computers, humans are far more comfortable remembering names rather than a string of numbers. The Domain Name System (DNS) provides the translation between those names and numbers. Virtually every piece of software, device, and service on the Internet utilizes DNS to communicate with one another. DNS also makes this information available across the entire span of the Internet, allowing users to find information remotely.

Comodo Secure DNS is a broadly distributed Recursive DNS service that gives you full control to determine how your clients interact with the Internet. It requires no hardware or software and provides reliable, faster, smarter and safer Internet experience.

- Reliable - Comodo Secure DNS Directory Services Platform currently spans across five continents around the world. This allows us to offer you the most reliable fully redundant DNS service anywhere. Each node has multiple servers, and is connected by several Tier 1 carriers to the Internet.
- Faster - Our strategically placed nodes are located at the most optimal intersections of the Internet. Unlike most DNS providers, Comodo Secure DNS Directory Services Platform uses Anycast routing technology - which means that no matter where you are located in the world, your DNS requests are answered by the closest available Comodo Secure DNS set of servers. Combine this with our huge cache and we can get the answers you seek faster and more reliably than anyone else. Furthermore, our "name cache invalidation" solution signals the Comodo Secure DNS recursive servers anytime one of our authoritative customers or partners updates a DNS record, fundamentally eliminating the concept of a TTL.
- Smarter - Comodo's highly structured search and guide pages get you where you want to be, when you inadvertently attempt to go to a site that doesn't exist.
- Safer - As a leading provider of computer security solutions, Comodo is keenly aware of the dangers that plague the Internet today. Secure DNS helps users keep safe online with its malware domain filtering feature. Secure DNS references a real-time block list (RBL) of harmful websites (i.e. phishing sites, malware sites, spyware sites, excessive advertising sites, etc.) and will warn you whenever you attempt to access a site containing potentially threatening content. Additionally, our 'name cache invalidation' solution signals the Comodo Secure DNS recursive servers whenever a DNS record is updated - fundamentally eliminating the concept of a TTL. Directing your requests through highly secure servers can also reduce your exposure to the DNS Cache Poisoning attacks that may affect everybody else using your ISP.

To start Comodo Secure DNS service the DNS settings of your computer has to be modified to point to our server's IP addresses. Comodo Internet Security automatically modifies the DNS settings of your system during its installation to get the services. You can also modify the DNS settings of your system manually, if you haven't selected the option during installation. You can also revert to the previous settings if you want, at anytime.

Click the following links to get the instructions for manually modifying the DNS settings on your router or on your computer.

- Router
- Windows XP
- Windows 7/ Windows Vista

Router - Manually Enabling or Disabling Comodo Secure DNS Service

You can manually enable or disable Comodo Secure DNS service in your Router by modifying the DNS settings accessible

through DNS Server settings of your router. Comodo recommends making the change on your router so that with one change, all the computers on your network can benefit from Comodo Secure DNS.

To enable the Comodo Secure DNS service, modify the DNS server IP address settings to Comodo Secure DNS server IP addresses. The IP address are:

Primary DNS : 8.26.56.26

Secondary DNS : 8.20.247.20

To stop Comodo Secure DNS service

- Modify the DNS server IP address to your previous settings.

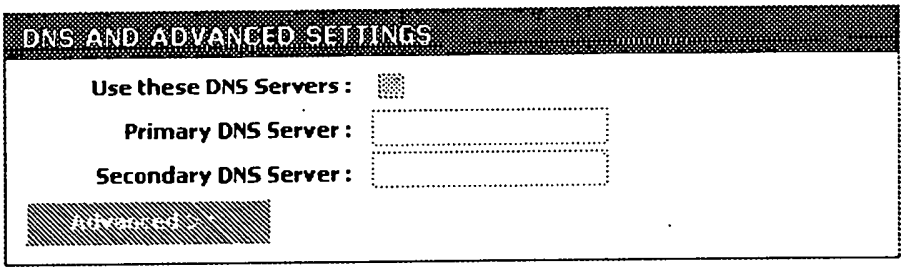
To modify the DNS settings

1. Login to your router. To log in and configure your router, you can open it up in your web browser. If you don't know the IP address for your router, don't worry, it is typically one of the following:

   http://192.168.0.1
   http://192.168.1.1
   http://192.168.10.1

   If you have forgotten your router's username and/or password, the most common username is "admin" and the password is either blank, "admin", or "password". If none of those work, you can often reset the password to the manufacturer default by pressing a button on the router itself, or in some cases access without a password if you try to access your router quickly after you've cycled the power to it.

2. Find the DNS Server Settings. Look for "DNS" next to a field which allows two or three sets of numbers (these fields may be empty).

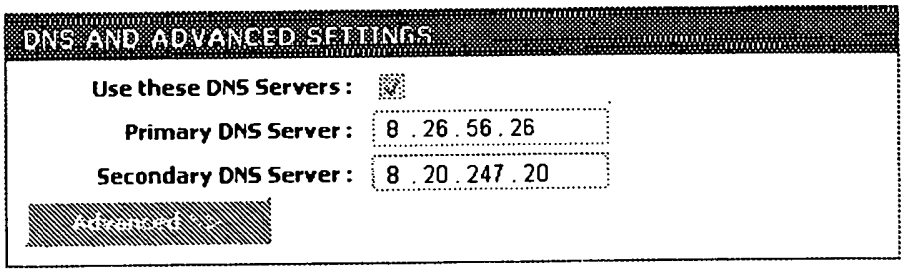


3. Select the check box Use these DNS Servers, type the Comodo Secure DNS Server settings as your DNS server settings and click 'Save'/'Apply'.

   Primary DNS server address for Comodo Secure DNS is: 8.26.56.26

   Secondary DNS server address for Comodo Secure DNS is: 8.20.247.20

   When you are done, the above example would look like this.



You can disable Comodo Secure DNS by:

- Deselecting the check box 'Use these DNS servers' address automatically'. This means that you use the DNS server provided by your ISP. This is the option that most home users should choose if they wish to disable the service.

or

- Entering different preferred and alternate DNS server IP addresses.

Windows XP - Manually Enabling or Disabling Comodo Secure DNS Service

You can manually enable or disable Comodo Secure DNS service in your Windows XP computer by modifying the DNS settings accessible through Control Panel > Network Connections.

To enable the Comodo Secure DNS service, modify the DNS server IP address settings to Comodo Secure DNS server IP addresses. The IP address are:

Preferred DNS : 8.26.56.26

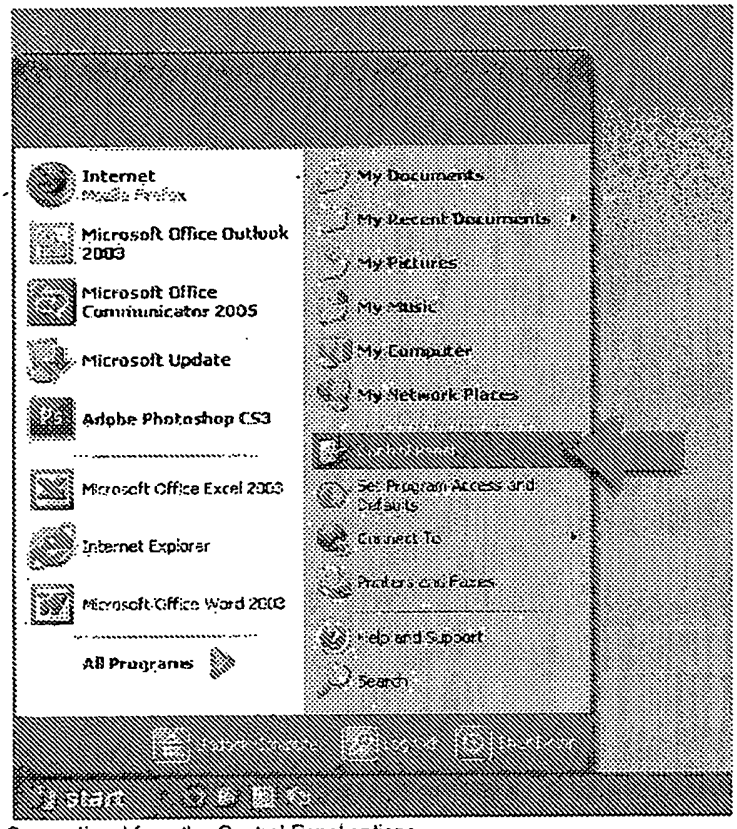
Alternate DNS : 8.20.247.20

To stop Comodo Secure DNS service

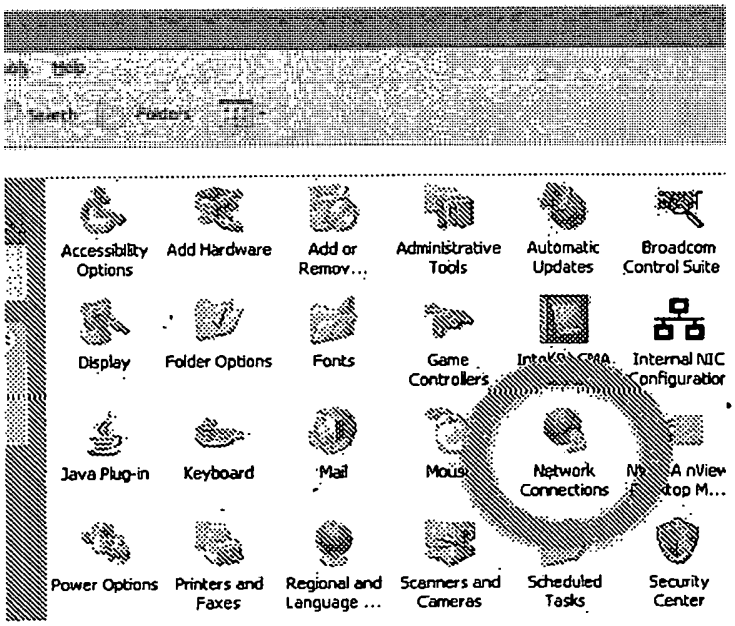
- Modify the DNS server IP address to your previous settings.

To modify the DNS settings

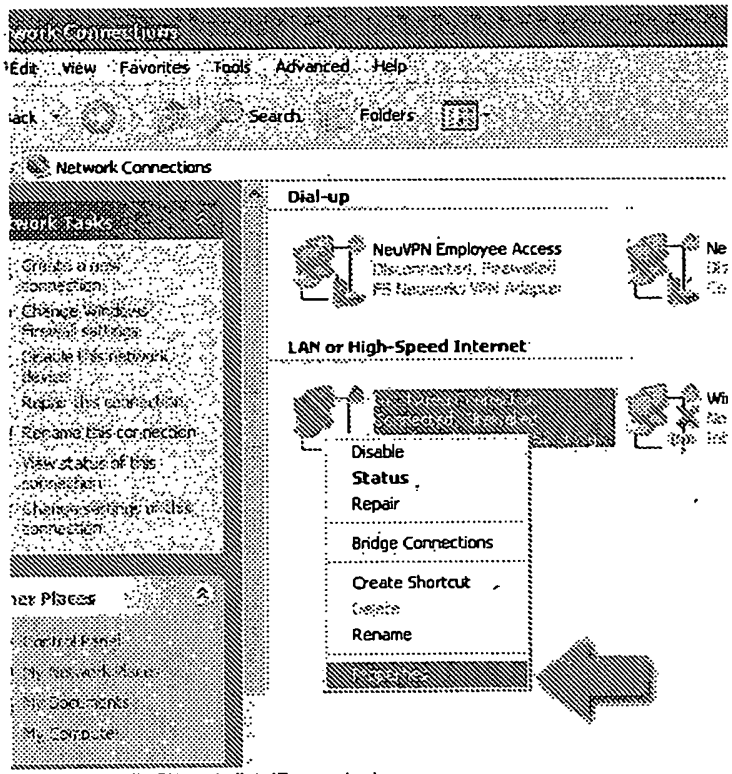
1. Select the 'Control Panel' from the Start Menu.



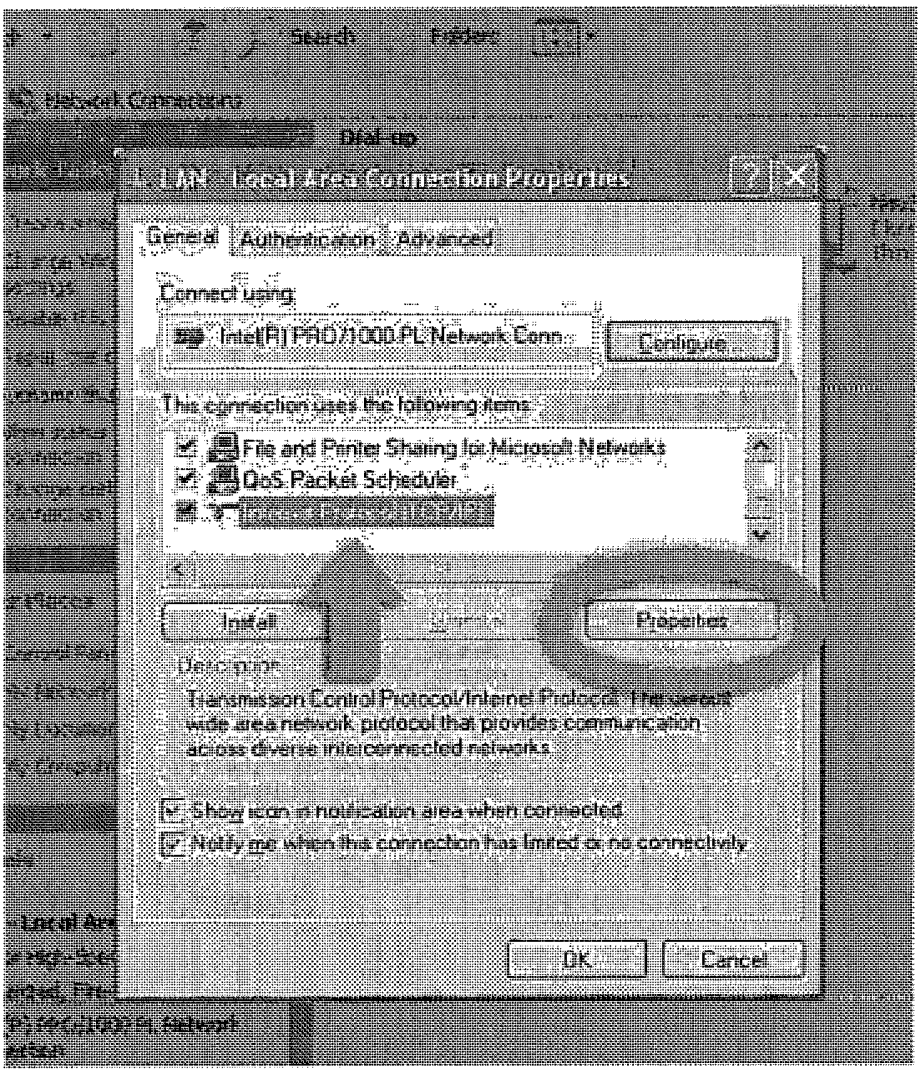
2. Click 'Network Connections' from the Control Panel options.



3. Right click on your connection from the Network Connections window and click 'Properties'.



4. Select 'Internet Protocol (TCP/IP)' and click 'Properties'.

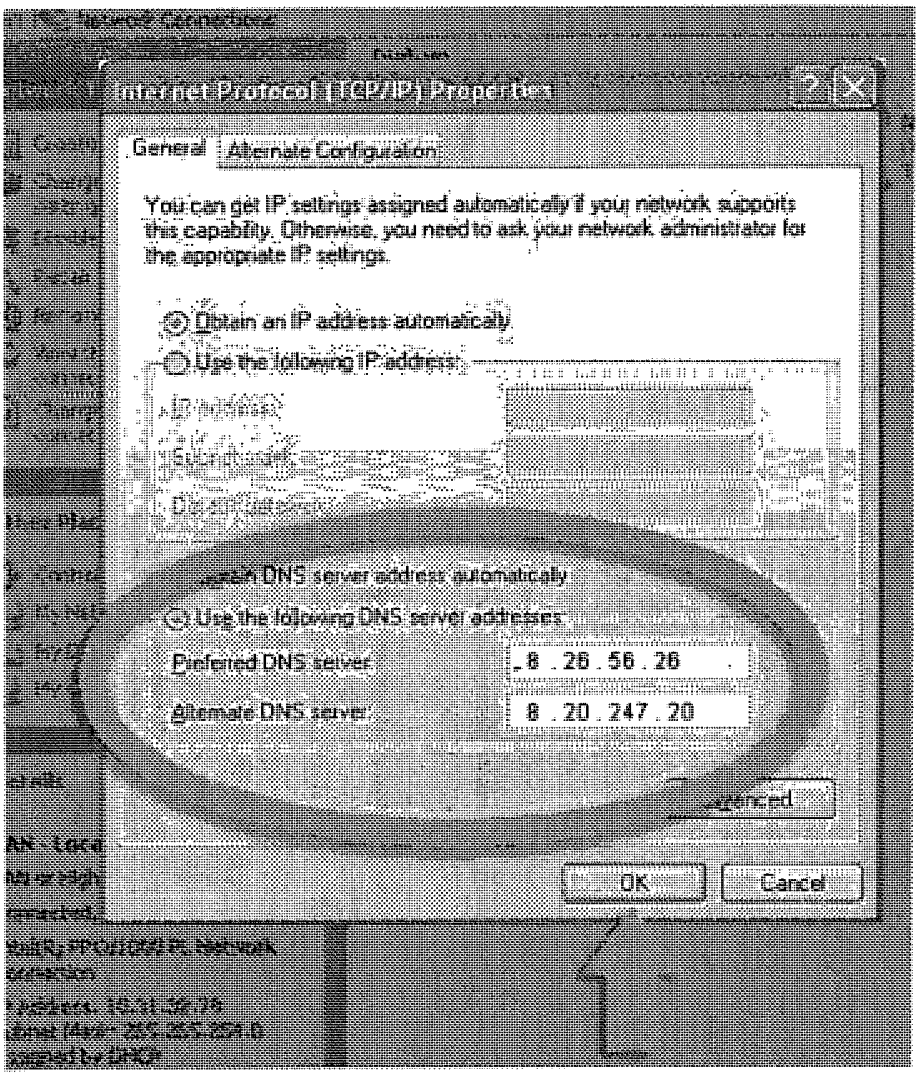


5. Click the radio button Use the following DNS server addresses and type in Comodo Secure DNS addresses in the Preferred DNS server and Alternate DNS server fields.

   Please note down your current DNS settings before switching to Comodo Secure DNS, in case you want to return to your old settings for any reason.

   Preferred DNS server address for Comodo Secure DNS is: 8.26.56.26

   Alternate DNS server address for Comodo Secure DNS is: 8.20.247.20

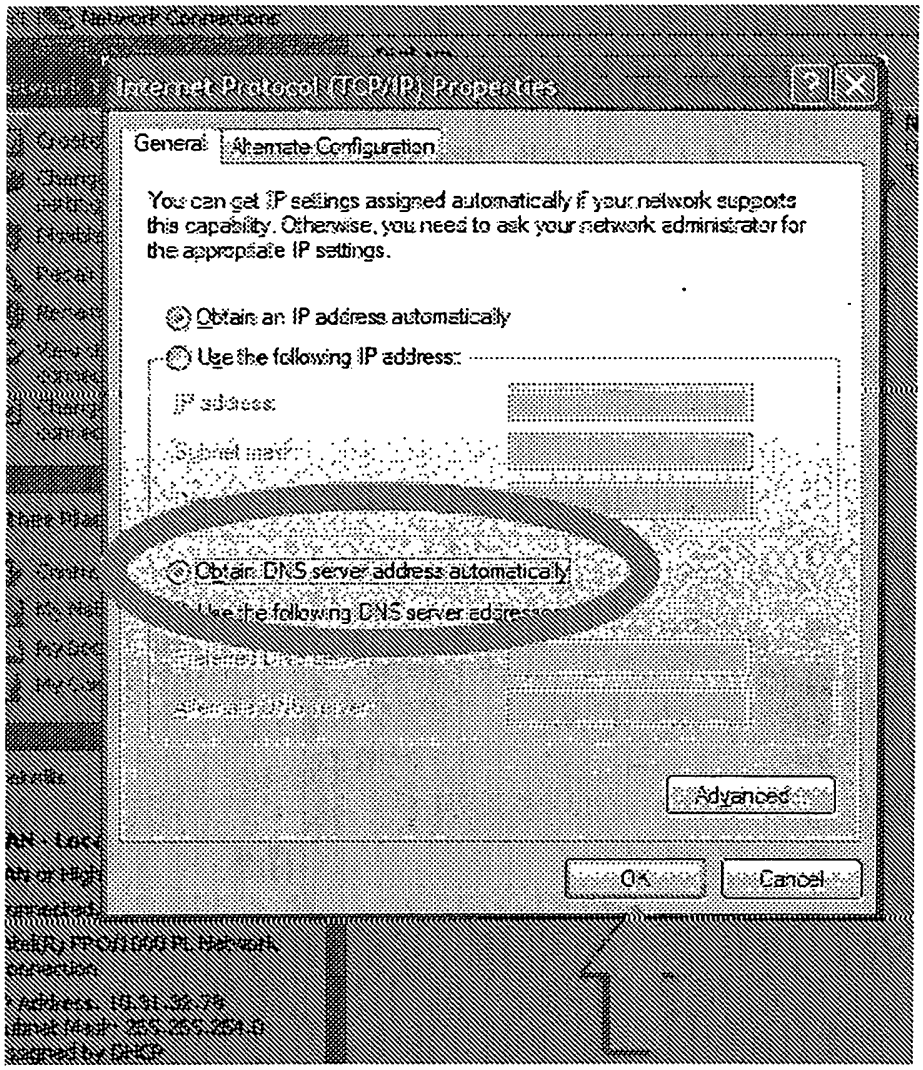


You can disable Comodo Secure DNS by:

- Selecting 'Obtain DNS server address automatically'. This means that you use the DNS server provided by your ISP. This is the option that most home users should choose if they wish to disable the service.

or

- Entering different preferred and alternate DNS server IP addresses.



Windows 7 / Vista - Manually Enabling or Disabling Comodo Secure DNS Service

You can manually enable or disable the Comodo Secure DNS service by changing your DNS server addresses to:

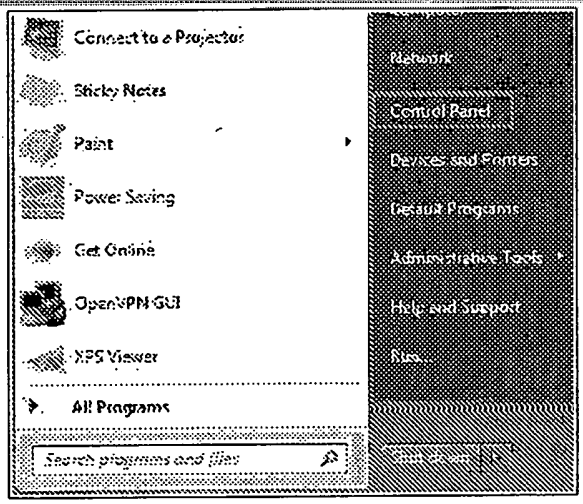
- Preferred DNS : 8.26.56.26
- Alternate DNS : 8.20.247.20

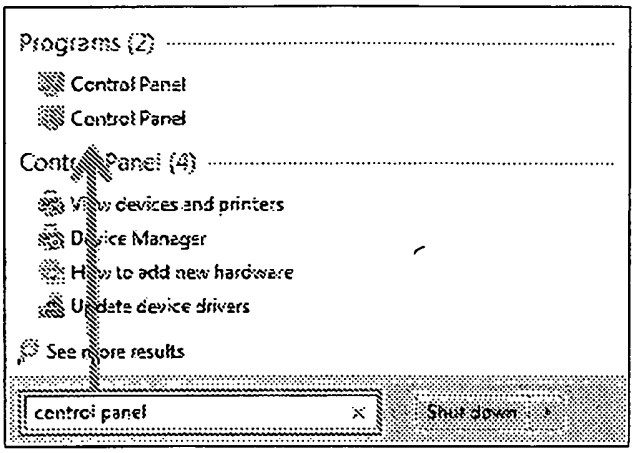
Enabling Comodo DNS in Windows 7 / Vista

Disabling Comodo DNS in Windows 7 / Vista

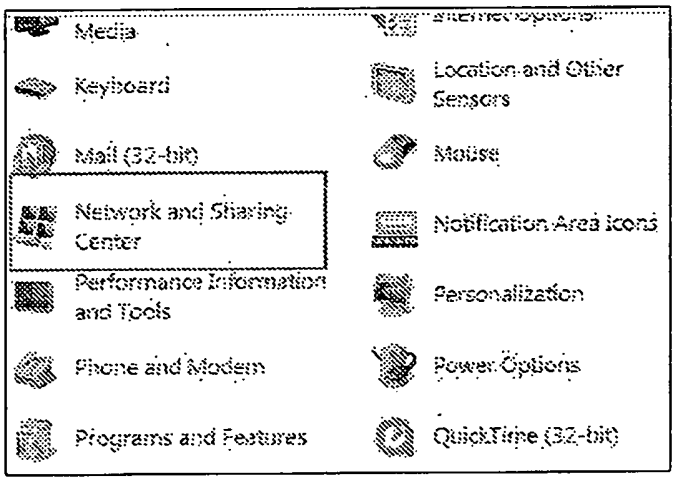
Enabling Comodo DNS in Windows 7 / Vista

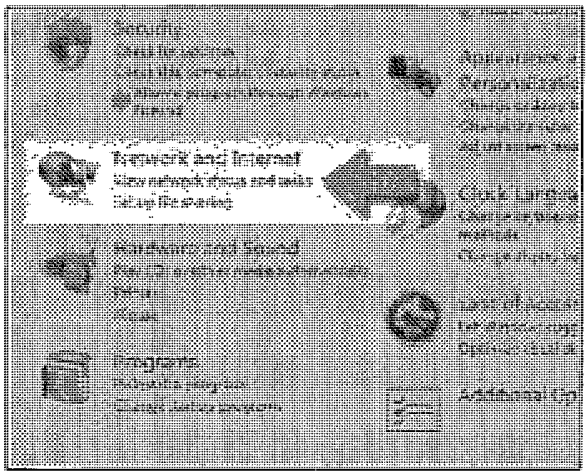
1. Open the control panel by either selecting it from the Windows 'Start' menu or by typing 'control panel' into the search box then clicking the program name.



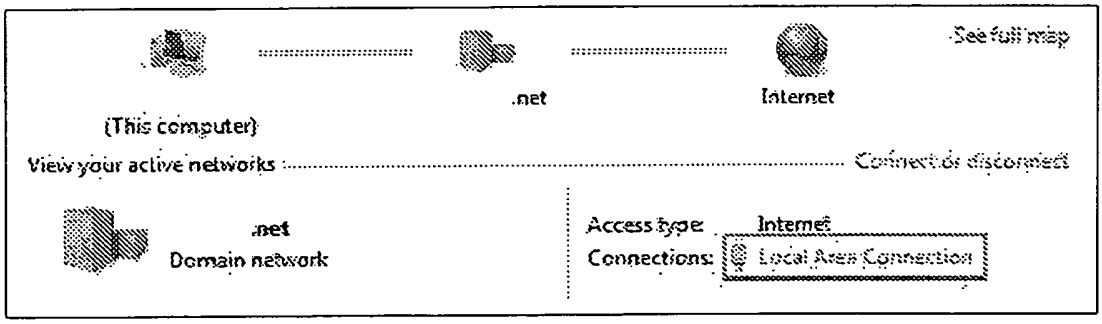


2. From the control panel menu, select 'Network and Sharing Center' (Windows 7) or 'Network and Internet (Vista):

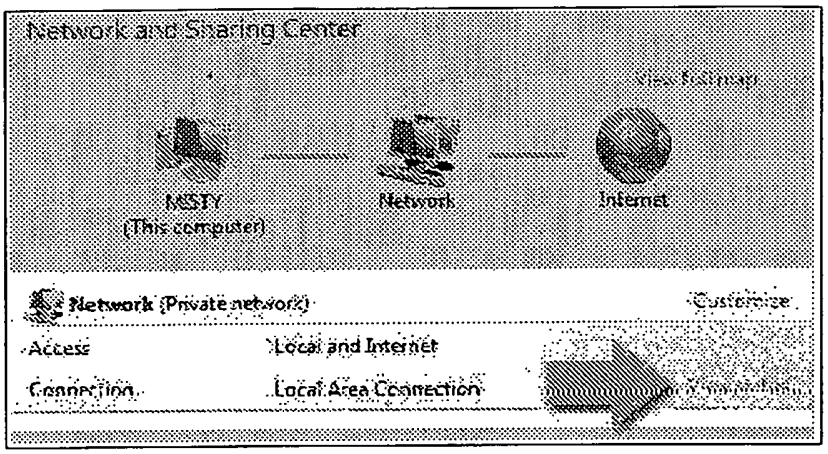




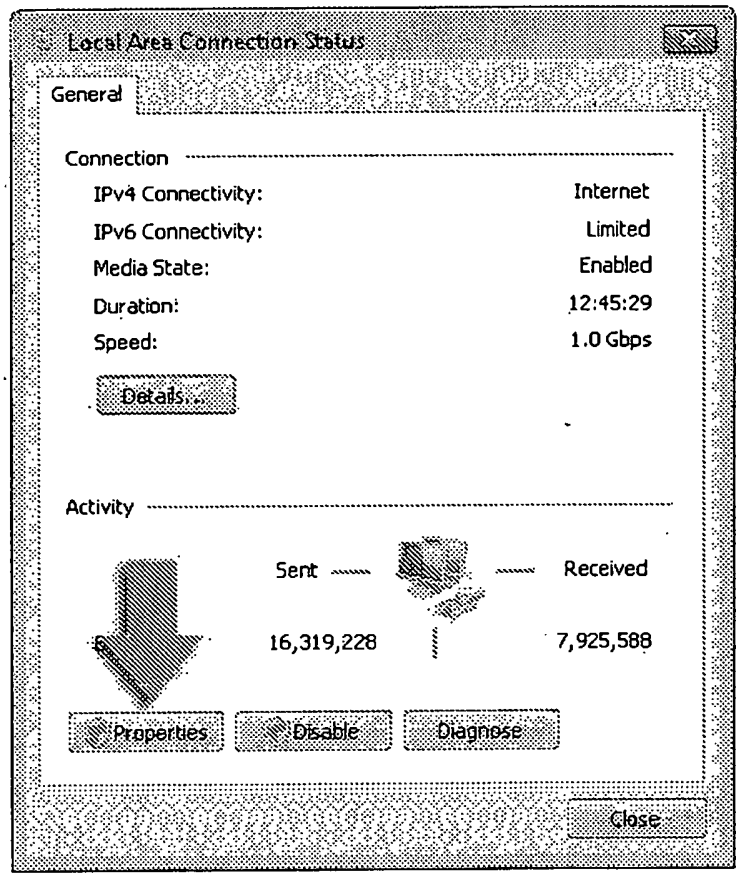
3. In the Network and Sharing center, click the connection type next to 'Connections' (Windows 7):



or 'View Status' (Vista):

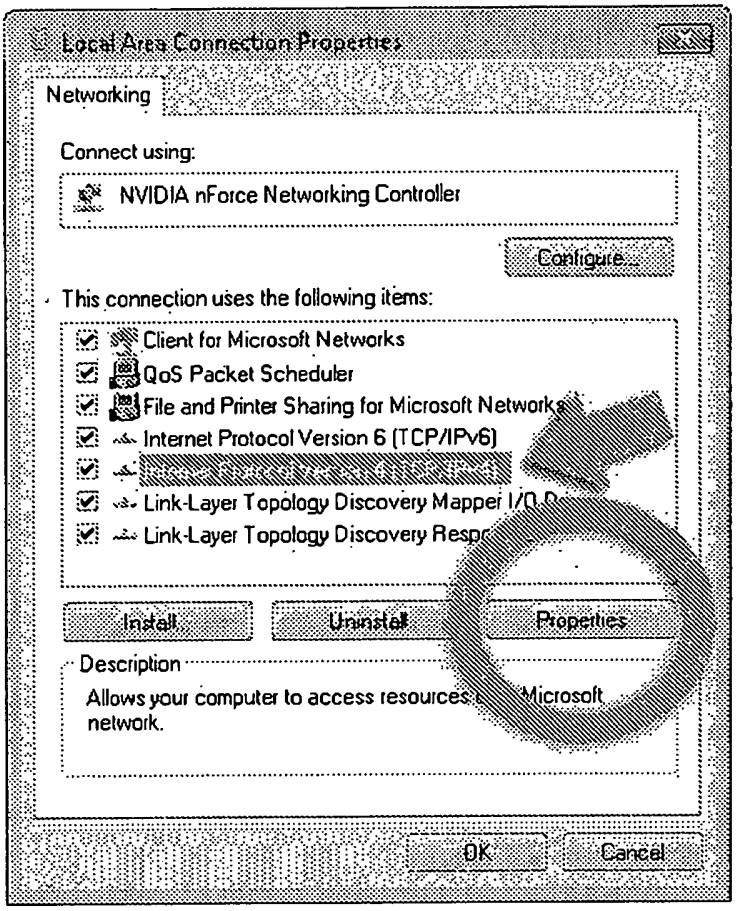


4. This will open the 'Local Area Connection Status' dialog. Click the 'Properties' button:

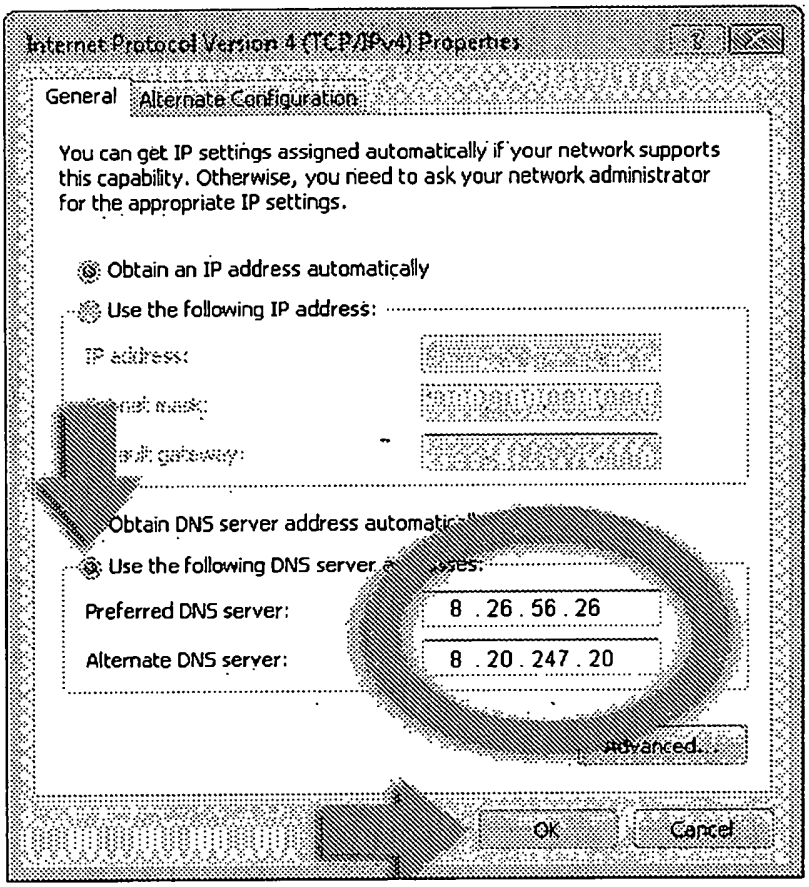


At this point, Windows might ask for your permission to continue or request that you enter an Administrator password.

5. Once you have granted permission/entered an admin password, you will be presented with the 'Local Area Connection Properties' dialog. Scroll down the list and select 'Internet Protocol Version 4 (TCP/IP)' then click the 'Properties' button:



6. Enable 'Use the following DNS server addresses'. Doing so will allow you to enter the addresses of Comodo DNS servers in the fields provided. Enter the addresses listed below then click 'OK' to activate your settings:
    - Preferred DNS : 8.26.56.26
    - Alternate DNS : 8.20.247.20

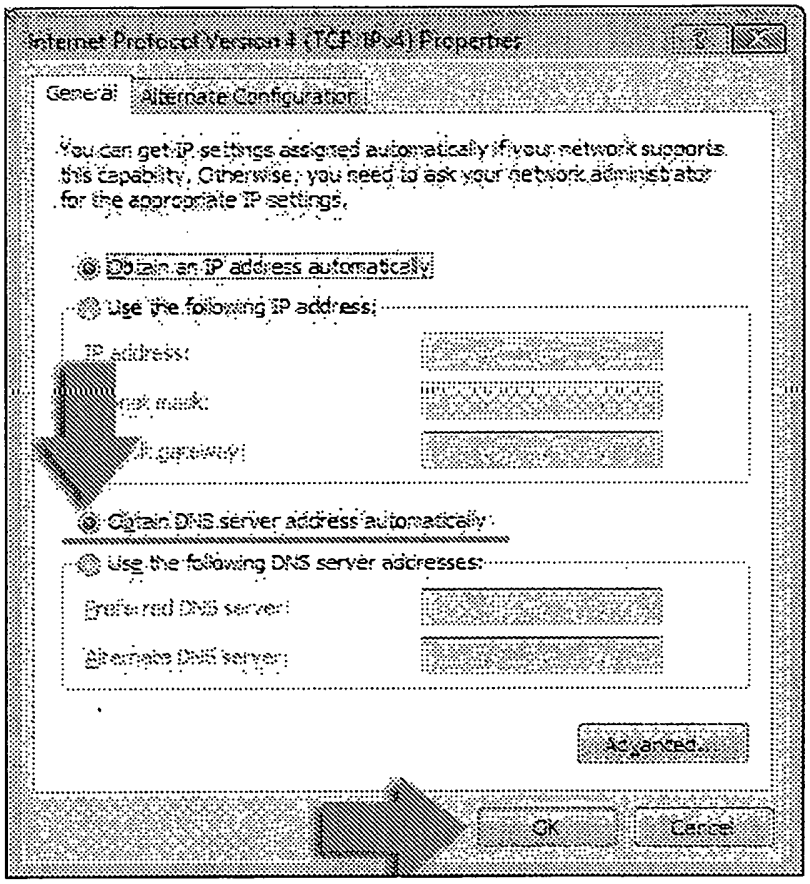


Your computer will now use Comodo DNS as it's default domain name resolution service for all applications that connect to the Internet.

Disabling Comodo DNS in Windows 7 / Vista

To disable Comodo DNS, you need to instruct Windows to automatically obtain the address of a DNS server. Doing so means you will use the DNS server provided by your ISP. To do this:

- Follow steps 1 to 7 of the 'Enabling Comodo DNS in Windows 7 / Vista' tutorial to open the IP4 properties dialog
- Enable 'Obtain DNS server address automatically' then click 'OK'.



Note: Alternatively, you can enter the server addresses of a different DNS service before clicking 'OK'

Appendix 3 - Glossary Of Terms

A B C D E F G H I J K L M N O P Q R S U V W X Y Z

A

ACK

The acknowledgment bit in a TCP packet. (ACKnowledgment code) - Code that communicates that a system is ready to receive data from a remote transmitting station, or code that acknowledges the error-free transmission of data.

Back to the top

Adware

Adware also known as advertising-supported application is designed as a tool to deliver advertisements that provides a source of revenue to its developer. The ads may appear on the screen during the installation process or on the user interface of the application. Since adware is mostly installed along with another software without the user's knowledge and may be used for malicious activities, the term 'adware' is often associated with malware.

Back to the top

Antivirus

An antivirus software is an application which is capable of detecting and removing malicious software such as viruses, trojans, worms and scripts from a computer system. A traditional (or 'classic') antivirus relies on a system of 'black-listed' signatures to detect malicious software. Under this system, antivirus vendors create digital signatures of any executable identified as malware. They then send this list of signatures to their customer's local antivirus software via regular (often daily) updates. The customer's antivirus software will then flag as a virus any program with a signature matching a signature on the blacklist.

One drawback with the signature system is its reactive nature - it can only detect 'known' threats. The vendor has to first identify the file as a virus before they can create a signature of it. In many cases, this means the virus has to have already infected someones computer before a signature can be created to combat it.

Because of this limitation, most modern anti-viruses now deploy a wide range of layered technologies to determine the threat level of a particular file. Such technologies include heuristics, behavior analysis, cloud-based scanning, sand-boxing, host intrusion prevention and file-lock up services.

Back to the top

Antivirus Scan

An audit performed by an antivirus application in order to detect malware and viruses in the file system and/or memory of a computer.

Back to the top

ARP

Address Resolution Protocol (ARP) is a protocol for mapping an IP address to a physical machine address, also known as MAC address, in an Ethernet local area network.

Back to the top

Attached Resource Computer NETwork (ARCNET)

ARCNET is a local area network (LAN) protocol, similar in purpose to Ethernet or Token Ring. ARCNET was the first widely available networking system for microcomputers and became popular in the 1980s for office automation tasks. It has since gained a following in the embedded systems market, where certain features of the protocol are especially useful.

Back to the top

Auto-sandbox

Auto-sandboxing describes the process whereby applications and processes which are unknown to Comodo Internet Security will be automatically run in a isolated operating environment. Sandboxed applications are run under a set of access restrictions so they cannot cause damage the underlying file structure or operating system. The access restriction level applied to

sandboxed applications can be set by the user and includes 'Limited', 'Partially Limited', 'Restricted', 'Untrusted', 'Blocked' and 'Fully Virtualized'.

Conceptually, the auto-sandbox is designed to securely handle 'unknown' executables - those which are not present on Comodo's black-list (definitely malicious) or white-list (definitely safe). If the unknown file turns out to be malicious then it cannot cause any harm because the sand-boxing process denied it access to critical system resources. On the other hand, programs that are unknown but perfectly harmless will run just as well in the sandbox. This allows safe applications the freedom to run as intended while denying malicious applications the ability to cause damage.

The auto-sandbox process is further enhanced if it is married to a system that can subsequently classify these unknown files as either 'safe' or 'malicious'. In Comodo Internet Security, sandboxed files can be submitted to Comodo servers* for automated behavior analysis. If this analysis discovers the file is malicious then it is added to the black-list which is distributed to all CIS users. If the file does not exhibit malicious behavior it is passed to Comodo labs for more in-depth tests and possible inclusion on the white-list.

** if enabled by the user*

Back to the top

B

Behavior Analysis

An activity performed by CIS to determine whether an unknown application in the sandbox is malicious or not. Unknown files are analyzed by Comodo Cloud Scanners and Comodo's Instant Malware Analysis (CIMA) servers. If found to be safe, they will be submitted to Comodo labs for further checks.

Back to the top

Behavior Blocker

A Host Intrusion Protection (HIPS) mechanism that monitors the behavior of software and files in your system and prevents them from taking actions that would cause damage.

Back to the top

Brute-force

Brute-force search is a trivial but very general problem-solving technique, that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

Back to the top

Buffer Overflow

A buffer overflow is an anomalous condition where a process/executable attempts to store data beyond the boundaries of a fixed-length buffer. The result is that the extra data overwrites adjacent memory locations, often causing the process to crash or produce incorrect results. Hackers use buffer overflows as a trigger to execute to execute malicious code.

Back to the top

Bug

Error in a program that cause problems.

Back to the top

C

CA - Certification Authority

A Certificate Authority (CA) is trusted third party that validates ownership information about a web-server then issues an SSL/TLS certificate to the organization that owns the server. The certificate is then placed on the web-server and is used to secure connections between the server and any clients (browsers) that connect to it. For example, an online store would use a certificate to secure its order forms and payment pages.

A Certificate Authority (CA) such as Comodo CA will sign the certificates it issues with their private key. However, for the website's certificate to operate correctly, there is a reciprocal client side requirement - the internet browser that the visitor is using MUST physically contain the certificate authority's 'root certificate'. This root is required to successfully authenticate any website certificates that have been signed by the CA. If the root certificate is not embedded in a browser, then the website's certificate will not be trusted and visitors will see an error message. Certificate Authorities proactively supply browser vendors with their root certificates for inclusion in the browser's 'certificate store' - an internal repository of root certificates that ships with

each browser.

Back to the top

CIS Widget

The CIS Widget is a handy control panel that shows information about the security status of your computer, the speed of outgoing and incoming traffic and other useful information. The widget also has shortcuts to common CIS tasks and allows users to launch sandboxed instances of any internet browser they have installed on their system. By default, the widget is displayed on the desktops of Windows computers running CIS version 6.0 and above.

Back to the top

COM Interfaces

Component Object Model (COM) is Microsoft's object-oriented programming model that defines how objects interact within a single application or between applications - specifying how components work together and inter-operate. COM is used as the basis for Active X and OLE - two favorite targets of hackers and malicious programs to launch attacks on a computer. Comodo Internet Security automatically protects COM interfaces against modification.

Back to the top

Computer Network

A computer network is a connection between computers through a cable or some type of wireless connection. It enables users to share information and devices between computers and other users within the network.

Back to the top

D

Debugging

The process of identifying a program error and the circumstances in which the error occurs, locating the source(s) of the error in the program and fixing the error.

Back to the top

DHCP

Dynamic Host Configuration Protocol (DHCP) is a communications protocol that lets network administrators manage and automate the assignment of Internet Protocol (IP) addresses in an organization's network. DHCP allows devices to connect to a network and be automatically assigned an IP address.

Back to the top

Digital Certificate

A digital certificate is a file used to cryptographically bind a company's Public Key to its identity. Like a driving license or passport binds a photograph to personal information about its holder, a digital certificate binds a Public Key to information about that company. They are issued for between 1 and 5 year validity periods.

Digital certificates are issued by a Certificate Authority like Comodo. Each CA acts as a trusted third party and conducts background checks on a company to ensure they are legitimate before issuing a certificate to them. Apart from providing an encrypted connection between a internet browser and a website, digital certificates are intended to reassure website visitors that the company they are about to make a purchase from can be trusted.

To get a digital certificate, a company must first generate a Certificate Signing Request (CSR) on their web-server. This CSR contains their public key and their identity information. They then enroll and pay for the certificate and send their CSR to the CA.

The CA's validation department will check that the identity information in the CSR is correct by conducting background checks and will sometimes request that the company supplies documentation such as articles of incorporation. Once validation is satisfactorily completed, the CA will issue the certificate to the customer. The customer will then install it on their website to secure sensitive areas like payment pages.

Back to the top

Digital Signature

Digital signatures are used for authentication and integrity, meaning it guarantees that the person sending a message is indeed the same person who he/she claims to be and the message has not been altered. To authenticate oneself using a digital signature, a person needs to download and install Digital Certificates in their systems from Certificate Authorities such as Comodo. The client certificate then can be imported into their browsers and email clients. The same certificate can also be used

to digitally sign a document before sending it. The recipient can easily find out if the document has been tampered with en-route.

Back to the top

DNS

DNS stands for Domain Name System. It is the part of the Internet infrastructure that translates a familiar domain name, such as 'example.com' to an IP address like 123.456.789.04. This is essential because the Internet routes messages to their destinations on the basis of this destination IP address, not the domain name. When a user searches for a website name like 'www.domain.com', their browser will first contact a DNS server to discover the IP address associated with that domain name. Once it has this information, it can successfully connect to the website in question.

Back to the top

Dynamic IP

The procedure of allocating temporary IP addresses as they are needed. Dynamic IP's are often, though not exclusively, used for dial-up modems.

Back to the top

E

Encryption

Encryption is a technique that is used to make data unreadable and make it secure. Usually this is done by using secret keys and the encrypted data can be read only by using another set of secret keys. There are two types of encryption - symmetric encryption and asymmetric encryption.

Symmetric encryption is applying a secret key to a text to encrypt it and use the same key to decrypt it. The problem with this type of encryption lies during the exchange of secret keys between the sender and the recipient over a large network or the Internet. The secret keys might fall into wrong hands during the exchange process.

Asymmetric encryption overcomes this problem by using two cryptographically related keys, a key pair - a public key and a private key. The private key is kept secret in your system and the public key is made available freely to anyone who might want to exchange messages with you. Any message, be it text, documents or binary files that are encrypted using the public key can be decrypted using the corresponding private key only. Similarly anything that is encrypted using the private key can be decrypted using the corresponding public key. Typically public keys are made available to everyone by using Digital Certificates. The certificates are issued by a Certificate Authority (CA), which identifies a server or user and usually contains information such as the CA who issued it, the organization's name, email address of the user and country and the public key of the user. When a secure encrypted communication is required between a client and a server, a query is sent over to the other party for the certificate and the public key can be extracted from it.

Back to the top

End User

The person who uses a program after it's been compiled and distributed.

Back to the top

EPKI Manager

Enterprise Public Key Infrastructure Manager. The EPKI Manager allows you to issue bulk numbers of:

- SSL Certificates for use on domain names owned by your Company;
- SecureEmail Certificates (S/MIME) for use by employees of your Company.

Your nominated EPKI Manager Administrator(s) will be able to manage all the company's Certificates from a central web based console. Additional certificates may be purchased through the console in minutes; ensuring new servers and employee email may be secured in minutes rather than days. For more information about EPKI Manager click here.

Back to the top

Ethernet

Ethernet is a frame-based computer networking technology for local area networks (LANs). The name comes from the physical concept of ether. It defines wiring and signaling for the physical layer, and frame formats and protocols for the media access control (MAC)/data link layer of the OSI model. Ethernet is mostly standardized as IEEEs 802.3. It has become the most widespread LAN technology in use during the 1990s to the present, and has largely replaced all other LAN standards such as token ring, FDDI, and ARCNET.

Back to the top

Executable Files

An 'executable' is a file that instructs a computer to perform a task or function. Every program, application and device run on computer requires an executable file of some kind to start it. The most recognizable type of executable file is the '.exe' file. For example, when Microsoft Word is started, the executable file 'winword.exe' instructs the computer to start and run the Word application. Other types of executable files include those with extensions .cpl .dll, .drv, .inf, .ocx, .pf, .scr, .sys.

Back to the top

F

False Positive

When an antivirus scan is run and the scanner reports that some programs are infected with malware which may not be the actual case and the files are safe. This kind of false alert is called 'False Positive'. Too much of False Postive results can be annoying and the user might just ignore legitimate warning or delete legitimate files causing the relevant program or operating system to malfunction.

Back to the top

Firewall

A firewall is an application that helps an user or administrator to have a control over how the system should be connected with other network/systems or over the Internet.

Back to the top

FS type

Type of file system.

Back to the top

FTP

File Transfer Protocol (FTP) is a protocol used for file transfer from computer to computer across a TCP network like the internet. An anonymous FTP is a file transfer between locations that does not require users to identify themselves with a password or log-in. FTP uses the TCP/IP protocols to enable data transfer. FTP is most commonly used to download files from a server or to upload a file to a server.

Back to the top

G

Graphical User Interface (GUI)

The visual symbols and graphics with which a user controls a piece of software or device. Most software has a GUI that comprises of windows, menus, and toolbars. The user interacts with the GUI by clicking their mouse on a GUI element. Operating systems like Windows use GUI's because most users find them easier to use than less friendly interfaces like a command line.

Back to the top

H

Heuristics

Heuristics is a technique that continuously evolves based on experience for solving problems, discovery and learning. When the term is used in computer security parlance, Heuristics is about detecting virus-like behavior or attributes rather than looking for a precise virus signature that match a signature on the virus blacklist. Comodo Internet Security applies this technology in the application, which is a quantum leap in the battle against malicious scripts and programs as it allows the engine to 'predict' the existence of new viruses - even if it is not contained in the current virus database.

Back to the top

HIPS

A Host Intrusion Protection System (HIPS) is designed to identify and block zero malware by monitoring the behavior of all applications and processes. It is designed to prevent actions that could cause damage to your operating system, system-memory, registry keys or personal data.

Security software using a HIPS system will generally enforce rules prescribing the permitted activities of processes and executables at the point of execution. Examples of such activities can include changes to files or directories, accessing protected COM interfaces, modifications to the registry, starting up another application or writing to the memory space of another

application. The precise nature of these rules can be set by the user or pre-configured by the vendor.

If an executable or process attempts to perform an action that transgresses these rules then the HIPS system will block the attempt and generate an alert notifying the user of that action. Most HIPS alerts will also include security advice.

Back to the top

HTTP

HTTP (Hypertext Transfer Protocol) is the foundation protocol of the World Wide Web. It sets the rules for exchanges between browser and server. It provides for the transfer of hypertext and hypermedia, for recognition of file types, and other functions.

Back to the top

I

ICMP

The Internet Control Message Protocol (ICMP) is part of Internet Protocol (IP) suite and used to report network applications communications errors, network congestion, timeouts and availability of remote hosts.

Back to the top

IDS

An Intrusion Detection System (IDS) is software/hardware that detects and logs inappropriate, incorrect, or anomalous activity. IDS are typically characterized based on the source of the data they monitor: host or network. A host-based IDS uses system log files and other electronic audit data to identify suspicious activity. A network-based IDS uses a sensor to monitor packets on the network to which it is attached.

Back to the top

IMAP

Internet Message Access Protocol. IMAP is a method of distributing email. It is different from the standard POP3 method in that with IMAP, email messages are stored on the server, while in POP3, the messages are transferred to the client's computer when they are read. Thus, using IMAP allows you to access your email from more than one machine, while POP3 does not. This is important because some email servers only work with some protocols.

Back to the top

Information Security Exposure

An information security exposure is a mistake in software that allows access to information or capabilities that can be used by a hacker as a stepping-stone into a system or network.

Back to the top

Internet Service Provider (ISP)

A company or organization that provides the connection between a local computer or network, and the larger Internet.

Back to the top

IP - Internet Protocol

The Internet Protocol (IP) is a data-oriented protocol used by source and destination hosts for communicating data across a packet-switched network. An IP address is a numeric address that is used to identify a network interface on a specific network or subnetwork. Every computer or server on the Internet has an IP address. When a user types a domain name such as www.domain.com into the address bar of their browser, the browser still needs to find the IP address associated with that domain in order to reach the website. It finds the IP address by consulting with a DNS server.

There are currently two versions of IP in use today - IPv4 and ipv6.

IPv4 (Internet Protocol version 4) was developed in 1981 and is still the most widely deployed version - accounting for almost all of today's Internet traffic. However, because IPv4 uses 32 bits for IP addresses, there is a physical upper limit of around 4.3 billion possible IP addresses - a figure widely viewed as inadequate to cope with the further expansion of the Internet. In simple terms, the number of devices requiring IP addresses is in danger of exceeding the number of IP addresses that are available.

IPv6 is intended to replace IPv4, which uses 128 bits per address (delivering 3.4×1038 unique addresses) and is viewed as the only realistic, long term solution to IP address exhaustion. IPv6 also implements numerous enhancements that are not present in IPv4 - including greater security, improved support for mobile devices and more efficient routing of data packets.

Back to the top

K

Key Logger

Key logger is a software application or a hardware device that keeps tracks of computer activity in real time including the keys that are pressed. Key loggers are used to troubleshoot technical problems in computer systems. The application can also be used for malicious purposes such as to steal passwords and other sensitive information.

Back to the top

L

LAN

A local area network (LAN) is a computer network covering a small local area, like a home, office, or small group of buildings such as a home, office, or college. Current LANs are most likely to be based on switched Ethernet or Wi-Fi technology running at 10, 100 or 1,000 Mbit/s (1,000 Mbit/s is also known as 1 Gbit/s).

Back to the top

Leak Test

Leak Test is a way to find out how well your system is protected by your security software from external and internal threats. Typically these tests are down-loadable and should not cause any harm to your system while being run. The Firewall Leak Tests are used to test how effective the firewall component of your security software is at detecting and blocking outgoing connection attempts. If an application is able to connect to the internet without your knowledge, it poses a real danger meaning it can easily retrieve private and confidential information from your system and transmit it.

Host Intrusion Prevention System (HIPS) tests are designed to test how well your security software is capable of protecting your internal system from malicious attacks such as viruses. A good HIPS system will deny the malware from accessing your critical operating system files, registry keys, COM interfaces and running processes.

Back to the top

License

The official terms of use for a specific program. A software license is a legal document since it formally restricts the rights of the user.

Back to the top

M

MAC Address

A Media Access Control (MAC) address is a number that is hardwired in network adapters and is used to identify the device or system in which it is installed.

Every device on a network has two addresses: a MAC (Media Access Control) address and an IP (Internet Protocol) address. The MAC address is the address of the physical network interface card inside the device, and never changes for the life of the device (in other words, the network card inside the PC has a hard coded MAC address that it keeps even if installed it in a different machine). On the other hand, the IP address can change if the machine moves to another part of the network or the network uses DHCP to assign dynamic IP addresses. In order to correctly route a packet of data from a host to the destination network card it is essential to maintain a record of the correlation between a device's IP address and it's MAC address. The Address Resolution Protocol performs this function by matching an IP address to its appropriate MAC address (and vice versa). The ARP cache is a record of all the IP and MAC addresses that the computer has matched together.

Back to the top

Malicious File

Often called 'Malware', a malicious file is software designed to damage computer systems, steal sensitive information or gain unauthorized access to private computer systems. For example it may be coded to gather sensitive information from a system such as passwords, credit card details and send them back to the creator of the malware.

Back to the top

Malware

Malware is short for 'malicious software'. It is an umbrella term that describes a wide range of malicious software including viruses, trojans, worms, scripts and root kits. When installed on a computer system or network, malware can disrupt operations, steal sensitive and personal information, delete important data, create zombie networks and perform other destructive operations.

Back to the top

N

Network (computer)

Networking is the scientific and engineering discipline concerned with communication between computer systems. Such networks involves at least two computers, which can be separated by a few inches (e.g. via Bluetooth) or thousands of miles (e.g. via the Internet). Computer networking is sometimes considered a sub-discipline of telecommunications.

Back to the top

Network Zone

A Network Zone can consist of an individual machine (including a single home computer connected to Internet) or a network of thousands of machines to which access can be granted or denied. The creation of network zones helps an administrator to apply changes for all the computer(s) in selected zone(s).

Back to the top

NIDS

NIDS - Network-Based Intrusion Detection System. Detects intrusions based upon suspicious network traffic. A network intrusion detection system (NIDS) is a system that tries to detect malicious activity such as denial of service attacks, port-scans or even attempts to crack into computers by monitoring network traffic.

Back to the top

NNTP

Network News Transfer Protocol - Refers to the standard protocol used for transferring Usenet news from machine to machine. A protocol is simply a format used to transfer data to two different machines. A protocol will set out terms to indicate what error checking method will be used, how the sending machine will indicate when it is has finished sending the data, and how the receiving machine will indicate that it has received the data.

Back to the top

O

Operating System (OS)

The essential software to control both the hardware and other software of a computer. An operating system's most obvious features are managing files and applications. An OS also manages a computer's connection to a network, if one exists. Microsoft Windows, Macintosh OS, and Linux are operating systems.

Back to the top

P

Ping

Ping is a computer network tool used to test whether a particular host is reachable across an IP network.

Back to the top

PKCS

PKCS refers to a group of Public Key Cryptography Standards devised and published by RSA Security.

Back to the top

PKCS#7

See RFC 2315. Used to sign and/or encrypt messages under a PKI. Used also for certificate dissemination (for instance as a response to a PKCS#10 message). Formed the basis for S/MIME, which is now based on RFC 3852, an updated Cryptographic Message Syntax Standard (CMS).

Back to the top

PKCS#10

See RFC 2986. Format of messages sent to a certification authority to request certification of a public key. See certificate signing request.

Back to the top

PKCS#12

Defines a file format commonly used to store private keys with accompanying public key certificates, protected with a password-based symmetric key.

Back to the top

Plugin

A program that allows a Web browser to display a wider range of content than originally intended. For example: the Flash plugin allows Web browsers to display Flash content.

Back to the top

POP2

There are two versions of POP. The first, called POP2, became a standard in the mid-80's and requires SMTP to send messages. The newer version, POP3, can be used with or without SMTP.

Back to the top

POP3

POP3 is the abbreviation for Post Office Protocol - a data format for delivery of emails across the Internet.

Back to the top

Ports

A computer port is an interface that allows communication between applications or processes running on a host computer and other computers, devices or networks.

Your computer sends and receives data to other computers and to the Internet through a port. There are over 65,000 numbered ports on every computer - with certain ports being traditionally reserved for certain services. For example, your machine almost definitely connects to Internet using port 80 and port 443. Your e-mail application connects to your mail server through port 25.

Back to the top

Potentially Unwanted Applications

A potentially unwanted application (PUA) is a piece of software that (i) a user may or may not be aware is installed on their computer, and/or (ii) may have functionality and objectives that are not clear to the user. Example PUA's include adware and browser toolbars. PUA's are often installed as an additional extra when the user is installing an unrelated piece of software. Unlike malware, many PUA's are 'legitimate' pieces of software with their own EULA agreements. However, the 'true' functionality of the software might not have been made clear to the end-user at the time of installation. For example, a browser toolbar may also contain code that tracks a user's activity on the Internet. Because of this ambiguity, many antivirus companies use the term 'Potentially Unwanted Application' to identify such software.

Back to the top

Q

Quarantined Files

After an antivirus scan, files that are detected as malware may either be deleted immediately or isolated in a secure environment known as 'quarantine'. Any files moved into quarantine are encrypted so they cannot be run or executed. This prevents infected files from corrupting the rest of a computer.

Back to the top

R

Registry Keys

The Windows Registry serves as an archive for collecting and storing the configuration settings of all computer hardware, software and Windows components. Every time an application or hardware is started, it will access the registry keys relating to it. Applications will also access and modify their registry keys constantly during the course of their execution. As the registry is one of the most regularly accessed parts of Windows, it plays a critical role in the stability, reliability and performance of a computer. Indeed, many computer problems are caused by registry errors. Corrupt keys and invalid keys left by uninstalled applications can often cause severe degradation in system performance, crashes and, in extreme cases, can render a system un-bootable. Inexperienced users are, however, discouraged from making manual adjustments to the registry because a single change can have potentially devastating consequences. There are several dedicated registry cleaners available today, including Comodo PC TuneUp.

Back to the top

S

S/MIME

S/MIME (Secure / Multipurpose Internet Mail Extensions) is a standard for public key encryption and signing of email

encapsulated in MIME.

Back to the top

Single User Certificate

A single use certificate refers to the x.509 and associated private key generated by SecureEmail on Alice; stored on SES and downloaded by Bob after a successful SSL client authentication.

Back to the top

SMB

A message format used by DOS and Windows to share files, directories and devices. NetBIOS is based on the SMB format, and many network products use SMB. These SMB-based networks include Lan Manager, Windows for Workgroups, Windows NT, and Lan Server. There are also a number of products that use SMB to enable file sharing among different operating system platforms.

Back to the top

SMTP

Simple Mail Transfer Protocol is the most widely used standard for email transmission across the Internet. SMTP is a relatively simple, text-based protocol, where one or more recipients of a message are specified (and in most cases verified to exist) and then the message text is transferred.

Back to the top

SNMP

Simple Network Management Protocol. The network management protocol used almost exclusively in TCP/IP networks. SNMP provides a means to monitor and control network devices, and to manage configurations, statistics collection, performance, and security.

Back to the top

Spyware

Spyware is a program that performs certain actions without the consent of the user such as displaying advertisements, collecting personal and sensitive information and changing the configuration of the computer. Not all tracking software are malicious since you may have agreed to the conditions as a trade-off for obtaining certain services for free. The tracking software will monitor your online activities to decide what kind of ads should be shown for you.

Back to the top

SSL

Secure Sockets Layer (SSL) is a commonly used protocol for ensuring secure message transmission on the internet. It facilitates an encrypted connection between a web server and an internet browser. It was developed by Netscape in 1994 as a direct response to growing concerns over internet security.

The encryption provided by SSL means that all data passed between a web server and a browser is private and cannot be eavesdropped on. You can tell if you are in an SSL session if the URL begins with https.

SSL is used on the payment pages of millions of websites to protect their online transactions with their customers.

Back to the top

STATIC IP

An IP address which is the same every time you log on to the Internet. See IP for more information.

Back to the top

Stealth Port

Port Stealthing is a security technique whereby ports on an Internet connected PC are hidden so that they provide no response to a remote port scan.

A computer sends and receives data to other computers and to the Internet through an interface called a port. There are over 65,000 numbered ports on every computer - with certain ports being traditionally reserved for certain services. For example, most computers connect to the internet using ports 80 and port 443. Most e-mail applications connect to their mail server through port 25. A 'port scanning' attack consists of sending a message to each port to find out which are open and which are

being used by services. With this knowledge, a hacker can determine which attacks are likely to work against a particular computer. Port stealthing effectively makes it invisible to a port scan. This differs from simply 'closing' a port as NO response is given to any connection attempt ('closed' ports respond with a 'closed' reply- revealing to the hacker that there is actually a PC in existence).

Back to the top

Stateful Packet Inspection

Stateful Packet Inspection, also known as SPI, is an enhanced firewall technique that uses dynamic packet filtering method over the older method of static packet filtering. SPI scrutinizes the packet contents, monitors traffic and keeps track of the sources of packets. A network administrator can configure the firewall that uses SPI according to the needs of the organization, for example, close ports until requested by legitimate users to open them.

Back to the top

SYN

SYN (synchronize) is a type of packet used by the Transmission Control Protocol (TCP) when initiating a new connection to synchronize the sequence numbers on two connecting computers. The SYN is acknowledged by a SYN/ACK by the responding computer.

Back to the top

T

TCP

TCP stands for Transmission Control Protocol. TCP is one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

Back to the top

Token-Ring

LAN technology was developed and promoted by IBM in the early 1980s and standardized as IEEE 802.5 by the Institute of Electrical and Electronics Engineers. Initially very successful, it went into steep decline after the introduction of 10BASE-T for Ethernet and the EIA/TIA 568 cabling standard in the early 1990s. A fierce marketing effort led by IBM sought to claim better performance and reliability over Ethernet for critical applications due to its deterministic access method, but was no more successful than similar battles in the same era over their Micro Channel architecture. IBM no longer uses or promotes Token-Ring. Madge Networks, a one time competitor to IBM, is now considered to be the market leader in Token Ring.

Back to the top

Trojan

A Trojan is a type of malware that looks like a legitimate piece of software and users are tricked to install and execute in their computers. The malware takes the name from the Greek mythology, Trojan Horse, a wooden horse that was used by the Greeks to infiltrate the city of Troy. Once the malware is activated, it can damage the system, spread other computer viruses and also create a back door so as to allow online fraudsters to take access or control the system.

Back to the top

Trusted Files

In Comodo Internet Security, a trusted file is one that is considered safe and is allowed to run on a user's computer. This type of file can also be referred to as a 'safe' file or a 'white-listed' file.

A file will be treated as safe if it is in the 'Trusted Files' list OR if it is digitally signed by a 'Trusted Software Vendor'. Comodo Internet Security ships with a list of trusted files and a list of Trusted Vendors. Users can add their own trusted files and vendors to their local installation. They can also submit files and vendors to Comodo so they can be considered for inclusion in future safe lists.

Back to the top

Trusted Software Vendor

A Trusted Software Vendor (TSV) is a publisher of software that is automatically trusted by Comodo Internet Security software. Executable files that have been digitally signed by a TSV will be allowed to run normally and will not be placed in the sandbox.

Many software vendors digitally sign their software with a code signing certificate. Digitally signed software helps a user to identify the publisher and to be sure that the software he/she is downloading is genuine and has not been tampered with. Each code signing certificate is counter-signed by a trusted certificate authority (CA) after the CA has conducted detailed checks that the vendor is a legitimate company.

Back to the top

U

User

A person who uses a computer, including a programmer or end user.

Back to the top

V

Virtual Desktop

The Virtual Desktop is a standalone sandbox featured in Comodo Internet Security which allows users to run any applications in a completely virtual environment. Software in the virtual desktop will not affect other processes, programs or data on the user's computer. Similarly, internet browsers running in the virtual desktop leave behind no personally identifying cookies or history on an employee's real system. The virtual desktop also features a virtual keyboard which provides additional security when entering usernames and passwords on website login pages. Although the virtual desktop is primarily intended for users to test unknown or beta software and for launching highly secure browsing sessions, it can be used to run most software. The virtual desktop interface is available in both desktop and tablet optimized versions.

Back to the top

Virtual Machine (VM)

Virtual machine is a software application that emulates a computing environment in which a program or an operating system can be installed and run. There are many advantages in using a VM such as for testing out new applications or procedures without affecting the host system.

Back to the top

Virus

A computer virus is an executable application capable of causing damage to computer files, folders and components. Viruses are also capable of self-replication so can infect multiple items on a system if left unchecked. The malicious activities performed by a virus are wide ranging and include stealing confidential information, modifying user data, overwriting or damaging files and erasing hard disk content.

Back to the top

Viruscope

Viruscope is an innovative subsystem that monitors all the processes running on a computer in real time to find any suspicious actions taken by any of the processes. If a suspicious activity is identified, Viruscope generates an alert. The alert allows the user to quickly block the process, reverse the effects of the action and move the parent application of the process to quarantine, if the activity is found to be malicious, or to allow the process, if the action is found to be legitimate.

Back to the top

Virus Database

A database of the digital signatures of all known computer viruses and malware. This database, sometimes referred to as a 'black list', enables antivirus software to detect any malware running on a customer's computer.

Every time a file or executable is identified as being malware, antivirus companies will create a digital signature of the file and add it to their database of blacklisted files. This database is then distributed to their customers as an update to their antivirus software. If the blacklisted signature of the malware is found anywhere on a customers computer, then the file is flagged as infected and may be quarantined or deleted.

Comodo has a dedicated team of technicians and crawlers that are continually searching for new virus strains to add to our database. Comodo's virus database is available for public download at http://internetsecurity.comodo.com/updates/vdp/database.php

Back to the top

Vulnerability

In network security, a vulnerability refers to any flaw or weakness in the network defense that could be exploited to gain unauthorized access to, damage or otherwise affect the network.

Back to the top

W

Website Filtering

Website Filtering is a security technique whereby access to specific websites can be selectively blocked or allowed to particular users of the computer. The website filtering is very useful for parental control as it allows to block inappropriate websites to juvenile users. Also, in work environments, administrators can prevent employees from visiting social networking sites during working hours.

Back to the top

Web server

The term Web server can mean one of two things:

1. A computer that is responsible for accepting HTTP requests from clients, which are known as Web browsers, and serving them Web pages, which are usually HTML documents and linked objects (images, etc.).

2. A computer program that provides the functionality described in the first sense of the term.

Back to the top

Worm

A Worm, another type of malware, unlike virus is capable of spreading from computer to computer without any human help. The worm with its capability to replicate itself several times over consumes most of the system memory causing the computer to slow down or crash altogether. It can also cause bandwidth jam while spreading to other computers in the network.

Back to the top

Wildcard

Wildcards are symbols that add flexibility to a keyword search by extending the parameters of a search word. A wildcard item is usually denoted with the asterisk symbol, '*'. This stands for one-or-more characters (useful for all suffixes or prefixes). In digital certification terms, a 'wildcard certificate' means that the certificate will secure the domain plus unlimited sub-domains of that domain. A wildcard certificate is applied for using the format '*.domain.com'.

Back to the top

X

X.509

An internationally recognized standard for certificates that defines their required parts

Back to the top

Z

Zero-Day Malware

Zero-day malware describes new computer viruses or worms that have been discovered in the public realm but which antivirus vendors have not yet created a digital signature for. The term means that the antivirus companies have had 'zero-days' to react. New malware can reasonably be called 'zero-day' for the the length of time between its discovery and the creation of a signature to combat it. For most antivirus vendors, this is usually measured in a matter of hours. Of course, the malware itself may have been at large for a much longer period of time before it was discovered. Because of this window of vulnerability, most security software has grown beyond a reliance on traditional, signature based detection. Most antivirus software now contains layers of prevention-based technologies intended to detect and neutralize 'unknown' malware until such time as a signature can be created. Example technologies include heuristic detection, host intrusion prevention (HIPS), automatic sandboxing and real-time behavior analysis.

Back to the top

Appendix 4 – CIS Versions

Comodo Internet Security is available in three versions – Premium (free), Pro and Complete. The Pro version includes Comodo GeekBuddy (Comodo support experts available 24/7 to fix any problem with your computer) and the Virus Free Guarantee (if your computer becomes damaged as a result of malware and Comodo support services cannot return it to a working condition then we'll pay the costs of getting it repaired. Please see the End User License Agreement for full details). CIS Complete includes GeekBuddy, the Virus Free Guarantee, TrustConnect (a secure Internet proxy service that ensures 128 bit encrypted connectivity from any public wireless hotspot) and a Comodo Online Backup account (50GB of online storage space).

Product	Includes							Price*
	Antivirus	Firewall	GeekBuddy	TrustConnect	Online Storage	Protection Plan Virus Free Guarantee (VFG) / Identity Protection (IDP)	Virus Removal Service	
CIS Premium 7.x	✓	✓	✓	✗	✗	✗	✗	Free
CIS Pro 7.x	✓	✓	✓	✗	✗	✓	✓	$39.99/year or $3.99/month
CIS Complete 7.x	✓	✓	✓	✓ (10 GB / Month)	✓ (50 GB Free. Upgrades available)	✓	✓	$89.99/year or $8.99/month
CAV Free	✓	✗	✗	✗	✗	✗	✗	Free
CAV Advanced	✓	✗	✗	✗	✗	✗	✓	$19.99/year or $3.49/month
Comodo Firewall	✗	✓	✗	✗	✗	✗	✗	Free

* Most CIS products also have discounts for multi-year purchases.

About Comodo

The Comodo companies are leading global providers of Security, Identity and Trust Assurance services on the internet. Comodo CA offers a comprehensive array of PKI Digital Certificates and Management Services, Identity and Content Authentication (Two-Factor - Multi-Factor) software, and Network Vulnerability Scanning and PCI compliance solutions. In addition, with over 10,000,000 installations of its threat prevention products, Comodo Security Solutions maintains an extensive suite of endpoint security software and services for businesses and consumers.

Continual innovation, a core competence in PKI and a commitment to reversing the growth of Internet-crime distinguish the Comodo companies as vital players in the Internet's ongoing development. Comodo, with offices in the US, UK, China, India, Romania and the Ukraine, secures and authenticates the online transactions and communications for over 200,000 business customers and millions of consumers, providing the intelligent security, authentication and assurance services necessary for trust in on-line transactions.

Comodo Security Solutions, Inc.

1255 Broad Street

Clifton, NJ, 07013

United States

Email: EnterpriseSolutions@Comodo.com

Comodo CA Limited

3rd Floor, 26 Office Village, Exchange Quay, Trafford Road, Salford, Greater Manchester M5 3EQ,

United Kingdom.

Tel : +44 (0) 161 874 7070

Fax : +44 (0) 161 877 1767

For additional information on Comodo - visit http://www.comodo.com.

Doc Code: DIST.E.FILE Document Description: Electronic Terminal Disclaimer - Filed	PTO/SB/26 U.S. Patent and Trademark Office Department of Commerce

Electronic Petition Request	**TERMINAL DISCLAIMER TO OBVIATE A DOUBLE PATENTING REJECTION OVER A "PRIOR" PATENT**
Application Number	16141378
Filing Date	25-Sep-2018
First Named Inventor	Glenn Coleman
Attorney Docket Number	LCOM_ENT_CLOUD_US01
Title of Invention	NETWORK ISOLATION WITH CLOUD NETWORKS

☒ Filing of terminal disclaimer does not obviate requirement for response under 37 CFR 1.111 to outstanding Office Action

☒ This electronic Terminal Disclaimer is not being used for a Joint Research Agreement.

Owner	Percent Interest
L3 Technologies, Inc.	100%

The owner(s) with percent interest listed above in the instant application hereby disclaims, except as provided below, the terminal part of the statutory term of any patent granted on the instant application which would extend beyond the expiration date of the full statutory term of prior patent number(s)

10558798

10554475

as the term of said prior patent is presently shortened by any terminal disclaimer. The owner hereby agrees that any patent so granted on the instant application shall be enforceable only for and during such period that it and the prior patent are commonly owned. This agreement runs with any patent granted on the instant application and is binding upon the grantee, its successors or assigns.

In making the above disclaimer, the owner does not disclaim the terminal part of the term of any patent granted on the instant application that would extend to the expiration date of the full statutory term of the prior patent, "as the term of said prior patent is presently shortened by any terminal disclaimer," in the event that said prior patent later:
- expires for failure to pay a maintenance fee;
- is held unenforceable;
- is found invalid by a court of competent jurisdiction;
- is statutorily disclaimed in whole or terminally disclaimed under 37 CFR 1.321;
- has all claims canceled by a reexamination certificate;
- is reissued; or
- is in any manner terminated prior to the expiration of its full statutory term as presently shortened by any terminal disclaimer.

- (◉) Terminal disclaimer fee under 37 CFR 1.20(d) is included with Electronic Terminal Disclaimer request.
- (○) I certify, in accordance with 37 CFR 1.4(d)(4), that the terminal disclaimer fee under 37 CFR 1.20(d) required for this terminal disclaimer has already been paid in the above-identified application.

Applicant claims the following fee status:

- (○) Small Entity
- (○) Micro Entity
- (◉) Regular Undiscounted

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

THIS PORTION MUST BE COMPLETED BY THE SIGNATORY OR SIGNATORIES

I certify, in accordance with 37 CFR 1.4(d)(4) that I am:

- (◉) An attorney or agent registered to practice before the Patent and Trademark Office who is of record in this application

  Registration Number 76913

- (○) A sole inventor
- (○) A joint inventor; I certify that I am authorized to sign this submission on behalf of all of the inventors as evidenced by the power of attorney in the application
- (○) A joint inventor; all of whom are signing this request

Signature	/Jeung Huh/
Name	Jeung Huh

*Statement under 37 CFR 3.73(b) is required if terminal disclaimer is signed by the assignee (owner).
Form PTO/SB/96 may be used for making this certification. See MPEP § 324.

Electronic Patent Application Fee Transmittal

Application Number:	16141378
Filing Date:	25-Sep-2018
Title of Invention:	NETWORK ISOLATION WITH CLOUD NETWORKS
First Named Inventor/Applicant Name:	Glenn Coleman
Filer:	Jeung Huh/Melissa Inzaris
Attorney Docket Number:	LCOM_ENT_CLOUD_US01

Filed as Large Entity

Filing Fees for Utility under 35 USC 111(a)

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
STATUTORY OR TERMINAL DISCLAIMER	1814	1	170	170
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Extension-of-Time:				
Miscellaneous:				
			Total in USD ($)	**170**

Doc Code: DISQ.E.FILE
Document Description: Electronic Terminal Disclaimer – Approved

Application No.: 16141378

Filing Date: 25-Sep-2018

Applicant/Patent under Reexamination: Coleman

Electronic Terminal Disclaimer filed on September 14, 2021

☒ APPROVED

This patent is subject to a terminal disclaimer

☐ DISAPPROVED

Approved/Disapproved by: Electronic Terminal Disclaimer automatically approved by EFS-Web

U.S. Patent and Trademark Office

Electronic Acknowledgement Receipt

EFS ID:	43756674
Application Number:	16141378
International Application Number:	
Confirmation Number:	2440
Title of Invention:	NETWORK ISOLATION WITH CLOUD NETWORKS
First Named Inventor/Applicant Name:	Glenn Coleman
Customer Number:	158266
Filer:	Jeung Huh/Melissa Inzaris
Filer Authorized By:	Jeung Huh
Attorney Docket Number:	LCOM_ENT_CLOUD_US01
Receipt Date:	14-SEP-2021
Filing Date:	25-SEP-2018
Time Stamp:	11:17:18
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	CARD
Payment was successfully received in RAM	$170
RAM confirmation Number	E20219DB17142141
Deposit Account	
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Terminal Disclaimer-Filed (Electronic)	eTerminal-Disclaimer.pdf	41302 fde846c82597c481f82789d50b01c442a5d7213e	no	2
Warnings:					
Information:					
2	Fee Worksheet (SB06)	fee-info.pdf	37945 216e13ab6c2e45e33be89d37a4ce2074584c58d2	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	79247		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: **L3 Technologies, Inc.**	**Confirmation No.: 2440**
Application No.: 16/141,378	**Group Art Unit: 2492**
Filing Date: September 25, 2018	**Examiner: KORSAK, OLEG**

For: NETWORK ISOLATION WITH CLOUD NETWORKS

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

REPLY PURSUANT TO 37 CFR § 1.111

In response to the Official Action dated **March 23, 2021**, reconsideration is respectfully requested in view of the amendments and/or remarks as indicated below:

☐ **Amendments to the Specification** begin on page N/A of this paper.

☒ **Amendments to the Claims** are reflected in the listing of the claims which begins on page 2 of this paper.

☐ **Amendments to the Drawings** begin on page N/A of this paper and include an attached replacement sheet.

☒ **Remarks** begin on page 7 of this paper.

☒ The Commissioner is hereby authorized to charge any fee deficiency, charge any additional fees, or credit any overpayment of fees, associated with this application in connection with this filing, or any future filing, submitted to the U.S. Patent and Trademark Office during the pendency of this application, to Deposit Account No. 50-5519.

IBM - Ex. 1002-Part 10 of 10, Page 3315
IPR2025-00379 (IBM v. Croga)

This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Previously Presented) A host computer system:

a memory; and

a processor configured to:

implement a workspace, wherein the workspace is configured to enable operation of a first set of one or more applications or processes via a first memory space;

implement an isolated computing environment, the isolated computing environment using a host operating system and comprising a sandboxed computing environment that uses a second memory space to enable operation of a second set of one or more applications or processes, wherein the isolated computing environment is configured to access an Internet-based cloud service via at least one application of the second set of one or more applications or processes;

isolate the isolated computing environment from the workspace using an internal isolation firewall;

authenticate the isolated computing environment with an authentication device; and

communicate with a proxy server to access the Internet-based cloud service to allow communication between the at least one application and the Internet-based cloud service when the isolated computing environment has been authenticated.

2. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to access the Internet-based cloud service via one or more of a local area network (LAN), a wide area network (WAN), a cellular network, a Wi-Fi network, a digital subscribers line (DSL) network, a cloud computing network, a data center, or a satellite network.

IBM - Ex. 1002-Part 10 of 10, Page 3316
IPR2025-00379 (IBM v. Croga)

3. (Previously Presented) The host computer system of claim 1, wherein the Internet-based cloud service comprises one or more of an Infrastructure as a Service (IaaS), a Software as a Service (SaaS), or a Platform as a Service (PaaS).

4. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to select the proxy server from a plurality of proxy servers, the plurality of proxy servers comprising one or more of an Internet-based proxy server or a local network-based proxy server.

5. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to determine whether to send web traffic to a local network-based proxy server or an Internet-based proxy server based on a location of the host computer system.

6. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to determine whether to send web traffic to a local network-based proxy server or an Internet-based proxy server based on an identity of a network to which the host computer system is connected.

7. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to determine whether to send web traffic to a local network-based proxy server or an Internet-based proxy server based on an identity of the at least one application running in the isolated computing environment.

8. (Previously Presented) The host computer system of claim 1, wherein the internal isolation firewall is configured to segregate the first memory space from the second memory space, and wherein the processor is configured to enforce policies for communication between the first memory space and the second memory space using the internal isolation firewall.

IBM - Ex. 1002-Part 10 of 10, Page 3317
IPR2025-00379 (IBM v. Croga)

9. (Previously Presented) The host computer system of claim 8, wherein the internal isolation firewall is further configured to generate a message to prompt a user of the host computer system, the message indicating whether to allow communication between the first memory space and the second memory space.

10. (Previously Presented)A cloud computing system comprising:

a cloud computing network comprising a server, wherein the server is configured to:

implement a workspace configured to enable operation of a first set of one or more applications or processes;

implement an isolated computing environment using a host operating system and comprising a sandboxed computing environment and being configured to enable operation of a second set of one or more applications or processes;

isolate the isolated computing environment from the workspace using an internal isolation firewall; and

authenticate the isolated computing environment with an authentication device; and

a proxy configured to control communication between the server and an Internet resource, wherein the proxy is configured to allow communication between the second set of one or more applications or processes running in the isolated computing environment and the Internet resource when the isolated computing environment has been authenticated and to block communication between the first set of one or more applications or processes running in the workspace and the Internet resource.

11. (Previously Presented) The cloud computing system of claim 10, wherein the server is configured to perform at least one processing task on behalf of a client device.

IBM - Ex. 1002-Part 10 of 10, Page 3318
IPR2025-00379 (IBM v. Croga)

12. (Previously Presented) The cloud computing system of claim 11, wherein the proxy is a first proxy, and the server is configured to determine whether to send traffic to the first proxy or a second proxy.

13. (Previously Presented) The cloud computing system of claim 12, wherein the server is configured to determine whether to send traffic to the first proxy or the second proxy based on user-provided credentials.

14. (Previously Presented) The cloud computing system of claim 10, further comprising a border firewall configured to segregate the cloud computing network from one or more Internet destinations.

15. (Previously Presented) The cloud computing system of claim 10, wherein the internal isolation firewall is further configured to allow communication between the workspace and the isolated computing environment on a condition that the communication is in a predetermined format.

16. (Previously Presented) The cloud computing system of claim 11, wherein the server is further configured to receive input data from the client device and send one or more data communications to the client device.

17. (Previously Presented) The cloud computing system of claim 11, wherein the at least one processing task performed on behalf of the client device comprises a processing task for an Infrastructure as a Service (IaaS), a Software as a Service (SaaS), or a Platform as a Service (PaaS).

18. (Canceled)

IBM - Ex. 1002-Part 10 of 10, Page 3319
IPR2025-00379 (IBM v. Croga)

19. (Previously Presented) A method implemented in a host computer system, the method comprising:

implementing a workspace, wherein the workspace is configured to enable operation of a first set of one or more applications or processes via a first memory space;

implementing an isolated computing environment, the isolated computing environment using a host operating system and comprising a sandboxed computing environment using a second memory space to enable operation of a second set of one or more applications or processes, wherein the isolated computing environment is configured to access an Internet-based cloud service via at least one application of the second set of one or more applications or processes;

isolating the isolated computing environment from the workspace using an internal isolation firewall;

authenticate the isolated computing environment with an authentication device; and

communicating with a proxy server ~~in order~~ to access the Internet-based cloud service to allow communication between the at least one application and the Internet-based cloud service when the isolated computing environment has been authenticated.

20. (Original) The method of claim 19, wherein the Internet-based cloud service comprises one or more of an Infrastructure as a Service (IaaS), a Software as a Service (SaaS), or a Platform as a Service (PaaS).

IBM - Ex. 1002-Part 10 of 10, Page 3320
IPR2025-00379 (IBM v. Croga)

REMARKS

Claims 1-17, 19, and 20 are pending in the application. No claims have been amended, canceled, or added. No new matter has been added.

Allowable Subject Matter

Applicant thanks Examiner Korsak for indicating that claims 1-17, 19, and 20 are allowable over prior art.

Double Patenting Rejections over U.S. Patent No. 10,544,475 and 10,588,798

Claims 1-17, 19, and 20 are provisionally rejected under the judicially created doctrine of obviousness-type double patenting as allegedly being obvious over claims 1-20 of U.S. Patent No. 10,554,475 and over claims 1-20 of U.S. Patent No. 10,558,798.[1] However, Applicant is submitting an electronic terminal disclaimer to disclaim claims 1-20 of U.S. Patent No. 10,554,475 and over claims 1-20 of U.S. Patent No. 10,558,798. So, Applicant requests that the double patenting rejection of claims 1-17, 19, and 20 be withdrawn.

Double Patenting Rejections over co-pending Applications

Claims 1-20 are also provisionally rejected under the judicially created doctrine of obviousness-type double patenting as allegedly being obvious over claims of various co-pending applications.[2]

As discussed above, all pending claims are allowable over prior art and an electronic terminal disclaimer is being filed herewith for Patent No. 10,554,475 and U.S. Patent No. 10,558,798. Consequently, the only rejection remaining in this application is the double patenting rejections over following twelve co-pending applications:

1. 16/141,535 – earliest filing date is 09/28/2017
2. 16/144,609 – earliest filing date 09/28/2017

[1] Office Action dated 03/23/2021 at pages 3-5.
[2] *Id.*

IBM - Ex. 1002-Part 10 of 10, Page 3321
IPR2025-00379 (IBM v. Croga)

3. 16/144,761 – earliest filing date 09/28/2017
4. 16/146,149 – earliest filing date 09/28/2017
5. 16/142,810 – earliest filing date 09/28/2017
6. 16/146,490 – earliest filing date 09/28/2017
7. 16/163,796 – earliest filing date 10/23/2017
8. 16/163,884 – earliest filing date 10/23/2017
9. 16/164,342 – earliest filing date 10/23/2017
10. 16/059,381
11. 16/109,275
12. 16/138,696

First, the double patenting rejections that are based on co-pending applicants 1-9 should be withdrawn under § MPEP 804. MPEP § 804 states that "[i]f a provisional nonstatutory double patenting rejection is the only rejection remaining in an application having the earliest effective U.S. filing date … compared to the reference application(s), the examiner should withdraw the rejection in the application having the earliest effective U.S. filing date and permit that application to issue as a patent…." The present application was filed on September 25, 2018 and claims priority to U.S. Provisional Application No. 62/563,237, filed September 26, 2017. Consequently, co-pending applications 1-9 have later U.S. filing date than the present application. So, the double patenting rejections that are based on co-pending applicants 1-9 should be withdrawn under § MPEP 804.

Second, the double patenting rejections that are based on co-pending applications 10-12 should be withdrawn because the Office Action has not demonstrated how each element of claims 1-20 may be found in co-pending applications 10-12. Specifically, MPEP § 804 provides that for an anticipation analysis, "the examiner typically should explain why it would have been obvious to select the claimed species or sub-genus given the genus claimed in the potentially conflicting patent or application." However, in the present case, no such explanation was provided. Rather, the Office Action merely stated that "[a]lthough the claims at issue are not

IBM - Ex. 1002-Part 10 of 10, Page 3322
IPR2025-00379 (IBM v. Croga)

identical, they are not patentably distinct from each other because each element of the claims of the present application is anticipated by the claims of cited applicants and as such are unpatentable for obviousness-type double patenting…."[3] Consequently, Applicant requests that the double patenting rejections based on co-pending applications 10-12 be withdrawn.

For at least the reasons above, Applicant requests reconsideration and withdrawal of the double patenting rejections.

CONCLUSION

Applicant submits that the present application is in condition for allowance and a Notice of Allowance is requested.

Date: September 13, 2021

/Jeung Huh/
Jeung (Jacob) Huh
Registration No. 76,913

Condo Roccia Koptiw LLP
1800 JFK Blvd., Suite 1700
Philadelphia, PA 19103
Telephone: (215) 558-5740
Facsimile: (215) 558-5676

[3] Office Action dated 03/23/2021 at p. 4-5.

IBM - Ex. 1002-Part 10 of 10, Page 3323
IPR2025-00379 (IBM v. Croga)

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	1	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENTS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	36	7150040	B2	2006-12-12	Brownell, David	
	37	7467408	B1	2008-12-16	O'Toole Jr., James W.	
	38	7694328	B2	2010-04-06	Joshi et al.	
	39	7716720	B1	2010-05-11	Marek et al.	
	40	7870153	B2	2011-01-11	Croft et al.	
	41	7926086	B1	2011-04-12	Violleau et al.	
	42	8090797	B2	2012-01-03	Chinta et al.	
	43	8539561	B2	2013-09-17	Gupta et al.	
	44	8566398	B2	2013-10-22	Barnier et al.	
	45	8613070	B1	2013-12-17	Borzycki et al.	
	46	8769268	B2	2014-07-01	Morozov et al.	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	2	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENTS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	47	8931054	B2	2015-01-06	Huynh et al.	
	48	8966464	B1	2015-02-24	Christopher et al.	
	49	9294492	B1	2016-03-22	Martini et al.	
	50	9386021	B1	2016-07-05	Pratt, Ian	
	51	9438634	B1	2016-09-06	Ross et al.	
	52	9465734	B1	2016-10-11	Myrick et al.	
	53	9531715	B1	2016-12-27	Rodgers et al.	
	54	9602524	B2	2017-03-21	Ghosh et al.	
	55	9609026	B2	2017-03-28	Ross et al.	
	56	9680873	B1	2017-06-13	Halls et al.	
	57	9729579	B1	2017-08-08	Marino et al.	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	3	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENTS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	58	9787639	B1	2017-10-10	Sun et al.	
	59	9864600	B2	2018-01-09	Larimore et al.	
	60	9921860	B1	2018-03-20	Banga et al.	
	61	9942198	B2	2018-04-10	Hoy et al.	
	62	10037199	B2	2018-07-31	Hung et al.	
	63	10055231	B1	2018-08-21	Li et al.	
	64	10122703	B2	2018-11-06	Innes et al.	
	65	10397273	B1	2019-08-27	Stickle et al.	
	66	10521612	B2	2019-12-31	Arad et al.	
	67	10931669	B2	2021-02-23	Coleman et al.	

Substitute for 1449/PTO				Application Number	16/141,378
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
(use as many sheets as necessary)				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	4	of	12	Attorney Docket Number	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	68	20020023212	A1	2002-02-21	Proudler, Graeme John	
	69	20020040439	A1	2002-04-04	Kellum, Charles W.	
	70	20020069366	A1	2002-06-06	Schoettger, Chad	
	71	20050144467	A1	2005-06-30	Yamazaki, Takeshi	
	72	20060282795	A1	2006-12-14	Clark et al.	
	73	20070136579	A1	2007-06-14	Levy et al.	
	74	20080016313	A1	2008-01-17	Murotake et al.	
	75	20090055924	A1	2009-02-26	Trotter, Douglas H.	
	76	20090164994	A1	2009-06-25	Vasilevsky et al.	
	77	20090249472	A1	2009-10-01	Litvin et al.	
	78	20090265760	A1	10-22-2009	Zhu, et al.	

Substitute for 1449/PTO				Application Number	16/141,378
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
(use as many sheets as necessary)				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	5	of	12	Attorney Docket Number	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	79	20110154431	A1	2011-06-23	Walsh, Daniel J.	
	80	20110299515	A1	2011-12-08	Robertson et al.	
	81	20120017213	A1	2012-01-19	Hunt et al.	
	82	20120265976	A1	2012-10-18	Spiers et al.	
	83	20130132948	A1	05-23-2013	Hari, et al.	
	84	20130145463	A1	2013-06-06	Ghosh et al.	
	85	20140040979	A1	2014-02-06	Barton et al.	
	86	20140059642	A1	2014-02-27	Deasy et al.	
	87	20140075535	A1	2014-03-13	Soffer et al.	
	88	20140108558	A1	2014-04-17	Borzycki et al.	
	89	20140115646	A1	2014-04-24	Rajgopal et al.	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	6	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	90	20140282890	A1	2014-09-18	Li et al.	
	91	20140344912	A1	2014-11-20	Chapman et al.	
	92	20140351516	A1	2014-11-27	Larimore et al.	
	93	20150281176	A1	2015-10-01	Banfield, Bret	
	94	20160057167	A1	2016-02-25	Bach, Timothy	
	95	20160134549	A1	2016-05-12	Abel et al.	
	96	20160226834	A1	2016-08-04	Dawson, Martin	
	97	20160246974	A1	2016-08-25	Broz et al.	
	98	20160248754	A1	2016-08-25	Jahr, Jeffrey Stephen	
	99	20160292694	A1	2016-10-06	Goldschlag et al.	
	100	20170076092	A1	2017-03-16	Kashyap et al.	

Substitute for 1449/PTO				Application Number	16/141,378
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
(use as many sheets as necessary)				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	7	of	12	Attorney Docket Number	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	101	20170142144	A1	2017-05-18	Weinberger et al.	
	102	20170180413	A1	2017-06-22	Petry et al.	
	103	20170206351	A1	2017-07-20	Jay et al.	
	104	20170208067	A1	2017-07-20	Sriramakrishnan, Sarma	
	105	20170250997	A1	2017-08-31	Rostamabadi et al.	
	106	20170317978	A1	2017-11-02	Diaz-Cuellar et al.	
	107	20170353496	A1	2017-12-07	Pai et al.	
	108	20170359309	A1	2017-12-14	Bolte et al.	
	109	20180027006	A1	2018-01-25	Zimmermann et al.	
	110	20180115519	A1	2018-04-26	Bonomi et al.	
	111	20180139178	A1	2018-05-17	Gan, Yongcun	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	8	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	112	20180196945	A1	2018-07-12	Kornegay et al.	
	113	20180198824	A1	2018-07-12	Pulapaka et al.	
	114	20180203995	A1	2018-07-19	Yuen et al.	
	115	20180234422	A1	2018-08-16	Odom et al.	
	116	20180276396	A1	2018-09-27	Yablokov et al.	
	117	20180330257	A1	2018-11-15	Dodson et al.	
	118	20180367528	A1	2018-12-20	Schwarz et al.	
	119	20190097970	A1	2019-03-28	Coleman et al.	
	120	20190097972	A1	2019-03-28	Coleman et al.	
	121	20190098007	A1	2019-03-28	Coleman et al.	
	122	20190098020	A1	2019-03-28	Martz et al.	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	9	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS

Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	123	20190121961	A1	2019-04-25	Coleman et al.	
	124	20190121962	A1	2019-04-25	Coleman et al.	
	125	20190213325	A1	2019-07-11	McKerchar et al.	

FOREIGN PATENT DOCUMENTS

Examiner Initials*	Cite No	Foreign Document Number[3]	Country Code[2]	Kind Code[4]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines where Relevant Passages or Relevant Figures Appear	T

NON-PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T
	126	"NPL Search Term", Web Search History, 1 page	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	10	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

NON-PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T
	127	AZAB et al., "SICE: A Hardware-Level Strongly Isolated Computing Environment for x86 Multi-Core Platforms", Proceedings of the 18th ACM Conference on Computer and Communications Security, October 2011, pp. 375-388	
	128	BRUCKMAN et al., ""Georgia Computes!": Improving the Computing Education Pipeline", Proceeding of the 40th ACM Technical Symposium on Computer Science Education, 2009, pp. 86-90	
	129	GUPTA et al., "A Secure Architecture Design Based on Application Isolation, Code Minimization and Randomization", IEEE 6th Symposium on Security Analytics and Automation 2013, 2013, pp. 423-429	
	130	HAQ et al., "Design and Implementation of Sandbox Technique for Isolated Applications", IEEE School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China, 2016, pp. 557-561	
	131	HSU et al., "Enforcing Least Privilege Memory Views for Multithreaded Applications", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, October 2016, pp. 393-405	
	132	JAUERNIG et al., "Trusted Execution Environments: Properties, Applications, and Challenges", IEEE Computer and Reliability Societies, Resilient Security, March-April 2020, pp. 56-60	
	133	JUNG et al., "An Architecture for Virtualization-Based Trusted Execution Environment on Mobile Devices", 2014 IEEE 11th International Conference on Ubiquitous Intelligence and Computing and 2014 IEEE 11th International Conference on Autonomic and Trusted Computing and 2014 IEEE 14th International Conference on Scalable Computing and Communications and, Associated Symposia/Workshops, 2014, pp. 540-547	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	11	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

NON-PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T
	134	KOSTIAINEN et al., "Dedicated Security Chips in the Age of Secure Enclaves", IEEE Computer and Reliability Societies; Hardware-Assisted Security, September-October 2020, pp. 38-46	
	135	LE et al., "Computing the Real Isolated Points of an Algebraic Hypersurface", International Symposium on Symbolic and Algebraic Computation, August 24, 2020, 12 pages	
	136	LIU et al., "Thwarting Memory Disclosure with Efficient Hypervisor-Enforced Intra-Domain Isolation", Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, October 2015, pp. 1607-1619	
	137	SRIVASTAVA et al., "Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections", School of Computer Science, Georgia Institute of Technology, 2008, pp. 39-58	
	138	YADAV et al., "Impact & Analysis of Virtual Workspace on Grid Computing", International Journal of Science & Technology (IJST), Vol. 2, Issue 2, April 2012, pp. 24-32	
	139	YEE et al., "Native Client: A Sandbox for Portable, Untrusted x86 Native Code", 2009 30th IEEE Symposium on Security and Privacy, 2009, pp. 79-93	

EXAMINER SIGNATURE			
Examiner Signature		Date Considered	

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	12	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

PTO/AIA/22 (10-20)
Approved for use through 11/30/2020. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a)

Docket Number (Optional): LCOM_ENT_CLOUD_US01

Application Number: 16/141,378

Filed: September 25, 2018

For: NETWORK ISOLATION WITH CLOUD NETWORKS

Art Unit: 2492

Examiner: KORSAK, OLEG

This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above-identified application.

The requested extension and fee are as follows (check time period desired and enter the appropriate fee below):

		Fee	Small Entity Fee	Micro Entity Fee	
☐	One month (37 CFR 1.17(a)(1))	$220	$110	$55	$
☐	Two months (37 CFR 1.17(a)(2))	$640	$320	$160	$
☑	Three months (37 CFR 1.17(a)(3))	$1,480	$740	$370	$ 1480
☐	Four months (37 CFR 1.17(a)(4))	$2,320	$1,160	$580	$
☐	Five months (37 CFR 1.17(a)(5))	$3,160	$1,580	$790	$

☐ Applicant asserts small entity status. See 37 CFR 1.27.

☐ Applicant certifies micro entity status. See 37 CFR 1.29.
Form PTO/SB/15A or B or equivalent must either be enclosed or have been submitted previously.

☐ A check in the amount of the fee is enclosed.

☐ Payment by credit card. Form PTO-2038 is attached.

☐ The Director has already been authorized to charge fees in this application to a Deposit Account.

☑ The Director is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number 50-5519.

☑ Payment made via EFS-Web.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

I am the

☐ applicant.

☑ attorney or agent of record. Registration number 76,913.

☐ attorney or agent acting under 37 CFR 1.34. Registration number _____.

/Jeung Huh/ — Signature

September 14, 2021 — Date

Jeung (Jacob) Huh — Typed or printed name

215-558-5740 — Telephone Number

NOTE: This form must be signed in accordance with 37 CFR 1.33. See 37 CFR 1.4 for signature requirements and certifications. Submit multiple forms if more than one signature is required, see below*.

☑ * Total of 1 (one) forms are submitted.

This collection of information is required by 37 CFR 1.136(a). The information is required to obtain or retain a benefit by the public, which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 6 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Mail Stop PCT, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

Web Search History

date, time	web site	search string
8/13/2020 6:16:55 AM	Google Scholar	isolating browser instance untrusted
8/13/2020 6:16:57 AM	Google Scholar	application isolation in network communication virtual machine firewall
8/13/2020 9:35:17 AM	Google Scholar	isolating untrusted url
8/13/2020 9:35:42 AM	Google Scholar	isolating untrusted url in sandbox

SICE: A Hardware-Level Strongly Isolated Computing Environment for x86 Multi-core Platforms*

Ahmed M. Azab
North Carolina State University
amazab@ncsu.edu

Peng Ning
North Carolina State University
pning@ncsu.edu

Xiaolan Zhang
IBM T.J. Watson Research Center
cxzhang@us.ibm.com

ABSTRACT

SICE is a novel framework to provide hardware-level isolation and protection for sensitive workloads running on x86 platforms in compute clouds. Unlike existing isolation techniques, SICE does not rely on any software component in the host environment (i.e., an OS or a hypervisor). Instead, the security of the isolated environments is guaranteed by a trusted computing base that only includes the hardware, the BIOS, and the System Management Mode (SMM). SICE provides fast context switching to and from an isolated environment, allowing isolated workloads to time-share the physical platform with untrusted workloads. Moreover, SICE supports a large range (up to 4GB) of isolated memory. Finally, the most unique feature of SICE is the use of multi-core processors to allow the isolated environments to run concurrently and yet securely beside the untrusted host.

We have implemented a SICE prototype using an AMD x86 hardware platform. Our experiments show that SICE performs fast context switching (67 μs) to and from the isolated environment and that it imposes a reasonable overhead (3% on all but one benchmark) on the operation of an isolated Linux virtual machine. Our prototype demonstrates that, subject to a careful security review of the BIOS software and the SMM hardware implementation, current hardware architecture already provides abstractions that can support building strong isolation mechanisms using a very small SMM software foundation of about 300 lines of code.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Design, Security

Keywords

Isolation, Trusted Computing, Virtualization Security

*We would like to thank our shepherds Úlfar Erlingsson and Leendert van Doorn for helping us improve the quality of the paper, and Adrian Perrig and Jonathan McCune for their helpful discussion. This work is supported by U.S. National Science Foundation (NSF) under grant 0910767, the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI)), and an IBM Open Collaboration Faculty Award.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCS'11, October 17–21, 2011, Chicago, Illinois, USA.
Copyright 2011 ACM 978-1-4503-0948-6/11/10 ...$10.00.

1. INTRODUCTION

In the last few years, a significant portion of the IT industry has moved toward cloud computing. However, cloud computing services, particularly those relying on hardware sharing, have been the source of major security concerns.

Firstly, the owner of a workload that runs inside the cloud needs to trust the cloud service provider. This trust is imposed by the commodity x86 hardware architecture that gives full memory access to the highest privileged software, which is typically an OS or a hypervisor. Secondly, recent attacks [16, 32] and vulnerability reports [23, 24] show that hypervisors are subject to security exploits and can be compromised. These attacks form a major threat to users who intend to run their workloads beside other potentially malicious ones inside the cloud.

Thus, a need emerges for a solution that provides strong isolation to workloads running in the cloud, yet still allowing hardware sharing to reduce the operating costs. To provide strong isolation, we need to minimize the code base that is granted full access to the memory of running workloads. Thus, we can minimize the exposure to security vulnerabilities that can evade the isolation provided to these workloads. Moreover, this code base should be provided with enhanced security protection and the ability to attest to its integrity.

To achieve this objective, we introduce a prototype system that provides a strongly isolated execution environment, which relies on a trusted computing base (TCB) composed of the hardware, the BIOS, and the System Management Mode (SMM).

Our prototype represents a test system that aims to explore the capability of current hardware platforms in providing more secure isolated environments. We demonstrate that current hardware architecture already provides abstractions that can support strong isolation. Moreover, we show that building strong isolation mechanisms on top of those abstractions requires a very small software foundation of about 300 lines of code (LOC), which tremendously reduces the TCB size compared with previous techniques.

Since the SMM was neither designed nor implemented with high-assurance security mechanisms in mind, detailed security reviews by both CPU and platform vendors would be necessary to verify that current SMM implementations

IBM - Ex. 1002-Part 10 of 10, Page 3339
IPR2025-00379 (IBM v. Croga)

are properly done to support such strong isolation guarantees provided by our prototype in practice.

1.1 Previous Attempts

There have been a few recent attempts to tackle the problem of isolating sensitive workloads while eliminating the host OS or hypervisor from the TCB of these workloads. These attempts can be divided into two main categories: (1) microhypervisor-based approaches, and (2) hardware-based approaches.

Microhypervisor-based Approaches: These approaches rely on a thin, privileged software layer (i.e., a thin hypervisor) to provide the required isolation for sensitive workloads. Among the notable research efforts in this direction are NOVA [27] and Trustvisor [19].

NOVA proposes to replace current hypervisors with a microhypervisor that is around 9 KLOC in size. Despite having a small TCB compared to commodity hypervisors, NOVA is still responsible for several management tasks (e.g., address space management, interrupt and exception handling, and communication between the running workloads). Thus, its TCB is still relatively complicated.

Trustvisor minimizes the code base of the microhypervisor even further (about 2 KLOC for its core functions), so that it can be used for isolation purposes only. However, Trustvisor is only designed to handle workloads with a small code base and only supports systems with a single processor core. This makes it unsuitable for typical compute clouds.

An effort closely related to these microhypervisor-based approaches is seL4 [15]. seL4 proposes a technique to formally verify a microkernel, which is around 8.7 KLOC, to avoid security vulnerabilities. Although this microkernel can be used for isolation purposes, the formal verification process imposes several restrictions on the microkernel functionality. Thus, it cannot be extended to fully support all the functionalities required from micorhypervisors yet.

Hardware-based Approaches: These approaches rely on hardware security extensions to provide isolation for sensitive workloads. The main advantage of these approaches is the enhanced protection for the isolated workloads (compared with software based techniques).

One notable research effort in this direction is Flicker [20], which is a system that uses the late launch capability to run a secure verifiable workload. However, the late launch capability, provided by both Intel [11] and AMD [1], incurs significant overhead (in the magnitude of hundreds of milliseconds) on every context switch to the isolated environment. Hence, it cannot provide a practical solution for cloud computing environments.

Another effort, NoHype [14], also relies on hardware isolation through assigning dedicated processor cores to each running workload. However, NoHype still relies on a thin software layer to achieve the required protection and manage the hardware resources (e.g., page tables). Moreover, it requires architectural changes to processors and hardware peripherals, which are slow to realize.

It should be noted that all of the above techniques are research prototypes that make several simplifying assumptions about their target platforms. These assumptions may hinder the applicability or even weaken the security guarantees provided by these techniques. For instance, the formal verification introduced by seL4 does not include the firmware or the SMM code, which could negate all seL4 guarantees. Thus, a practical deployment of any of these research prototypes, including the prototype we present in this paper, requires a comprehensive consideration of the target platform that includes the full hardware and firmware specifications.

1.2 Introducing SICE

In this paper, we present SICE, which stands for Strongly Isolated Computing Environment, a framework that provides a hardware-level isolated execution environment for x86 hardware platforms. SICE's main objective is to minimize the TCB required to create an isolated execution environment on commodity x86 platforms. This isolated environment can be used to host a security sensitive workload.

SICE achieves this objective by relying on a TCB that is only composed of the hardware, the BIOS, and the SMM. The TCB, which is fundamentally different from previous research, gives SICE principal advantages over both microhypervisors and hardware-based isolation techniques. We summarize these advantages below.

Smaller Attack Surface: SICE utilizes the hardware protection provided by commodity x86 processors for the SMM and the memory that hosts its code, which is called System Management RAM (SMRAM). There are two fundamental differences between the SMM and microhypervisors.

First, the SMM can only be triggered by a single interface, which is to invoke a System Management Interrupt (SMI). In SICE, the SMI handler is required to execute one of only four functions upon receiving an SMI, which are to create, enter, exit and terminate an isolated environment. Implementing these functions requires the system to run *briefly* in the SMM. Moreover, the SMI handler is not required to handle any other interrupts because all interrupts are disabled upon entering the SMM mode. The SMI handler is also *not* responsible for managing the communication between running workloads. On the other hand, microhypervisors reside at the system's highest privileged level. Thus, they have to handle all system events (e.g., hypercalls, interrupts and exceptions). They are also required to manage the communication channels (e.g, shared memory pages) between different workloads.

Second, SICE uses the SMRAM to provide the needed memory isolation. After the SMRAM is initialized by the BIOS, it can be locked so that no software can access its contents except for the SMM code. The SMM code can manage the SMRAM through modifying only two registers, which has a huge impact on decreasing the size of the TCB. In contrast, microhypervisors that rely on hardware virtualization have to manage a different set of page tables for every isolated environment to provide memory protection.

To sum up, SICE's SMM code base is better protected and less complicated than any microhypervisor. Moreover, this code does not provide any functionality other than the required isolation, which results in a very small code base. For instance, our prototype SMI handler consists of around 300 LOC (excluding cryptographic libraries). This is around an order of magnitude less than current microhypervisors (e.g., Trustvisor and NOVA). As discussed in [15], this is a significant difference when it comes to verification cost. For instance, using the industry rules-of-thumb of $10K per LOC for common criteria certification as a guideline for the cost of code verification, the cost for verifying 500LOC is $5M versus $100M for 10 KLOC.

Compatibility with Existing Software Systems: Un-

IBM - Ex. 1002-Part 10 of 10, Page 3340
IPR2025-00379 (IBM v. Croga)

like isolation techniques that monopolize the highest privileged execution level of the target platform (e.g., Trustvisor), SICE does not exclude running legacy workloads (e.g., a hypervisor with multiple VMs) on the same physical platform. In other words, a platform using SICE can offer isolated environments, and at the same time accommodate legacy virtualization software.

Feasible Hardware-based Isolation: SICE uses existing hardware features to provide the required isolation. We have successfully implemented a SICE prototype using a commodity AMD processor. Thus, it does not require fundamental changes to current hardware architecture.

Moreover, our performance evaluation shows that SICE performs a secure context switching with an isolated environment that is four orders of magnitude faster than systems that rely on the late launch capability (e.g., Flicker [20]). It also uses multi-core processors to allow isolated workloads to run in parallel to a legacy hypervisor or OS. Thus, SICE avoids the two main drawbacks of using late launch, which are the high performance overhead, and the dedication of all system resources to only one isolated workload.

1.3 SICE Overview

SICE introduces novel techniques that allow the isolated workloads to run in parallel with a host OS or hypervisor. For convenience, we refer to the OS or hypervisor along with all the software running on it as the *legacy host*, a strongly isolated computing environment, which supports an isolated workload, as an *isolated environment*, and the code that manages the isolation between these environments as the *SMI handler*, or simply SICE.

The SMI handler, which represents the TCB of the isolated environments, resides inside the SMRAM. It is the only part of our framework that executes in the SMM. The SMI handler is responsible for two main tasks: 1) maintaining the memory isolation of the isolated environments, 2) securely initializing the isolated environments and attesting to their integrity. In SICE, these tasks require the SMI handler to run for a very short time.

An isolated environment is composed of two components: an isolated workload and a security manager. The isolated workload is a user-provided system that runs in the isolated environment. It can be any software, ranging from a single program (e.g., a program that manages secret keys) to a complete VM (e.g., a VM that runs a web server).

The security manager is a thin software layer that has limited functionalities such as handling exceptions and managing page tables. It is mainly responsible for confining the isolated workload. Due to the commodity hardware limitation on SMRAM size, SICE's unique hardware-level isolation is not used to protect the legacy host from the isolated environments. Thus, SICE uses the security manager to prevent the isolated workload from accessing the memory of the legacy host. A separate copy of the security manager is generated by SICE for every isolated workload running on the system. Both the security manager and the isolated workload run after the system returns from the SMM.

Though SICE requires the legacy host to trust the security managers, it does not weaken the hardware-level isolation provided to the isolated workloads. SICE uses SMM to protect isolated environments from the legacy host. Even if a malicious workload (in one isolated environment) compromises its own security manager and consequently the legacy host, it will not be able to compromise any other isolated environment running on the same platform.

SICE's protection of the legacy host may appear to be equivalent to those provided by microhypervisor-based approaches such as NOVA. Indeed, the security manager, which is a thin privileged software layer, is similar to a microhypervisor in terms of its required tasks and code size. However, SICE also provides hardware-level, stronger protection for the isolated environments, which are not available in microhypervisor-based approaches.

SICE provides two operating modes: *time-sharing mode* and *multi-core mode*. In both modes, the SICE philosophy is based on using the isolated environments to run security sensitive workloads, while the legacy host is used for running less sensitive workloads and managing hardware peripherals. In a typical execution scenario, a communication channel is used so that the legacy host provides hardware services (e.g., networking) to the isolated environments. The communication channel can be established using a shared memory outside of the memory range protected by SICE. Communication channels are not controlled or managed by the SMI handler. Thus, the code responsible for managing the communication is not a part of the TCB.

In the time-sharing mode, time multiplexing is used to share the hardware platform between the isolated environments and the legacy host. During the environment switching, SICE guarantees a fresh start of the processor, complete memory isolation between these environments, and a timely switching that does not largely impact the performance.

In the multi-core mode, SICE assigns one or more processor cores to each isolated environment, while the other cores are used to run the legacy host. SICE guarantees isolation of both the processor cores and the memory dedicated to each of the concurrently running environments.

In both modes, SICE attests to the integrity of each isolated environment to remote users, while avoids revealing sensitive information about the workload to the legacy host.

The current SICE design relies on hardware features provided by AMD processors. Some of these features are not currently supported by Intel processors, particularly the ability to resize the SMRAM at runtime and defining a separate SMRAM for each processor core. Proposing alternative techniques to implement SICE on Intel platforms will require further research. We discuss these issues in Appendix A.

We implement a prototype of SICE on an IBM LS22 blade server that uses AMD processors. We use our SICE prototype to run a complete Linux VM in the isolated environment. Our experimental evaluation shows that the time required to enter and exit an isolated environment using SICE is around 67 μs. We conduct experiments to evaluate the performance of the isolated VM. In the multi-core mode, SICE incurs a low (under 3%) overhead on VM operations that do not require frequent communication with the legacy host. The time-sharing mode shows a higher overhead (around 10%) due to the time for the context switching. A test dedicated to measure the performance of network emulation shows a higher overhead. However, this overhead is expected to decrease using an optimized implementation of network emulation for isolated VMs.

1.4 Summary of Contributions

We make several technical contributions in this paper:

- We provide a complete and feasible solution to share

IBM - Ex. 1002-Part 10 of 10, Page 3341
IPR2025-00379 (IBM v. Croga)

hardware resources with an isolated execution environment that does not rely on any host software. Instead, it relies on a TCB that is around an order of magnitude smaller than the state-of-the-art systems.

- We provide a novel technique that allows concurrent execution of the isolated environments with the untrusted host environment.
- We provide attestation to the integrity of the isolated workloads without revealing sensitive information about the workloads to untrusted host software.
- We implement a prototype of SICE on an AMD platform. We use this prototype to evaluate the performance overhead introduced by SICE.

The rest of this paper is organized as follows. Section 2 provides background information on SMM. Section 3 discusses our assumptions and threat model. Section 4 presents SICE in detail. Section 5 presents our prototype implementation. Section 6 presents our experimental evaluation. Section 7 presents related work, and Section 8 concludes this paper with some future research directions. Finally, The Appendix discusses SICE's portability to Intel platforms.

2. BACKGROUND

In this section, we briefly give some background information on the System Management Mode (SMM).

Commodity x86 architecture supports the SMM as one of its operating modes. The processor enters the SMM when receiving an SMI. Upon an SMI, the processor saves its state to a dedicated state save map and switches to the SMM. To return from the SMM, the special instruction `RSM` restores the saved processor state and resumes normal execution.

SMM code is stored in a designated memory called SMRAM. To provide protection of the SMM code and data, both AMD and Intel provide the capability of locking the SMRAM. When the SMRAM is locked, all accesses to it, except from within the SMM, are prevented. All interrupts, including non-maskable ones, are disabled upon entering the SMM. Thus, no other code running on the system can interfere with the SMI handler.

Current hardware can support up to 4 GB of SMRAM. There are two types of SMRAM: the `ASeg`, which is located at a fixed low address, and the `TSeg` which is located at a variable base and has a variable size. SICE uses `TSeg` due to the flexibility it offers. Defining the SMRAM location and range differs slightly between AMD and Intel architectures. In the following, we take a closer look at AMD architecture, on which this paper focuses.

The AMD architecture defines the `TSeg` memory range using two Model Specific Registers (MSRs) that are local to each processor core. The first is `SMM_Addr`, which specifies the SMRAM base address, while the second is `SMM_Mask`, which specifies its range. Since MSRs are local to each processor core, all cores have to set their MSRs to provide complete protection of the SMRAM.

The `SMMLOCK` bit in the `HWCR` MSR can be set to prevent changing the `TSeg` range. Moreover, the AMD architecture uses a password mechanism to protect the modification of the `SMMLOCK` bit. When the SMRAM is unlocked, writes to the `SMM_KEY` MSR set the 64-bit password. When the SMRAM is locked, writing the correct password to the same MSR clears the `SMMLOCK` bit.

3. THREAT MODEL AND ASSUMPTIONS

Threat Model: SICE aims at defending against all malicious activities by software running in the legacy host that are targeted at compromising the isolation offered to the isolated environment (e.g., malicious activity that result from exploiting a vulnerability in the hypervisor). Specifically, SICE protects the isolated environment from all types of unauthorized memory accesses or any modification to its execution environment. The protection starts from the moment the isolated environment is initialized by SICE. Upon initialization, the initial image of the isolated workload, which is loaded by the legacy host, is measured so that SICE can further attest to its integrity.

We consider the following attacks out of the scope of this paper: Attacks aimed at the availability of SICE (e.g., denying its network access), and those that directly exploit vulnerabilities of the isolated environment through legitimate communication channels. Such attacks are not specific to the isolation mechanism, and should be addressed by other techniques such as keep-alive messages and patching the vulnerabilities. Moreover, SICE is not responsible for securing the hardware services provided by the host. Thus, the isolated workload should use other techniques to achieve this objective (e.g., encrypting its network traffic).

Side-channel attacks are also out of the scope of this paper. Adopting SICE prevents cache side-channels because the hardware automatically clears the cache upon entering and exiting from the SMM. However, these attacks in general are not unique to our approach and require further research.

Assumptions: We assume that our platform is physically secure (e.g., locked in a server room) so that an adversary cannot launch any hardware attack. Moreover, we assume that the target platform is equipped with trusted computing hardware, including the Core Root of Trust Measurement (CRTM) and Trusted Platform Module (TPM) [29]. This allows the attestation to the integrity of key software components (e.g., the SMI handler).

We assume that the SMM is properly isolated from other software running on the system and the hardware provides the SMRAM with proper isolation from all unauthorized memory accesses (e.g., cache poisoning, Direct Memory Access (DMA)). Recent incidents showed that attackers were able to subvert the SMRAM using cache poisoning [6, 33]. Fortunately, such attacks cannot be mounted on AMD platforms due to its SMRAM cache protection and can be easily defeated on Intel platforms using proper setting of the System Management Range Register (SMRR) [13].

4. SICE DESIGN

In this section, we present the design of SICE. The objective is to enable hardware-level strongly isolated computing environments that run in parallel with the legacy host on the same hardware platform.

Implementation requirements: To implement SICE, the SMI handler needs to be modified to include SICE's code. Recently, most hardware vendors use the BIOS to lock the SMRAM to prevent potential SMM misuse. Thus, SICE requires hardware vendors to allow adding its code to the SMI handler before locking the SMRAM.

The legacy host (e.g., the hypervisor) is required to add an interface that invokes an SMI to trigger SICE. Hardware management functions provided by the legacy host (e.g.,

IBM - Ex. 1002-Part 10 of 10, Page 3342
IPR2025-00379 (IBM v. Croga)

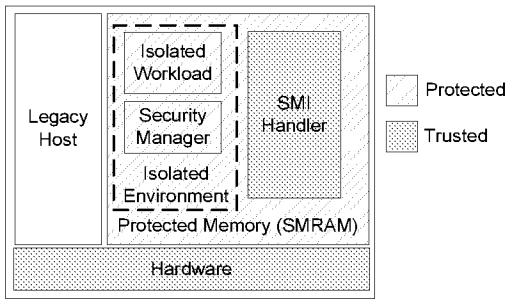


Figure 1: SICE Architecture

hardware device drivers) should be modified to use this interface to provide its services to the isolated environment. Note that the legacy host may refuse to invoke an isolated environment or deny it some services, thus threatening its availability. However, these attacks are easily detectable by SICE and its remote users (e.g., from the lack of response of the isolated workload), and can then be thwarted by replacing the faulty legacy host.

SICE Architecture: Figure 1 shows the architecture of SICE. It consists of three components: The SMI handler, the security manager, and the isolated workload. The security manager and the isolated workload constitute the isolated environment. SICE enables the isolated environment and the legacy host to share the physical platform's resources.

The security manager is a thin software layer that confines the isolated workload. It has limited functionalities such as handling exceptions and managing page tables. The initial image of the security manager should be loaded to the SMRAM, along with the SMI handler, upon the system initialization. Thus, we assume trust in both the SMI handler and the security manager based on the system's trusted boot, which can be attested to later using the TPM.

The isolated workload is a user-provided system that runs in the isolated environment. It can be any software, ranging from a single program to a complete VM. The only restrictions on this software are that it does not use more than 4GB of memory (due to SMRAM constraints) and that it does not manipulate hardware peripherals directly.

When the isolated environment is not actively running, its code and data are securely stored in the SMRAM (as shown in Figure 1). However, the SMM is a limited execution mode that is not suitable for running the isolated environment. For instance, code execution inside the SMM is slower than the protected mode [2]. Moreover, some hardware functionalities (e.g., starting a hardware assisted VM) are not supported in the SMM. Thus, SICE only uses the SMM to prepare and enter the isolated environment. SICE uses novel techniques to move the security manager and the isolated workload out of the SMRAM after the isolated environment is initialized.

In the following, we discuss SICE in a *time-sharing mode*, where the legacy host and the isolated environment time share the physical platform. We then present the *multi-core mode*, where the legacy host and the isolated environment run in parallel using multi-core processors.

4.1 Time-sharing Mode

In the time-sharing mode, SICE provides two important features: (1) fast context switch between the legacy host and the isolated environment (in the magnitude of few tens of microseconds), and (2) large protected memory range for the isolated workload (up to 4GB). The fast context switch allows the isolated workload to receive its input data and send its output to and from the legacy host without posing significant overhead on the system performance. The large protected memory range enables the isolated environment to securely keep its state across the context switches.

4.1.1 Initializing the Isolated Environment

Both the security manager and the SMI handler are initialized when the physical platform is booted. When an isolated workload is ready to be started, the legacy host first loads the initial image of this workload to a specific memory range. Then, it triggers SICE using an SMI. The SMI handler then measures the initial image of the isolated workload and copies it to the protected SMRAM. Details on measuring and attesting to the isolated environment are discussed in Section 4.3.

4.1.2 Entering the Isolated Environment

Figure 2 shows the process of entering the isolated environment. Whenever the legacy host requires the isolated workload to run, it triggers another SMI. The SMI then switches the processor's execution environment to the SMM and the execution jumps to the SMI handler. The SMI handler then prepares the isolated environment by changing the saved processor state so that the security manager, instead of the legacy host, runs after the processor returns from the SMM. The SMI handler also stores the processor state of the legacy host so that it can resume execution after the isolated environment finishes its execution.

In the time-sharing mode, SICE gives the isolated environment full control of the physical platform as soon as it runs. Thus, it changes the processor state (e.g., interrupt descriptors, system call handlers) so that only the security manager will have control after the SMI handler returns.

Unfortunately, the SMM cannot change most of the critical processor states (e.g., the interrupt descriptor table (IDT) register and the CR3 register). To overcome this challenge, the SMI handler relocates the code pointed to by the current processor state through modifying the page tables. Since the CR3 register is not writable, the SMI handler modifies the first level page table pointed to by the CR3 so that it directly points to the security manager. It also flushes the Translation Lookaside Buffer (TLB) to avoid potential race conditions with the cached page tables.

Before the SMI handler returns, we need to ensure that both the security manager and the isolated workload are accessible to the processor after it returns from the SMM. As shown in Figure 1, both the security manager and and the isolated workload are stored in the SMRAM before the isolated environment is entered. These two components need to be moved out of the SMRAM so that they can run after the processor returns from the SMM.

A straightforward solution is to copy the security manager and the isolated workload out of the SMRAM to an unprotected memory. However, this will introduce an unacceptable performance overhead.

To address this problem, we use the ability of AMD processors to resize the SMRAM during the system runtime. This feature, discussed in Section 2, relies on a password-protected mechanism to clear the SMM lock so that the `SMM_Addr` and the `SMM_Mask` registers can be updated.

IBM - Ex. 1002-Part 10 of 10, Page 3343
IPR2025-00379 (IBM v. Croga)

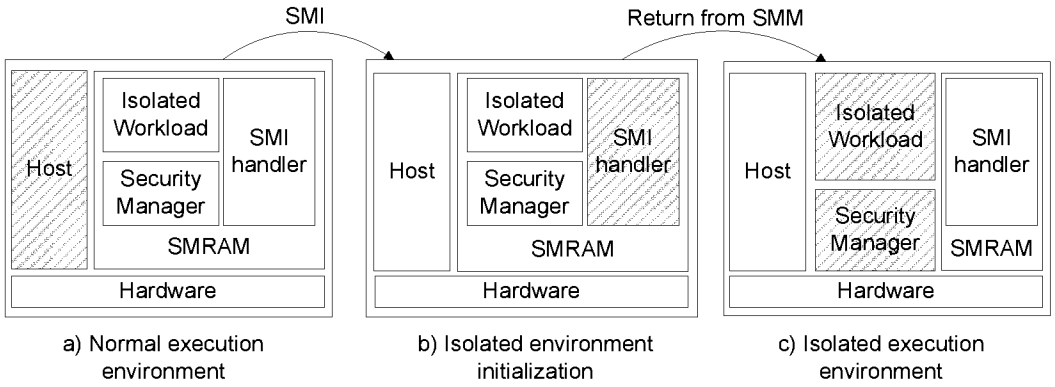


Figure 2: Entering SICE from the legacy host (actively running components are shaded): a) An SMI triggers SICE operations. b) The SMI handler prepares the isolated environment and the new SMRAM layout. c) After entering the isolated environment, the security manager prevents the isolated workload from accessing the legacy host memory.

Upon the initialization of the system, SICE generates a 64-bit random number, sets it as the password for locking/unlocking the SMRAM, and stores it inside the SMRAM. Before entering the isolated environment, the SMI handler uses this password to unlock the SMRAM and modify its protection range (by modifying the `SMM_Addr` and the `SMM_Mask` registers) to exclude both the security manager and the isolated workload. Thus, they can be accessed by the processor after the SMI handler returns.

As shown in Figure 2, the SMRAM protection remains for the SMI handler while the isolated workload runs. This guarantees that the SMI handler (including the SMRAM password) can never be accessed by any other software running on the system, including the isolated workload.

A subtle issue needs to be clarified. To guarantee full control over the system, the isolated environment should run in the highest privileged level after the SMI returns. However, this requires the legacy host to trigger the SMI from within the highest privileged level. In other words, the SMI cannot be triggered by a guest VM. Otherwise, the isolated environment will return to the context of a guest VM controlled by the legacy host.

To verify that the SMI is triggered by the highest privileged level, SICE requires the legacy host to disable virtualization by clearing the `SVME` bit in the `EFER` register before triggering the SMI. The SMI handler verifies that this bit is clear. Otherwise, it will not enter the isolated environment and keep the SMRAM protection for its memory.

4.1.3 *Managing the Isolated Workload*

Since the isolated workload is provided by the user, it is not trusted by SICE. Hence, it should not be allowed to tamper with the hardware configuration or access memory regions that belong to the security manager or legacy host.

Unfortunately, commodity hardware architecture cannot assign more than 4GB of memory to the SMRAM. Thus, using the SMRAM to protect the legacy host memory from the isolated workload will not be a feasible solution due to its restriction on the memory capacity of the legacy host.

To address this challenge, the security manager plays the role of a hypervisor and runs the isolated workload in the context of a guest VM. Therefore, the isolated workload's execution environment is restricted so that it cannot execute privileged instructions. Moreover, the security manager crafts the page tables of the isolated workload so that it can access a limited specific range of physical memory. Thus, the isolated workload cannot access any memory range that belongs to either the security manager or the legacy host.

4.1.4 *Exiting the Isolated Environment*

To allow the legacy host and the isolated environment to efficiently share the hardware resources, SICE provides a technique to securely and properly exit the isolated environment and return the execution back to the legacy host.

To return back to the legacy host, the isolated environment triggers an SMI. The SMI handler then: (1) Clears all general purpose registers, (2) flushes all cache levels, (3) uses the SMM password to change the SMRAM range so that it covers both the security manager and the isolated workload, and (4) restores all changes done to the legacy host page tables before the isolated environment was initialized.

The first two steps ensures that no sensitive data is leaked to the legacy host. The third step ensures the protection of the isolated environment's memory. Finally, the last step ensures that the legacy host resumes its operations correctly.

4.1.5 *Terminating the Isolated Workload*

When the isolated workload finishes execution, it needs to be securely terminated so that no confidential information are leaked to the legacy host. Thus, isolated workload termination requests must be forwarded to the SMI handler, which securely erases the memory belonging to the isolated workload and then removes it from the protected SMRAM.

4.2 Multi-core Mode

The SICE multi-core mode enables the legacy host and the isolated environments to run in parallel on different processor cores to better utilize the hardware resources.

Before presenting our solution, we first give some background information on multi-core processors. As shown in Figure 3, multi-core processors are equipped with one or more processing nodes. Each node has its own memory control hub (north bridge) and one or more processor cores. North bridge configuration registers can be accessed by any core on any node. On the other hand, each processor core has its own general-purpose registers, MSRs, APIC, and two

IBM - Ex. 1002-Part 10 of 10, Page 3344
IPR2025-00379 (IBM v. Croga)

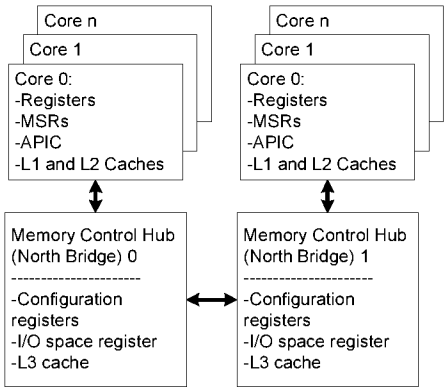


Figure 3: General multi-core processors architecture

levels of cache. These components can only be accessed by their designated core. Among these core-dependent components are the `SMM_Addr` and `SMM_Mask` MSRs that define the SMRAM memory range. Thus, each processor core can have its own protected SMRAM memory range that is independent from the SMRAM memory range defined on other processor cores that share the same node. SICE utilizes this architecture to provide the necessary protection for the isolated environment in the multi-core mode.

For simplicity, we assume a platform with two cores in our discussion: the *host core* that runs the legacy host, and the *isolated core* that runs the isolated environment. However, the same techniques can be used to assign more than one core to either the legacy host or the isolated environment. In general, two processor cores that belong to different execution environments can be either co-located on the same processing node or located among different nodes.

In the multi-core mode, SICE initializes the isolated environment in the same way as the time-sharing mode (i.e., by triggering an SMI in the legacy host). The SMI switches the execution environment of all processor cores that exist on the target platform to the SMM. The SMI handler performs a different task on each processor core. On the isolated core, the SMI handler prepares the isolated environment and returns to the security manager using the techniques described in Section 4.1.2. Meanwhile, on the host core, the SMI handler returns directly to the legacy host so that it can resume its operations at the same time.

Since both the legacy host and the isolated environment run in parallel on different processor cores, no context switching between these environments is needed.

To ensure the isolation between the legacy host and the isolated environments, SICE prevents the legacy host from affecting the execution of the isolated environment, and the isolated workload (excluding the security manager, which needs to be trusted by the legacy host) from affecting the execution of the legacy host. In particular, we need to prevent the interference through explicit inter-core communication as well as those through modifying each other's memory.

4.2.1 Handling Inter-core Communication

Communication between processor cores, which is done through Inter-Processor Interrupts (IPIs), can modify the execution environment on the recipient code by changing the execution path. This can pose a risk on the integrity of the isolated environment. Therefore, all IPIs should either be blocked or securely handled by the recipient core.

There are two types of IPIs: (1) Maskable IPIs that can be blocked by the recipient core's Advanced Programmable Interrupt Controller (APIC), and (2) non-maskable IPIs (e.g., NMI, SMI, startup, and INIT) that cannot be blocked.

Upon initializing the isolated environment, the SMI handler disables all maskable interrupts on the isolated core so that the host core cannot interfere with the isolated core. The security manager, which runs as a thin hypervisor on the isolated core, keeps these interrupts disabled.

On the other hand, non-maskable interrupts cannot be blocked. Moreover, a specific non-maskable interrupt, the startup interrupt, resets the processor core to start execution at a low physical memory address. Intuitively, this address can be modified by the host core to alter the course of execution on the isolated core.

To overcome this problem, SICE relies on the Global Interrupt Flag (GIF) introduced by AMD. When the GIF is clear, all interrupts, including non-maskable ones, are ignored or held pending. The security manager thus clears the GIF of the isolated core when the isolated environment is entered, and sets it only after the isolated environment completes its execution and all memory protection measures are taken.

The security manager then runs the isolated workload in the guest VM mode. Upon entering the VM mode, both global interrupts and maskable interrupts are re-enabled. Hence, the isolated workload can receive all interrupts including both IPIs and local processor interrupts. However, the received interrupts will cause a VM exit and jump to the security manager rather than modifying the execution environment.

Global interrupts are automatically disabled again before exiting the VM mode and jumping to the security manager. The security manager identifies the reason that causes the VM exit. If it is an interrupt that aims to modify the execution environment (e.g., INIT interrupt), then it indicates a malicious activity from the host core and all memory protection measures are taken. However, if the VM exit is caused by a local interrupt or an IPI, the interrupt is forwarded back to the isolated workload. This architecture allows the isolated workload to use IPIs as method of signaling, which is required to build a communication channel with the legacy host. This channel can be used to provide hardware services offered by the legacy host to the isolated workload.

On the other hand, SICE relies on hardware virtualization to prevent the isolated workload from perturbing the legacy host. The isolated workload, which runs in a guest VM, is not allowed to directly access the APIC to send any interrupts that can perturb the execution of the host core.

4.2.2 Memory Isolation

To prevent the legacy host from accessing the isolated environment's memory, SICE relies on a novel protection called the *memory double-view technique*, shown in Figure 4. This technique relies on the fact that AMD processor's SMRAM is defined based on the *core-dependent* MSRs. Thus, different cores can view the SMRAM differently depending on the values of their own `SMM_Addr` and `SMM_Mask` registers.

As shown in Figure 4, each processor core has its own view of the physical memory. From the host core's perspective, the isolated environment uses a physical memory that lies in the SMRAM memory range. Hence, the legacy host cannot

IBM - Ex. 1002-Part 10 of 10, Page 3345
IPR2025-00379 (IBM v. Croga)

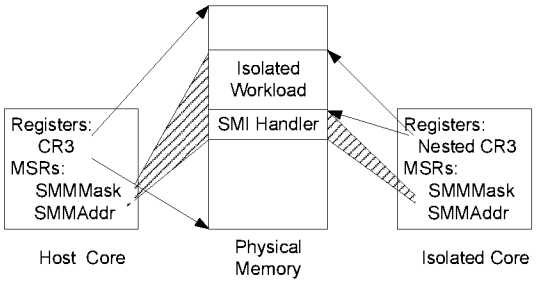


Figure 4: Double view of memory from different processor cores. From the host core's view, the isolated environment is part of the SMRAM. From the isolated workload's view, the SMI handler is protected by the SMRAM and the legacy host is protected by the security manager via hardware assisted paging.

access such memory due to the SMRAM protection. From the isolated core's perspective, the isolated environment lies within a normal memory region that is not part of the SMRAM. Hence, the isolated environment can run normally on the isolated core.

SICE also prevents the isolated workload from accessing the legacy host memory. The security manager, which plays the role of a hypervisor, restricts the isolated workload to its assigned physical memory. Nevertheless, a shared memory, which does not fall in the host core's SMRAM, is mapped to allow communication between the two environments.

4.3 Attestation and Secure Communication

Cloud computing users are required to trust the environment that runs their workloads. In SICE, attesting the integrity of the isolated environment is complicated by the fact that the legacy host has full access to all hardware peripherals and communication channels. In this section we provide a three-step solution to address this challenge.

Attesting to Integrity of SICE: SICE requires the platform to use standard trusted boot [28]. After a typical trusted boot, the TPM securely stores the measurement of the boot process, which includes the SMI handler, the code image of the security manager and the code that loads them and locks the SMRAM. The measurement is stored in TPM special registers that cannot be erased by malicious software.

The TPM can further use its private Attestation Identity Key (AIK) to attest to the integrity of this measurement to remote users. Unlike other system software that is continuously interacting with potential attackers, the SMI handler cannot be modified by any code running on the system and the trust in it can be maintained.

Attesting to Integrity of Isolated Workload: In turn, the SMI handler attests to the integrity of the isolated workload. It measures the isolated workload before it is first invoked and stores the measurement in the secure SMRAM.

To enable the attestation, we adopt an approach we previously used is HyperSentry [2]. Specifically, SICE generates a public/private key pair during system boot. The private key is securely stored in the SMRAM, and the public key is securely stored inside the TPM.

To attest to the integrity of the isolated workload, the SMI handler signs the measurement of the initial workload image using its private key. The remote user accepts the measurement output only if the private key matches the public key stored in, and attested to by, the TPM.

Secure Communication with the Isolated Environment: SICE allows the establishment of a secure communication channel between the isolated workload and its remote owner using standard cryptographic techniques. To enable the remote owner to authenticate the isolated workload, SICE provides a public/private key pair for the isolated workload. The public key is sent to the user as a part of the workload attestation evidence signed by the SMI handler. The private key is directly provided to the isolated workload. Thus, it is never exposed to potentially malicious code running inside the legacy host. On the other hand, the isolated workload should include the public key of its remote owner as a part of its initial image.

With both the security manager and the remote owner being able to validate each other's public keys, we can easily modify, for example, SSL for secure communication without leaking information to the (untrusted) legacy host.

4.4 Security Analysis

Now we discuss the security of the isolated environment, including its confidentiality, integrity, availability, and TCB. We also discuss the security implication of using the SMM.

4.4.1 The Security of the Isolated Environment

Confidentiality: To protect the confidentiality of the isolated environment, SICE prevents potential attackers from accessing its memory. SICE uses the SMRAM to achieve this objective. The hardware protects the SMRAM from access requests made by both the CPU and direct memory access (DMA) capable devices.

In the time-sharing mode, SICE modifies the SMRAM memory range according to the running environment. Whenever the legacy host is running, SICE extends the SMRAM to include the memory of the isolated environment. Thus, this memory is protected from all memory requests.

When the isolated environment is triggered, SICE takes two security measures to guarantee its confidentiality. First, SICE assures that the isolated environment fully controls physical platform. Hence, the legacy host is not active to threaten the isolated environment, given that modern processors do not use cached memory upon entering the SMM to avoid cache poisoning attacks [1, 13]. Second, SICE uses hardware DMA exclusion (e.g., AMD's DMA exclusion vector (DEV) [1] and Intel's VT-d [12]) to prevent DMA capable devices, which could be maliciously programmed, from accessing the protected memory.

In the multi-core mode, SICE's memory double-view technique provides the needed protection. According to architecture manuals, memory access requests made from a processor core are checked against its own SMRAM before being routed. Hence, the host core will not be able to access the protected memory that falls within its own SMRAM range. The same is true for requests to update or retrieve cache entries. Hence, cache poisoning is not possible.

In the multi-core mode, SICE cannot rely on hardware DMA exclusion because its control registers exist in the memory control hub, which is accessible by all processor cores. Thus, SICE relies on the SMRAM protection to prevent DMA access to the isolated environment's memory. Typically, the memory control hub does not allow DMA access to the SMRAM. However, the AMD hardware ar-

IBM - Ex. 1002-Part 10 of 10, Page 3346
IPR2025-00379 (IBM v. Croga)

chitecture manual does not precisely define which processor core's SMRAM range is used to prevent DMA access, particularly when different SMRAM ranges are defined on different cores. Thus, implementing SICE on a specific platform requires verifying which SMRAM range is protected from DMA. The specific processor core(s) that defines this range is the same one that should be used as the untrusted host core. As shown in figure 4, the host core's SMRAM memory range includes the whole memory range used by the running, one or more, isolated environments.

A final threat to the confidentiality of the isolated environment is brute force attacks against the SMM password. For each processor core, SICE generates a 64-bit random value to be used as the SMM password. Thus, it is computationally infeasible to break this password. The random password generation can be done using the TPM.

Integrity: In the time-sharing mode, the legacy host cannot threaten the integrity of the isolated environment because they do not run concurrently. In the multi-core mode, the host core cannot access the registers, MSRs and APIC of the isolated core. However, the host core can still modify system-wide configurations that rely on shared resources (e.g., the memory control hub and the IO control hub). Next, we prove that such configurations can only perturb the isolated environment, for example, by rebooting the system, without threatening its confidentiality or integrity.

According to the Advanced Configuration and Power Interface (ACPI) specification [10], processor cores keep their state as long as the processor runs in the S0 or S1 power modes. Thus, the isolated core will keep running the isolated environment as long as either S0 or S1 state is maintained. On the other hand, all other ACPI states will cause the processor core to lose its state and resume execution from the non-volatile memory (i.e., the BIOS). Since the BIOS belongs to the trusted computing base, changing the ACPI state will be detected by SICE. In all cases, the ACPI implementation of the target platform needs to be carefully reviewed to securely implement SICE.

As presented in Section 4.2, attacks that use IPI between processor cores are thwarted using APIC setting and the GIF of the processor core that runs the isolated environment.

Availability: SICE does not provide protection against attacks that target the availability of the isolated environment. Example attacks include perturbing the isolated environment through system reboots or denying it network access. However, this type of attacks is easily detectable by SICE and its remote users (e.g., from the lack of response of the isolated environment), and can be easily thwarted by removing the malicious code from the legacy host.

The TCB of SICE: SICE aims to minimize and enhance the protection of the TCB of the isolated environment so as to maximize its security.

The TCB of the isolated environment consists of the hardware, the BIOS, and the SMM. Using the SMM gives SICE two main advantages over microhypervisor-based isolation. First, SICE TCB enjoys the hardware protection provided for the SMRAM. Second, the SMM's attack surface is much smaller than that of microhypervisors. Thus, The TCB of the isolated environment is better protected and less complicated than that of microhypervisors. Section 1.2 compares the TCB of SICE with that of microhypervisors in detail.

Trusting the BIOS is required to start the trust chain based on the static root of trust management (SRTM) technique. However, recent incidents [17] show that the SRTM can be compromised. Nevertheless, SICE can adopt the Dynamic root of trust management (DRTM) to start the trust chain, which consequently eliminates the BIOS from the TCB. SICE can use the DRTM to invoke a trusted code that securely initializes the SMRAM, given that it will not be locked by the BIOS. A similar technique was used by Trustvisor [19] to initialize its TCB.

SICE requires the legacy host to trust the security manager(s). However, this does not make the security guarantee provided to the legacy host weaker than microhypervisor-based approaches such as NOVA, which assume trust in a thin hypervisor with typical duties. Indeed, SICE and the microhypervisor-based approaches provide similar security guarantees for the legacy host, while SICE additionally also provides a stronger protection for the isolated environments, as discussed in Section 1.3.

4.4.2 *SMM Security*

Despite being an integral part of the TCB of SICE, the SMM was not designed to provide security services. It was originally designed to create a shadow environment to run hardware management tasks. In the following, we discuss some of the important issues that need to be considered before deploying SICE, or any other system that relies on the SMM, in a production environment.

Legacy SMM Tasks: Intuitively, the TCB of SICE includes any code that exists in the SMRAM. To keep the TCB minimal, legacy system management tasks need to be eliminated from the SMM. This does not mean that SICE needs to completely eliminate them from the system. Instead, SICE can simply forward specific SMIs to (possibly relocated) legacy SMM code after taking the required memory protection measures.

Previous SMM Attacks: As mentioned in Section 3, there have been attempts to subvert the SMRAM using cache poisoning [6,33]. Although these attacks can be easily prevented by using proper hardware configuration, it shows that hardware vendors should undergo a rigorous review of the SMM security to avoid any further security problems.

It is worth mentioning that the implication of SMM attacks is beyond the systems that rely on the SMM for security. It is shown in [7] that a subverted SMM can be used to host a stealthy rootkit that undermines the security of any system. In general, trusting the underlying hardware, including the SMM, is imposed on all software systems.

Hardware Vendors Cooperation: As mentioned in Section 4, implementing SICE requires hardware vendors to allow adding its code to the SMI handler before locking the SMRAM. In fact, we advocate that SICE should be entirely implemented by hardware vendors. In this case, the system BIOS will be responsible for initializing SICE upon booting the system. This guarantees that SICE will always be compatible with the specification of the underlying hardware.

5. SICE PROTOTYPE: AN ISOLATED VM

In this section, we present a SICE prototype implemented on an IBM LS22 blade server, which is equipped with two 2.7GHz AMD Opteron 10h family quad-core processors. We use an Ubuntu 9.10 Linux as the OS. In our prototype, we replace the original SMI handler with SICE's SMI handler. Implementing the core functions of SICE's SMI handler,

IBM - Ex. 1002-Part 10 of 10, Page 3347
IPR2025-00379 (IBM v. Croga)

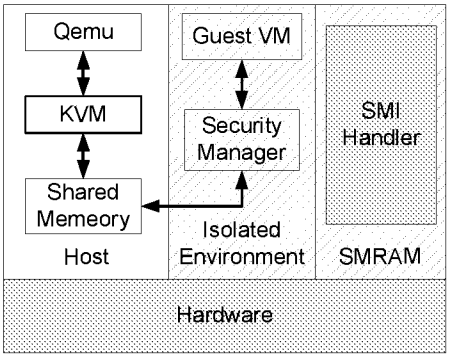


Figure 5: SICE Prototype

which are to prepare and enter the isolated environment, requires around 300 LOC (excluding cryptographic libraries)

As mentioned in Section 1.3, the isolated environment can support running any software, ranging from a single program to a complete VM. To run a single program inside SICE, it needs to be instrumented so that it directly runs on top of the security manager. The required instrumentation is similar to that required by previous research on running isolated security sensitive code (e.g., Flicker [20], Trustvisor [19]).

In our prototype, we use SICE to run a Linux VM, rather than running an instrumented single program. This is to demonstrate the flexibility provided to the isolated environment. Running a whole VM requires the ability to support a diverse set of applications, frequent context switching, and a large range of protected memory.

Figure 5 shows our SICE prototype. The security manager plays the role of the hypervisor in the isolated environment. It uses hardware assisted virtualization to manage and run the VM as the isolated workload. All hardware management functions are delegated to the legacy host, specifically, a modified version of Qemu/KVM [18] running in the legacy host. Qemu is a program that emulates hardware peripherals. KVM is a kernel module that manages other VM operations like scheduling and memory management.

We modified KVM so that it uses a shared memory to send the required VM configurations to the security manager. The same shared memory is used by the security manager to pass the information required to request hardware services from the legacy host. This architecture is mainly chosen to simplify the implementation efforts. In our prototype, the security manager is composed of around 2.1 KLOC, which is comparable to the size of current microhypervisors.

Our implementation provides both network interface and serial port emulation using Qemu. Networking is necessary to allow the VM to communicate with remote users, while the serial port is used as a console for debugging.

Our prototype does not support graphic display emulation, which is not necessary for cloud computing applications. Moreover, we do not support disk drive emulation because our experiments do not need a permanent storage.

Supporting a disk drive emulation is straightforward based on the technique used to implement the network interface emulation. However, the main question is how to secure the disk access from a compromised legacy host. The answer is dependent on the isolated workload rather than the isolation mechanism. For instance, security sensitive applications can either use full disk encryption (similar to Bitlocker [8]), or selectively encrypt secret files only (e.g., a file that contains user passwords). Other applications only need to keep secret data in memory without a permanent storage (e.g., web servers that process online purchases without storing customer credit card numbers).

In the following, we present more details about our prototype implementation.

5.1 Preparing and Initializing the VM

As discussed in Section 4.1.1, the legacy host provides SICE with the initial VM image, composed of the VM kernel and the initial ram disk. The SMI handler copies this image to the SMRAM. KVM also places the required VM configuration parameters (e.g., RAM size) in the shared memory then triggers an SMI to initialize the isolated workload. As mentioned in Section 4.3, the SMI handler measures the initial VM image to attest to its integrity to the remote owner of this VM. The integrity evidence should be extended to include the VM configuration parameters passed by KVM. Some of these parameters (e.g., the VM execution entry point, initial register values) are critical to its integrity.

The security manager prepares the VM page tables and its virtual machine control block (VMCB), based on the provided configuration parameters. The page tables are crafted using AMD's nested page tables (NPT) [1], which adds another layer to the virtual-to-physical memory translation.

5.2 Handling VM Exits

Certain events force the VM to exit its operations (e.g., external interrupts, page faults). These events need to be handled by the hypervisor.

We forward most of the VM exit events to KVM in the legacy host. Nevertheless, the security manager handles some specific VM exits that do not require much computation to avoid unnecessary context switching. Among these VM exit events are control register accesses and requests to execute privileged instructions like `CPUID` or `INVD`.

Other VM exists (e.g., writing to an IO port, accessing an IO memory) are directly forwarded to the legacy host. To preserve the VM confidentiality, the security manager should only send information that is necessary for handling the VM exits (e.g., VM exit reason and error code). This information is sent to the KVM in the legacy host through an established communication channel between the isolated environment and the legacy host.

5.3 Communication with the Legacy Host

Establishing a communication channel with the legacy host is not managed by the SMI handler. Instead, it is directly managed by the security manager and the legacy host. As discussed in Section 4, a shared memory that is outside the SMRAM protection is used to establish the communication between the legacy host and the isolated workload.

Signaling is required between the two environments to send notifications that data is placed into the shared memory. In the time-sharing mode, we use the context switching as the method of signaling. In the multi-core mode, IPIs between the host and the isolated cores are used for signaling, as discussed in Section 4.2.1.

5.4 Using Hardware Peripherals

In our prototype, hardware peripherals in the isolated workload are emulated using Qemu in the legacy host. There are three main methods to control a hardware peripheral:

IBM - Ex. 1002-Part 10 of 10, Page 3348
IPR2025-00379 (IBM v. Croga)

IO ports, IO memory, and Direct Memory Access (DMA). When the VM accesses an IO port or an IO memory, a VM exit occurs and the control is transferred to the host to emulate the hardware access.

DMA requests work differently because DMA is supposed to directly read or write physical memory. In legacy VMs, Qemu is granted full access to the VM physical memory to emulate DMA accesses. However, since Qemu is located in the legacy host, SICE prevents it from directly accessing the memory of the isolated environment. Thus, our prototype modifies Qemu to send a request to the security manager with the address, size, and type (read or write) of the emulated DMA. The security manager in turn copies the required memory between the VM memory and the shared memory. To preserve the confidentiality of the isolated workload, the security manager only allows Qemu to retrieve or modify VM memory areas assigned to DMA operations.

In our prototype, the security manager handles DMA access to simplify our implementation. Nevertheless, this operations should be directly handled by the isolated VM by allowing it to directly access a part of the shared memory, which will reduce the tasks required from the security manager, and consequently reduce its code size.

5.5 Attestation

As mentioned in Section 4.3, the SMI handler should be a part of a trusted boot process. However, the LS22 servers, used for our implementation prototype, are not equipped with a TPM. Due to the lack of a TPM, the attestation process is not included in our prototype. Nevertheless, it is worth mentioning that static attestation using the TPM, signature key generation and signing are all known techniques that have been implemented previously.

It is worth mentioning here that implementing these cryptographic operations may increase the size of the code base of SICE. For instance, a typical SHA1 library is around 120 LOC. Other cryptographic functions (e.g., generating an RSA signature) can be done using the TPM.

6. EXPERIMENTAL EVALUATION

We perform a set of experiments to evaluate the performance of SICE. There are two anticipated sources of performance overhead associated with SICE. The first is the direct overhead resulting from entering/exiting the isolated environment. The second is the indirect overhead that results from the cache and TLB flushing required for SICE operations. On the other hand, running the isolated environment outside the SMM avoids any execution slow-down and is not anticipated to cause any performance overhead, compared to running the same workload without SICE.

In the rest of this section, we present a measurement of the anticipated overhead. First, we measure the execution time needed for a full context switching to and from the isolated environment. Second, we use the SICE prototype (See Section 5) to compare the performance of the isolated guest VM with the same VM running without SICE isolation.

6.1 SICE Execution Time

We measure the execution time needed to perform each of the four major steps of entering and exiting the isolated environment. The measured steps include: (1) triggering the SMI, (2) preparing the isolated environment by the SMI handler, (3) entering the isolated environment, and (4) returning to the legacy host. These measures are obtained from the average of 100 runs.

To precisely measure the end-to-end execution time of each step, we use the `RDTSC` instruction to read the processor's Time Stamp Counter (TSC). We then convert cycles to microseconds based on the TSC speed (2.7GHz in our experimental platform).

Table 1: SICE execution time

Operation	Time (in μs)	Std. Dev.
Triggering an SMI	6.8	0.074
Preparing the isolated env.	20.7	0.155
Entering the isolated env.	9.3	0.233
Exiting the isolated env.	30.1	0.644
Total ($\approx$)	67	

Table 1 shows the experimental results for the time-sharing mode. Triggering an SMI and switching the processor context from the legacy host to the SMI handler needs an average of 6.8 μs. The SMI is invoked by the local APIC of the processor core by sending an IPI to all cores on the system.

The next step, which is to prepare the isolated environment, needs an average of 20.7 μs. The latency of this step is relatively high given that this step only requires changing a few entries in the page tables, modifying the interrupt vector table descriptor, and verifying the values of several registers and MSRs. We anticipate that this relatively high latency is caused by a slower processing speed in the SMM. A similar observation was made in [2].

The next step is entering the isolated environment. In this step, we execute the "return from SMM" instruction (`RSM`) to jump to the security manager. The time needed for this step is 9.3 μs, which is relatively high. We anticipate this latency is due to the change of the processor execution environment, particularly the page tables, which leads to invalidating all cache and TLB entries.

Finally, the time needed to return from the isolated environment to the legacy host is 30.1 μs, which is similar to executing all the previous steps in the reverse order (i.e., from the isolated environment to the legacy host). In total, the end-to-end time required to enter and exit the isolated environment is 67 μs.

In the time-sharing mode, the end-to-end execution time overhead obtained in this experiment occurs on every context switch between the isolated environment and the legacy host. A context switch is required every time the isolated environment needs a service from the legacy host. In contrast, it is a one-time overhead in the multi-core mode.

6.2 Isolated Environment Performance

Our next experiment is to evaluate the overall performance overhead on the isolated workload. We run this experiment using the same system configuration of our prototype system, discussed in Section 5. In our experiments, we assign a 256 MB to the VM. Our VM runs Linux kernel v2.6.28. and a ram disk that is equipped with BusyBox v1.10.2 [4]. (BusyBox is a program that combines common utilities into a single executable.)

To evaluate the performance of the guest VM, we assemble a set of tests and run each test on the same guest VM in three different environments: (1) unmodified Qemu/KVM using

IBM - Ex. 1002-Part 10 of 10, Page 3349
IPR2025-00379 (IBM v. Croga)

one processor core, (2) SICE time-sharing environment using a single processor core, and (3) SICE multi-core environment using two processor cores, used as the host core and the isolated core. In the multi-core test, we do not run any program other than Qemu/KVM on the host core, so that we can get a precise measurement of the performance overhead.

To measure the performance overhead, we use Qemu to emulate a serial port for the guest VM. We then configure the guest VM to use this serial console to accept shell command. Our performance measurement is the execution time needed for the guest VM to complete each of these tests. The execution time is calculated by using the `RDTSC` instruction in the legacy host. The time measurement is taken just before sending the last letter of the shell command (the carriage return) and right after the first response is received (echoing the carriage return).

To calculate the performance overhead of each test, we first measure T_K, the time needed to run the test using unmodified Qemu/KVM. Afterward, we measure T_S, the time needed to run the same experiment using SICE. The performance overhead is then calculated using the equation $(T_S - T_K)/T_K \times 100\%$.

We selected our test cases in three categories: The first is user level programs to test the user level operations of guest VMs running with SICE. The second is a group of tests that test the throughput of the guest VM kernel. The third tests the performance of the emulated network interface.

In the first category, we run three tests. The first measures the latency of copying a 2.1 MB file. Our prototype system does not support hard drive emulation. Thus the file is copied inside the virtual file system that represents the initial ram disk. Our main objective of this test is to estimate the impact of SICE on the guest VM memory operations, because it is basically a copy between two locations in memory rather than an actual disk access. The next two tests in the same category use the programs `gzip` and `gunzip` to compress and decompress the same file. These tests aim to test the impact of SICE on the guest VM's user level computations. The final results of these tests are obtained as an average of 100 runs.

In the second category, we measure the guest VM kernel responses to three main operations: `fork`, `getpid` and `insmod`. `fork` and `getpid` are tested using a custom program that directly calls these system calls 10,000 times and measures the average response time. For the `insmod` test case, we use the `insmod` program to insert a 16.4 KB loadable kernel module in the guest kernel. This test is repeated 10 times, and all test runs show very little variance.

In the last category, we run a single test that uses `wget` to retrieve a 156.6 KB file from an Apache server running in the legacy host. The test is repeated 10 times.

Figure 6 shows the results of our experiments. In the multi-core mode, all tests, except for `wget`, showed a slight performance overhead under 3%. We expect that the slight performance overhead is due to the time required to copy information to and from the shared memory between the legacy host and the security manager.

The `wget` test shows a 17% performance overhead. However, a significant portion of this overhead is due to the *non-optimized* implementation of the Qemu network emulator for SICE. When a packet is ready to be sent through the network, the VM sends an IO command to the network adapter to inform it that a packet is ready. In normal operations,

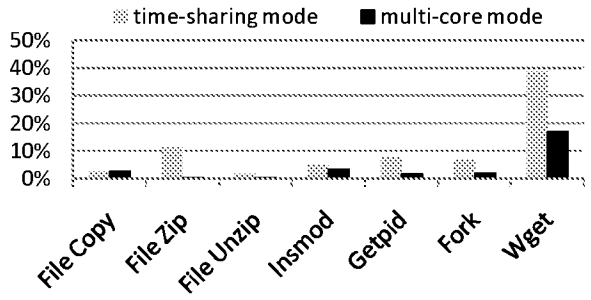


Figure 6: Performance overhead of an isolated VM compared with original Qemu/KVM

Qemu first receives this IO command, and then directly accesses the VM physical memory to emulate the DMA read of the packet. In our prototype, Qemu does not have direct access to the physical memory of the guest VM. Thus, we modified Qemu to send DMA requests to the security manager, which copies the packet from the protected guest VM memory to the shared memory so that Qemu can process it correctly. Similarly, an extra step is necessary when the VM receives a packet.

Hence, our prototype requires an extra communication step between Qemu in the legacy host and the security manager for every network packet. Intuitively, this extra communication step requires an extra context switch from the legacy host to the isolated environment in the time-sharing mode or an extra inter-process communication between the host core and the isolated core in the multi-core mode.

This extra communication step is the cause of the high overhead in the `wget` test. To support this claim, we measured the number of exits to and from the isolated environment during this test. In normal operations, there was an average of 1,249 guest VM exits that required hardware emulation by KVM and/or Qemu. In SICE's time-sharing mode, there was an average of 1,877 context switches between the isolated environment and the legacy host. There was a similar number of messages passed between the host core and the isolated core in the multi-core mode. This 50% increase in the communication time between the two environments explains the high overhead in this test.

To avoid this overhead, the VM device driver can be modified so that it uses the shared memory for the network adapter's DMA operations. Thus, the Qemu emulated driver would be able to directly read the passed packets without sending an extra request to the security manager.

The time-sharing mode showed a higher performance overhead, which is expectable due to the 67 μs needed to switch between the host and the isolated guest VM. The tests that show higher overhead, such as `gzip` and `wget`, are those that require a higher number of switching between the host and the isolated environment.

It is also noticeable that the time-sharing mode showed a higher overhead (around 40%) in the `wget` test. Receiving the target file in the normal VM operations needs 0.532 seconds compared to 0.739 seconds in the time-sharing mode. As mentioned above, this test requires 1,877 context switches from the isolated environment to the legacy host. Given that each context switch requires an average of 67 μs, the context switch overhead in this test was 0.126 seconds. This test clearly shows that the time-sharing mode is not suitable for operations that are IO intensive. It will be rather

IBM - Ex. 1002-Part 10 of 10, Page 3350
IPR2025-00379 (IBM v. Croga)

useful to run programs that require an enhanced isolation and a smaller IO footprint (e.g., a program that processes secret key operations).

7. RELATED WORK

Several researchers (e.g, [3, 5, 9, 22, 34]) attempted to use hypervisors to enable strong isolation between workloads running in a cloud computing environment. Nevertheless, recent attacks [16,32] and vulnerability reports [23,24] show that hypervisors are subject to security exploitation.

There have been several attempts (e.g., [2,30,31]) to verify the runtime integrity of hypervisors. These techniques still require cloud computing users to trust the host environment, which has a relatively large TCB and continuously interacts with mutually distrusted workloads.

As discussed in Section 1, the recent attempts that aim at eliminating the hypervisor from the TCB can be divided into microhypervisor-based and hardware-based approaches.

Hardware-based approaches: The introduction discussed how systems that rely on the late launch capability (e.g., Flicker [20], BIND [25]) fall short from our objectives. Moreover, Intel proposed a hardware service called Processor Measured Application Protection Service (P-MAPS) [21] to offer runtime memory isolation. However, there is insufficient detail about its ability to provide multi-core isolation. IBM also introduced interesting security features in its Cell Broadband Engine (Cell BE) architecture [26], which provides multi-core isolation. However, these features are unique to this specific architecture. SICE achieves the same level of isolation on the x86 architecture, which is used in the vast majority of general purpose computing platforms.

Microhypervisor-based approaches: We discussed two notable microhypervisors, NOVA [27] and Trustvisor [19], as well as seL4 [15] in the introduction. In particular, seL4 is a promising approach that uses formal verification to introduce a vulnerability-free microkernel, though it cannot be extended to commodity hypervisors due to its restrictions (e.g., it should run with interrupts mostly disabled). It would be interesting to study how we can apply formal verification to guarantee the integrity of SICE's security manager.

8. CONCLUSION

In this paper, we presented SICE, a research prototype that aims to explore the capability of current hardware platforms in providing more secure isolated environments. We demonstrated that current hardware architecture already provides abstractions such as SMM that can support strong isolation. We showed that building strong isolation mechanisms on top of the SMM abstraction requires a very small TCB of about 300 LOC, which tremendously reduces the TCB size compared with previous techniques. Nevertheless, practical deployment of SICE would require CPU and hardware platform vendors to undergo detailed security reviews of the BIOS software and the SMM implementation.

SICE provides a set of unique capabilities to the isolated environment, including (1) fast context switch to and from the legacy host, (2) protected concurrent execution with the legacy host, and (3) attestation to its integrity. Moreover, its ability to run isolated workloads concurrently with a legacy host provides a cost-effective solution to security sensitive workloads without using dedicated hardware platforms.

Our future research will focus on extending SICE to work on Intel platforms. In addition, we will explore how to apply formal verification methods on SICE's security manager.

9. REFERENCES

[1] Advanced Micro Devices. Amd64 architecture programmer's manual: Volume 2: System programming, September 2007.

[2] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. HyperSentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the 17th ACM conference on Computer and communications security (CCS '10)*, pages 38–49, 2010.

[3] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan. TVDc: managing security in the trusted virtual datacenter. *SIGOPS Oper. Syst. Rev.*, 42(1):40–47, 2008.

[4] BusyBox. `http://www.busybox.net/`.

[5] X. Chen, T. Garfinkel, E. C. Lewis, P Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D.R.K Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems (ASPLOS'13)*, pages 2–13, 2008.

[6] L. Duflot. Getting into the SMRAM: SMM reloaded. In *Proceedings of the 10th CanSecWest conference*, 2009.

[7] S. Embleton, S. Sparks, and C. Zou. SMM rootkits: a new breed of OS independent malware. In *Proceedings of the 4th international conference on Security and privacy in communication networks*, pages 1–12, August 2008.

[8] Niels Ferguson. Aes-cbc + elephant diffuser: A disk encryption algorithm for windows vista. Microsoft Corporation Technical Report, 2006.

[9] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM symposium on Operating systems principles (SOSP '03)*, pages 193–206, 2003.

[10] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification. revision 3.0b, October 2006.

[11] Intel Corporation. Trusted eXecution Technology preliminary architecture specification and enabling considerations. Document number 31516803, 2006.

[12] Intel Corporation. Intel virtualization technology for directed I/O. Document number 31516803, 2007.

[13] Intel Corporation. Software developer's manual vol. 3: System programming guide, June 2009.

[14] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. Nohype: virtualized cloud infrastructure without the virtualization. In *Proceedings of the 37th annual international symposium on Computer architecture (ISCA '10)*, pages 350–361, 2010.

[15] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. sel4: formal verification of an OS kernel.

IBM - Ex. 1002-Part 10 of 10, Page 3351
IPR2025-00379 (IBM v. Croga)

In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09)*, pages 207–220, 2009.

[16] K. Kortchinsky. Hacking 3D (and breaking out of VMWare). In *Black Hat conference*, 2009.

[17] Dartmouth PKI Lab. TPM reset attack. `http://www.cs.dartmouth.edu/~pkilab/sparks/`. Accessed in August 2011.

[18] Kernel Based Virtual Machine. `http://www.linux-kvm.org/`.

[19] J. McCune, Y. Li, N. Qu, A. Datta, V. Gligor, and A. Perrig. Efficient TCB reduction and attestation. In *the 31st IEEE Symposium on Security and Privacy*, May 2010.

[20] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki. Flicker: an execution infrastructure for TCB minimization. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, March/April 2008.

[21] R. Sahita, U. Warrier, and P. Dewan. Dynamic software application protection. Intel Corporation, April 2009.

[22] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. van Doorn. Building a MAC-based security architecture for the xen opensource hypervisor. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, pages 276–285, 2005.

[23] Secunia. Vulnerability report: Vmware esx server 3.x. `http://secunia.com/advisories/product/10757/`. Accessed in August 2011.

[24] Secunia. Xen multiple vulnerability report. `http://secunia.com/advisories/44502/`. Accessed in August 2011.

[25] E. Shi, A. Perrig, and L. van Doorn. BIND: A fine-grained attestation service for secure distributed systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

[26] K. Shimizu, H. P. Hofstee, and J. S. Liberty. Cell broadband engine processor vault security architecture. *IBM J. Res. Dev.*, pages 521–528, September 2007.

[27] U. Steinberg and B. Kauer. NOVA: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European conference on Computer systems (EuroSys'10)*, pages 209–222. ACM, 2010.

[28] Trusted Computing Group. `https://www.trustedcomputinggroup.org/`.

[29] Trusted Computing Group. TPM specifications version 1.2. `https://www.trustedcomputinggroup.org/downloads/specifications/tpm/tpm`, July 2005.

[30] J. Wang, A. Stavrou, and A. K. Ghosh. HyperCheck: A hardware-assisted integrity monitor. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID'10)*, September 2010.

[31] Z. Wang and X. Jiang. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 31st IEEE Symposium on Security and Privacy*, May 2010.

[32] R. Wojtczuk and J. Rutkowska. Xen 0wning trilogy. In *Black Hat conference*, 2008.

[33] R. Wojtczuk and J. Rutkowska. Attacking SMM memory via Intel CPU cache poisoning. Invisible Things Lab, 2009.

[34] J. Yang and K. G. Shin. Using hypervisor to provide data secrecy for user applications on a per-page basis. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE'08)*, pages 71–80, 2008.

APPENDIX

A. PORTABILITY TO INTEL PLATFORMS

In this section, we discuss the portability of SICE to Intel platforms. Although SMM is supported by both AMD and Intel, SICE uses some hardware features that are only provided by AMD. In the following, we discuss these features and propose alternative techniques for Intel platforms. Implementing SICE on Intel platforms using these techniques needs further research, particularly to achieve both the security and performance required from this system.

SMRAM Size and Location Modification: After the SMRAM is locked, Intel platforms do not allow runtime change of its size and location. This restriction has a direct impact on our techniques, which rely on changing the SMRAM size at run time to enable fast context switching to the isolated environment. However, some other Intel features can be used for SICE. For instance, new Intel hardware is equipped with special instructions to do a high throughput cryptographic operations using the Advanced Encryption Standard (AES). In the time-sharing mode, this feature can be used to encrypt and authenticate the memory of the isolated workload while the legacy host is running, and decrypt and verify the same memory in the isolated environment. Thus, the SMRAM can only be used as a permanent storage of encryption keys. Further research is needed to evaluate the feasibility and performance of this approach.

Memory Double-view: Intel platforms define the `TSeg` SMRAM in its memory controller hub, which is shared among multiple cores. Unless the implementation is modified to move the SMRAM definition to a core specific location, our memory double-view technique will not be able to isolate individual Intel processor cores.

Nevertheless, current Intel implementation can apply this technique on processor nodes as they do not share the same memory control hub, allowing a more coarse grained isolation of the resources on physical platforms.

Multi-core Protection: Intel does not provide a capability that can disable all system interrupts. As mentioned in Section 4.2, we rely on this capability to prevent the host core from using the startup IPI to modify the execution environment of the isolated core. However, the startup IPI can only set the instruction pointer to an address in the lowest 1MB of the physical memory. This address range can be easily modified to point to a non-writable memory chip.

IBM - Ex. 1002-Part 10 of 10, Page 3352
IPR2025-00379 (IBM v. Croga)

"Georgia Computes!": Improving the Computing Education Pipeline

Amy Bruckman, Maureen Biggers, Barbara Ericson, Tom McKlin, Jill Dimond, Betsy DiSalvo, Mike Hewner, Lijun Ni, and Sarita Yardi
College of Computing & CEISMC
Georgia Institute of Technology
Atlanta, GA 30332

{asb, ericson, jpdimond, bdisalvo, hewner, lijun, yardi}@cc.gatech.edu,
biggersm@indiana.edu, tom.mcklin@gatech.edu

ABSTRACT
Computing education suffers from low enrollment and a lack of diversity. Both of these problems require changes across the entire computing education pipeline. The "Georgia Computes!" alliance, funded by the National Science Foundation's *Broadening Participation in Computing* program, seeks to improve the computing education pipeline in Georgia. "Georgia Computes!" is having a measurable effect at each stage of the pipeline, but has not yet shown an impact across the whole pipeline.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education–*computer science education.*

General Terms
Experimentation and Human Factors.

Keywords
Computing education pipeline, summer camps, introductory courses

1. CHANGING A PIPELINE
The challenges in computing education today are well-documented [1-4]. "Georgia Computes!" highlights two concerns of interest: Low enrollment and a lack of diversity. Enrollment in computer science programs is at levels not seen since the 1970s. The students that do enroll are overwhelmingly male, and either Caucasian or Asian.

"Georgia Computes!" sees these as related problems. By improving diversity, Georgia could draw more students, including females and under-represented ethnic groups, into computer science. Both problems can be addressed, in part, by improving computing education across the pipeline. We know computing education problems are really pipeline issues—they cannot be addressed at a single point.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.
Copyright 2009 ACM 978-1-60558-183-5/09/03...$5.00.

Many pre-teens have already made up their minds that computer science is boring and characterized as nerdy and uninteresting [4]. Declining interest in high school computer science means fewer students are being drawn into undergraduate computer science. The high failure rates in undergraduate computer science make it more difficult to retain the students that are drawn into computer science classrooms. Changing only one aspect of the pipeline doesn't correct the problem, e.g., if we got pre-teens excited about innovations in computer science, they would only be discouraged in high school and undergraduate classes if those classes didn't reflect similar innovations.

The National Science Foundation *Broadening Participation in Computing* program funded the alliance, "Georgia Computes!" in October 2006. "Georgia Computes!" is an attempt to change an entire state's computing education pipeline. The alliance is directed by Mark Guzdial and is based at Georgia Institute of Technology (*Georgia Tech)*, as the lead technological institute in the University System of Georgia. The effort is an alliance between different units at Georgia Tech, the Georgia Department of Education, the University System of Georgia, the YWCA, and the Girl Scouts of Greater Atlanta, Incorporated. OurThe strategy is to increase interest in computing at the pre-teen level, improve quality of computing education at the high school level, draw students into the undergraduate level, and make apparent to students the opportunities for graduate study in computing. The number of participants is one measure of the growth of the alliance and its impact (see Table 1).

Why Georgia? The challenges and opportunities are enormous in Georgia, with high minority enrollment and low K-12 academic achievement. The University System of Georgia's systemwide enrollment is 22.7% African American. Only 57.6% of Georgia high school students graduate high school on-time, and the numbers are lower for specific under-represented groups: 47.5% for African American students and 42.4% for Hispanic students [5,6]. Georgia has one of the lowest rates of high school students going on to higher education. Benchmarks such as the SAT show Georgia's academic achievement level is among the worst in the nation, 47th out of the 50 states for 2008. Georgia's Black and Hispanic students, in particular, score well below the already low average on the SAT as well as on other standard measures of academic achievement, such as Advanced Placement test scores, college matriculation rates, and the need for college remediation. Georgia had one of the lowest levels of students taking the Computer Science AP exam, which isn't surprising given that CS

IBM - Ex. 1002-Part 10 of 10, Page 3353
IPR2025-00379 (IBM v. Croga)

AP was available in less than 10% of the public high schools in the state when we started. If "Georgia Computes!" makes a difference in Georgia, with everything else working against it, it will be a significant model for others to adopt and adapt.

Table 1: Participation in Georgia Computes! activities

	Pipeline	Years	Num
YWCA Workshops	K-12	2006-2007	30
		2007-2008	40
Girl Scouts Computing Workshops	K-12	2005-2006	190
		2006-2007	372
		2007-2008	1595
HCI Camps	K-12	2007	10
		2008	15
CS AP Teachers	Teachers K-12	2003-2004	44
		2007-2008	87
Faculty Workshops	Faculty	2007	26
		2008	14
Summer Workshops (262 unique teachers have taken one or more workshops with us)	Teachers	2004	40
		2005	72
		2006	79
		2007	82
		2008	92
Summer Camps at Colleges and Universities	Institutions	2006	1
		2007	4
		2008	8

2. STARTING EARLY

Previous work shows that student attitudes about computing are set early on [4]. During middle school, students make up their minds about whether computer science is worth exploring, or not. It is important for any pipeline effort to start at least that early.

Our aim in reaching out to students at these ages is to provide them with a broad definition of computing. Students tell stories using Carnegie Mellon's Alice software [7,8], explore robot programming with LEGO robot kits, create digital arts and crafts using PicoCrickets, and create animations and games using MIT's Scratch [9]. In our summer camps, students start programming in languages like Python.

"Georgia Computes!" offers summer computing camps to high school (started 2004) and middle school students (started 2006) in the Atlanta area. These camps have been highly successful—so successful, in fact, that they are completely full, all summer long, with multiple sessions of each. The "Georgia Computes!" alliance is now helping other University System schools to offer camps. This is discussed further in Section 4.

Additional work with the YWCA's after-school Teen Girls in Technology program in Atlanta reaches middle school girls. The students served by the YWCA program are predominantly African-American. The YWCA program is staffed with undergraduate and graduate students from Georgia Tech who serve as mentors and teachers to the students. We hire student mentors who are female and/or who are members of underrepresented groups, to serve as role models. We sponsored an all-female, majority African American YWCA team to compete in the LEGO FIRST competition (Figure 1).

The Girl Scouts of Greater Atlanta, Inc. has a membership of 40,000 girls in grades K-12 comprising rural, suburban, and urban Georgia. In Saturday computing workshop and summer camp sessions students explicitly explore computing concepts through tools like Alice, PicoCrickets, LEGO robots, and Scratch.

The Girl Scout workshops have been growing extremely fast. Work started with the Girl Scouts during 2005-2006. That year about 190 girls had some introduction to computing with LEGO robots. In 2006-2007 327 girls had some introduction to computing with LEGO robots or Alice. In 2007-2008 1595 girls had some introduction to computing using LEGO robots, Scratch, PicoCrickets, or Alice. The Saturday workshops grew from 20 girls at a time to 65 girls at a time. Additionally, three short workshops were offered for majority-Hispanic Girl Scout troops at a local elementary school. Thanks to the Anita Borg Systers Pass-it-On Grant we were able to provide transportation for Hispanic Girl Scouts to Georgia Tech for a 4 hour workshop.

The results of these activities have been overwhelmingly positive. Tom McKlin is the external evaluator on the team. The Saturday computing workshops are only four hours long; however, despite the short period of time, McKlin found statistically significant improvements in students' attitudes about computing in response to survey questions like "Computer jobs are boring," "Girls can do computing," and "Programming is hard." Of 13 events where we have both pre-surveys and post-surveys, 7 of the workshops had a statistically significant improvement (using Mann-Whitney U-Test, with $p < 0.10$) in participants' attitudes about computing.

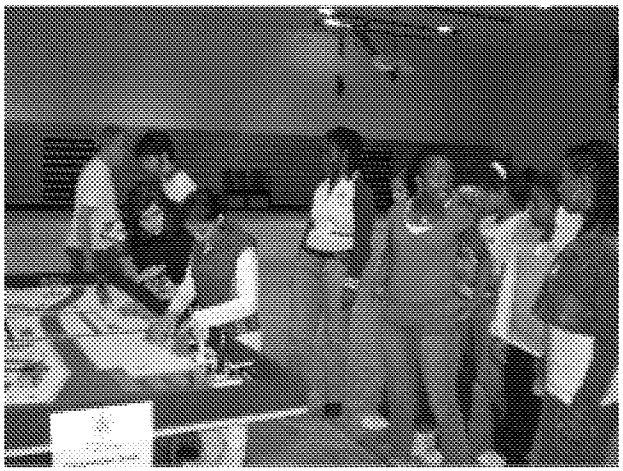


Figure 1: An All-Female, majority African-American LEGO FIRST team, sponsored by "Georgia Computes!" and a first for the Atlanta YWCA TGI-Tech Program.

3. HIGH SCHOOL

The "Georgia Computes!" alliance includes Georgia Tech's *Institute for Computing Education* (ICE@GT). ICE@GT is a collaboration between Georgia's Department of Education and Georgia Tech. The state wanted to grow the number of computer science teachers in the state, with a particular emphasis on

IBM - Ex. 1002-Part 10 of 10, Page 3354
IPR2025-00379 (IBM v. Croga)

developing Computer Science (CS) Advanced Placement (AP) teachers. The state's strategy was to teach existing high school teachers how to teach computer science.

ICE@GT developed a multi-year plan for developing high school teachers, many without previous computer science experience, into CS AP teachers [10]. During the first summer, teachers enroll in a one week summer workshop where they learn to teach an introductory computing course using LEGO robots, Scratch and/or Alice. Teachers should teach that course for at least one year and then take a beginning programming in Java workshop. Teacher training used the *Media Computation* approach [11] which has worked successfully with liberal arts and management (Business) majors [12]—the most common majors of existing Georgia high school computing teachers [13,14]. The teachers are expected to teach beginning programming for a year or more. During years three and four, the teachers take summer workshops in teaching intermediate programming and then CS AP. The combination of four years of experience teaching computing, with in-service workshops, plus summer workshops, can successfully help teachers to become CS AP teachers.

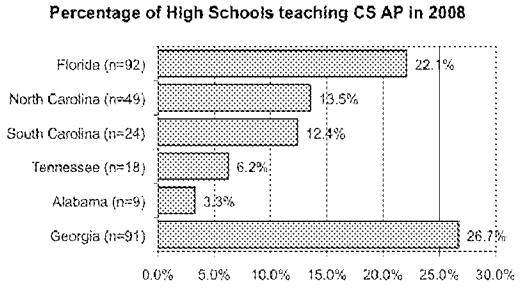


Figure 2: Percentage of High Schools Teaching CS AP in Georgia and Neighboring States

The goal for "Georgia Computes!" was to increase the number of CS AP teachers by 50% over the three years of our NSF BPC grant. By year two of the project, the number of Georgia schools authorized by the college board to teach AP CS had increased by 107%, (from 44 to 91). Georgia has a higher percentage of high schools teaching CS AP than any state in the Southeast (Figure 2). While we can't claim that "Georgia Computes!" and ICE@GT alone accounted for that increase, the broad reach of the increase to counties far away from Atlanta, where most CS AP initiatives have been based, suggests that "Georgia Computes!" has had a significant role in this change (Figure 3).

In addition to supporting high school teachers with training, teachers are provided direct support in their classrooms with student supporters. Marion Usselman, from Georgia Tech's Center for Education Integrating Science, Mathematics, and Computing (CEISMC), is the co-PI on a Student and Teacher Enhancement Partnership (STEP) NSF GK-12 program. CEISMC is the educational outreach center at Georgia Tech. The STEP program, partners advanced undergraduate and graduate student fellows (from fields supported by the National Science Foundation) for ten hours per week with metro-Atlanta area high school science, mathematics, and technology (STEM) teams that are led by master teacher-coordinators. The program uses the exceptional scholarly expertise available at Georgia Tech to assist in increasing the mathematics, science, and computing performance of Atlanta-area school students. "Georgia Computes!" funds STEP students to work in CS teachers' classrooms.

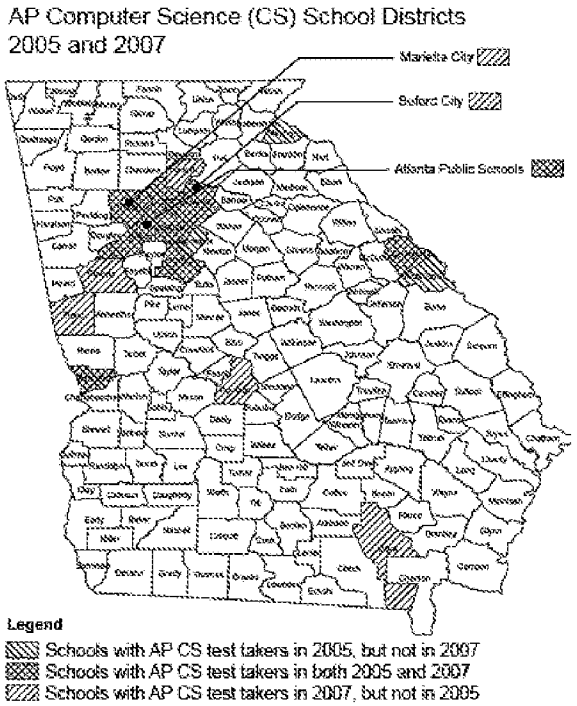


Figure 3: AP CS Test-Takers by County in Georgia. *Note increase beyond the central Atlanta counties.*

4. UNDERGRADUATE

Efforts at the undergraduate level are of two types:

- Workshops to teach Georgia university CS faculty how to offer innovative undergraduate classes that match (in context and motivation) the kinds of activities "Georgia Computes!" using at the pre-teen and high school levels.
- Training and seed-funding to help University faculty in offering middle school and high school student camps, like the ones offered at Georgia Tech.

The University System of Georgia (USG) consists of 35 public colleges and universities, overseen by a single Board of Regents and Chancellor. The University System includes two year, four year, and research universities. Not all 35 institutions offer degree programs in computing. Those institutions that do offer computing programs have representatives on the Academic Advisory Committee on the Computing Disciplines (AACCD). "Georgia Computes!" works directly with the AACCD, e.g., offering workshops at an AACCD meeting, and contacting AACCD representatives for available programs and workshops.

4.1 Faculty Workshops

"Georgia Computes!" has offered a half dozen workshops in a year and half to USG CS faculty. The first workshop was a half-day workshop following an AACCD meeting in February 2007. Since then, workshops have been offered in two day and three day formats in Atlanta (on both weekends, and during summers), and a four day workshop at the Georgia Southern University, to reach faculty in the southern part of the state.

IBM - Ex. 1002-Part 10 of 10, Page 3355
IPR2025-00379 (IBM v. Croga)

The focus of the workshops is to provide faculty with resources and ideas to teach introductory computer science using the contextualized approaches that we are using successfully at the earlier stages in the pipeline. Workshops demonstrate teaching introductory computing with media, robotics, and engineering, using a variety of languages (Python, Java, and MATLAB). These approaches have been shown to improve retention at Georgia Tech and at other schools [14]. Teachers are provided with materials (e.g., example texts, homework assignments, lecture slides) including infrastructure support (e.g., materials to evaluate their own classes, websites and mailing lists for additional support and collaboration with other teachers). The goal of the workshops is to support teachers in adapting these materials for their schools and contexts.

Three-fourths of the state's universities offering some kind of computing program have now had at least one faculty member take one of our workshops. In a study of USG faculty who attended workshops in summer 2007, half of all faculty who attended reported adopting some of the materials in classes [15].

4.2 Growing Summer Camps

As mentioned, the middle school and high school summer camps at Georgia Tech are completely full. While Georgia Tech could look for ways to expand the camp, a broader and more effective impact on the state's computing education pipeline is being developed by helping other USG institutions to offer their own middle and high school student summer camps. Camps at colleges and universities across the state that draw from their own regional school districts will have an added advantage in recruiting those local students into their degree programs.

Faculty are invited to attend workshops on "How to Run a Summer Camp." Each fall a one-day workshop is offered on the logistics of running a camp, e.g., how to advertise, the kinds of permission and consent forms required, and how to put together a sustainable business model. In the spring, a two day workshop is offered on curricula that has been used at Georgia Tech summer camps.

"Georgia Computes!" offers seed grants for up to $5,000 to up to four USG institutions a summer. Institutions are asked to create an application that describes the camp in detail, including the financial plan and how the seed grant will be used. Institutions are encourage to develop a plan where the camp fees sustain the camp, and the seed grant can be used as initial funding as a source for purchasing materials, like robot kits or PicoCricket kits. Institutions are encouraged to use the kits as a lending library to local middle and high schools during the academic year —thus creating another connection between the institutions and the local schools.

"Georgia Computes!" has funded seven new summer camp programs around the state. These range in type from a focus on robotics to a broad focus on mathematics and computing. Funded camps are a part of the evaluation effort, and analysis of their surveys is still on-going.

5. GRADUATE

Georgia Tech's College of Computing is the leader in graduate computing education for the University System of Georgia and is a national leader in creating graduate computing education that has a more diverse population. Georgia Tech offers two non-traditional-CS graduate degrees that emphasize the relevance of computing in everyday settings. The MS in Human-Computers in Interactions involves students in classes in computing, psychology, new media, and industrial design, and has had strong female participation (25% to 52%). The Ph.D. in Human-Centered Computing (HCC) emphasizes the study of computing for humans through the lens of psychology, sociology, learning sciences, and cognitive science, and has always been female-majority.

These graduate students are active in various "Georgia Computes!" programs, serving as role models from diverse backgrounds. Furthermore, these students' work demonstrates a broader definition of computing, including HCI and HCC. We hypothesize that a broader definition of computing may appeal to a larger and more diverse group of students.

6. CHALLENGES AND OPPORTUNITIES

While each of the individual efforts seems to be having impact, we have not yet shown effects throughout the pipeline. Part of the problem is simply the short time of the effort. In the two years of "Georgia Computes!" one can hardly expect pre-teens, whose attitudes may have been changed about computing, to have even completed high school, let alone entered undergraduate or graduate study in computing. Nonetheless, there are changes that we might expect which we would like to track—and yet are finding roadblocks to gathering this data.

One of these challenges is capturing data about enrollment in computing programs across the state of Georgia. Our efforts at the high school level should be influencing undergraduate computing enrollment, and efforts to innovate in computing curriculum should be having an effect on retention rates around the state. How do we gather data from 30+ individual institutions, with different degree programs and different course numberings (and meanings)? The original hope was to gather data from the USG's own "Data Warehouse" which gathers detailed data from every USG institution on its enrollment. However, the "Data Warehouse" does not map all data to a central format—rather, it stores 35 separate databases in 35 separate formats. We are now engaged in the tedious process of learning the mappings between course meanings at all relevant institutions and how to gather data from each individually.

While the challenges are large, so are the opportunities. The interactions in "Georgia Computes!" have created opportunities for new interventions and new research. Here are two brief examples of these kinds of opportunities.

6.1 Research at a Girl Scout Summer Camp

Each of the last two summers, we have had the opportunity to place a Georgia Tech student at a Girl Scout Summer Camp to teach computing. Girls could take robotics or animation as a summer camp activity, along with horseback riding and swimming. In this way, we give girls a chance to interact with technology outside of a classroom context, in a leisurely setting.

During Summer 2008, one of the HCC Ph.D. students, Jill Dimond, used the opportunity to trial a new approach to teaching computing. She taught the girls to design and build chat tools for the One Laptop Per Child XO laptop to be used in the developing world. The response was terrific—she had to hold two sessions daily, rather than the one she planned, because of the number of girls interested. This kind of activity furthered Dimond's research and offered the girls a new model for what graduate students in computing might do.

IBM - Ex. 1002-Part 10 of 10, Page 3356
IPR2025-00379 (IBM v. Croga)

6.2 Beyond the Girls at the YWCA

Betsy DiSalvo is an HCC Ph.D. student who is focusing her attention on African-American boys who were not being reached through our YWCA program. In partnership with Morehouse College, DiSalvo is starting a new effort to build on the interest of many African-American boys in sports-related console video games. She is trying to engage the boys in becoming video game *testers*, and she has signed up several video game companies to offer unreleased versions of their games for the boys to work on. By engaging the boys in testing games, she sets up the situation where they must necessarily dig into the underlying technology and try to understand how the games they love work. Work testing video games is coupled with contextualized computing education in an after-school program that she intends to transform into an after-school jobs program in year two or three. Low-income teens typically must work to earn money once they turn 16 years old. A jobs program could bring educational benefit through their work time.

7. CONCLUSION

The challenge of impacting an entire state's computing education pipeline is enormous. "Georgia Computes!" is doing it the best way we know how, piece-by-piece, by:

- Broadening the definition of computing at the level where students are most likely to lose interest, at the pre-teen/middle school level.
- Improving the training of high school teachers and providing them with support and interesting curricula.
- Innovating in undergraduate computing education across the state.
- Helping colleges and universities across the state to recruit from their local communities.
- Offering computing graduate programs that might draw a more diverse population of students.

The particular challenges today are in assessment. While evaluation results at the piece-by-piece level are strong, demonstrating an impact on the entire *pipeline*, that interventions are crossing the boundaries between levels, is the next big challenge.

At the same time, the opportunities that arise from this alliance are tremendous. We offer just two examples of how the connections that this alliance is creating allows for new kinds of interventions and new kinds of activities that probably would not happen without the alliance. We continue to develop these opportunities and look for ways to further the alliance's goal of improving computing education across the entire state.

8. ACKNOWLEDGMENTS

Our thanks to the NSF Broadening Participation in Computing Program for support (award CNS-0634629). All opinions reflected in this paper are those of the authors and not necessarily those of NSF.

9. REFERENCES

[1] B. Ericson, M. Guzdial, and M. Biggers, "Improving secondary CS education: progress and problems," ACM SIGCSE Bulletin, vol. 39, no. 1, 2007, pp. 298-301.

[2] J. Vargo, "CRA Taulbee Trends: Ph.D. Programs and Ethnicity," Computer Research Association, 2007.

[3] J. Vegso, "Drop in CS bachelor's degree production," Computing Research News, vol. 18, no. 2, 2006.

[4] S. Yardi, and A. Bruckman, "What is computing?: bridging the gap between teenagers' perceptions and graduate students' experiences," Proceedings of the 3rd International Workshop on Computing Education Research, 2007, pp. 39-50.

[5] Donsky, P. (2005). Georgia's failing grade: Some minorities have less than half a chance to earn a diploma on time. *The Atlanta Journal-Constitution*, pages A1,A6. May 18, 2005.

[6] Llewellyn, D. and Usselman, M. (2001). Ethnic Differences in Female Applicants to Engineering and Information Technology Majors. Presented at the 2001 AERA Conference, Seattle, April 2001.

[7] Cooper, S., W. Dann, et al. (2000). Alice: a 3-D tool for introductory programming concepts. *The Journal of computing in small colleges*. Ramapo College of New Jersey, Mahwah, New Jersey, United States, Consortium for Computing Sciences in Colleges.

[8] Pierce, J. S., K. Christiansen, et al. (1998). Alice: easy to learn interactive 3D graphics. CHI 98 conference summary on Human factors in computing systems. Los Angeles, California, United States, ACM Press.

[9] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick, "Scratch: a sneak preview [education]," Creating, Connecting and Collaborating through Computing, 2004. Proceedings. Second International Conference on, 2004, pp. 104-109.

[10] B. Ericson, M. Guzdial, and M. Biggers, "A model for improving secondary CS education," Proceedings of the 36th SIGCSE technical symposium on Computer science education, 2005, pp. 332-336.

[11] M. Guzdial, and B. Ericson, Introduction to computing & programming in Java: a multimedia approach, Upper Saddle River, NJ: Pearson Prentice Hall, 2007.

[12] Forte, A. and M. Guzdial. "Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences from Introductory Courses." *IEEE Transactions on Education,* 48 (2).

[13] Rich, L., H. Perry, et al. (2004). A CS1 Course Designed to Address Interests of Women. *Proceedings of the 35th SIGCSE technical symposium on Computer science education.*. ACM Press: NY.

[14] Tew, A. E., C. Fowler, et al. (2005). Tracking an innovation in introductory CS education from a research university to a two-year college. *Proceedings of the 36th SIGCSE technical symposium on Computer science education.* St. Louis, Missouri, USA, ACM Press.

[15] Ni, Lijun. 2009. "What Makes CS Teachers Change? Factors Influencing CS Teachers' Adoption of Curriculum Innovations." To appear in the Proceedings of SIGCSE '09. Chattanooga, TN, March, 2009.

IBM - Ex. 1002-Part 10 of 10, Page 3357
IPR2025-00379 (IBM v. Croga)

A secure architecture design based on application isolation, code minimization and randomization

Aditi Gupta
Purdue University
West Lafayette, IN
Email: aditi@purdue.edu

Michael S. Kirkpatrick
James Madison University
Harrisonbourg, VA
Email: kirkpams@jmu.edu

Elisa Bertino
Purdue University
West Lafayette, IN
Email: bertino@purdue.edu

***Abstract*—With fast evolving attacks, using software patches for fixing software bugs is not enough as there are often considerable delays in their application to vulnerable systems and the attackers may find other vulnerabilities to exploit. A secure architecture design that provides robust protection against malware must be guided by strong security design principles. In this work, we propose a system design based on the security principles that aim at achieving isolation, diversification and reducing attack surface. Our design leverages multi-core architecture to enforce physical isolation between application processes so that a malicious or infected application is unable to affect other parts of the system. We use randomization techniques to increase the entropy of the system and thwart various attacks such as code-reuse attacks. Further, we significantly reduce the software attack surface by executing each application on its own customized operating system image that is minimized to only contain the code required by a given application.**

I. Introduction

Recent years have seen a rise in sophisticated malware that employs various techniques, such as return oriented programming (ROP), to bypass system defenses. A typical response to such malware attacks is by patching the software vulnerability that was exploited to launch the malware. This involves deploying patches to vulnerable systems which involves considerable delay. As a result, the attack might have already propagated by the time the patch is applied. Further, patching vulnerabilities is not an ideal solution to building secure systems as attackers constantly find new vulnerabilities and craft new exploits. A secure architecture design that is resilient to evolving malware must be based on strong security principles that minimize the risk and limit damage if some part of the system is compromised. Further, these solutions should not be limited to software layer security but should also enforce protection at hardware layer.

In this paper, we propose a design for secure architectures that is guided by three security design principles. The first is *application isolation* which requires that applications be isolated from each other, thereby preventing a malicious application from compromising other applications. Often, isolation is implemented using software isolation techniques such as virtualization. It has been shown that software isolation is not sufficient since a malicious application may exploit vulnerabilities in the virtualization software to attack other parts of the system [1], [2], [3]. Also, software isolation techniques have been shown to be ineffective against side channel attacks that exploit shared channels such as memory usage statistics for accurate adversarial inference of other processes' behavior [4]. Thus, in addition to software isolation, secure systems must incorporate physical isolation by isolating the hardware resources as well.

The second principle of our design is *code minimization* which reduces the software attack surface available to an attacker. Widely used operating systems (OS) such as Windows and Linux are based on monolithic kernels, that is, the entire OS code executes with superuser privilege. This provides a huge computing base that can be potentially exploited by the attacker to perform privilege escalation attacks. Also, a typical process may use only a small fraction of services provided by a kernel. This leaves a significant chunk of privileged code base that is unused code. Our principle of code minimization aims at minimizing the kernel code so that it only contains the libraries and services required by the application. Code minimization can be enforced at a much finer granularity by also minimizing the libraries so that they only contain the functionality needed by an application. For example, the generic C library, *libc*, is quite large but only a fraction of it is utilized by an application.

The third security principle incorporated in our design is *diversification* that aims at increasing the entropy of a system, in effect increasing the attacker effort required to compromise the system [5], [6], [7]. In current systems, due to their homogeneous nature, the same attack instance is likely to work against several machines as these machines use similar configurations. Software diversification makes systems more robust towards such replicated attacks. For example, ASLR [8] randomizes the base address of libraries for different executions, thus rendering attacks such as return-to-libc [9] ineffective. Our design incorporates diversification using a runtime randomization technique for application code that thwarts attacks that rely on prior knowledge of program layout.

Our proposed design leverages multi-core architectures to achieve strong process isolation by executing each process on a separate set of dedicated processor cores. Each process executes on its customized OS image which is minimized to only contain the code (drivers, services etc.) required by the process. Each application provides a manifest file specifying its dependencies, that is, the list of libraries, services and drivers that this application needs to execute successfully. This manifest file serves as the starting point for dependency analysis that determines the external code required by a given application. For each application, we build a custom OS image consisting of a micro-kernel and the application along with its dependencies. This custom OS image will not have any extraneous service not required by the application.

978-1-4799-0895-0/13/$31.00 ©2013 IEEE

IBM - Ex. 1002-Part 10 of 10, Page 3358
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:33:44 UTC from IEEE Xplore. Restrictions apply.

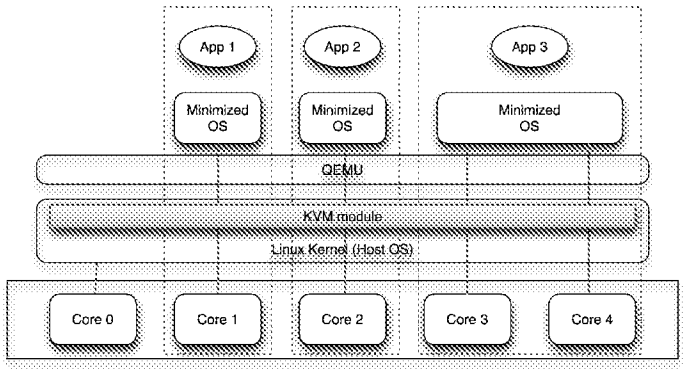


Fig. 1. Overview of system architecture

This paper is organized as follows. In section II, we present the high level design of the proposed architecture. In section III, we present an analysis of *libc* library usage by popular applications. We discuss related work in section IV. We then discuss research issues in section V and conclude in section VI.

II. Secure Architecture Design

We propose a secure architecture design that leverages multicore architectures to achieve strong isolation guarantees between applications. Our basic architecture is inspired from the NoHype architecture [10] which uses multicore architecture to isolate virtual machines (VM) in a cloud setting. Our goal is to isolate not just guest VMs, but also the applications that run on it. We incorporate the basic NoHype architecture into our design and extend it according to our requirements. Figure 1 shows an overview of our system design.

In this architecture, each application is isolated by executing on its own dedicated VM that only executes this application and no other application. The isolation guarantees are further strengthened by enforcing isolation at the hardware layer. This is achieved by executing the application's corresponding VM on a dedicated set of processor cores that are not shared with other applications. At any given time, the system maintains a pool of unassigned processor cores. When a new application starts, it is assigned to a set of cores from this pool using the concept of *processor core affinity*. Memory isolation between VMs is enforced by capitalizing on the hardware paging mechanism in modern processors. Also, like NoHype, I/O devices are dedicated to each VM by virtualizing the I/O devices. Communications between applications executing on different cores is allowed via a restricted interface that is mediated according to certain pre-defined policies.

The guest OS that hosts an application is also minimized according to this application's requirement so as to reduce the software attack surface. That is, only the OS functionality required by an application is retained and parts of the OS that are not required by the application are removed. The OS functionality that is considered for elimination consists of unused kernel services, libraries, drivers and other utilities. This reduces the software attack surface that is available to an attacker to craft a potential exploit. Further, each application is randomized at runtime to defend against attacks that rely on knowledge of memory layout of the target process. Randomizing executable code for each application for its every execution increases the robustness of the system. More on this is discussed later in section II-D.

Our design assumes a large number of processor cores which is not unreasonable given the recent advances in multicore hardware. Some commercial processors currently available in market already boast of more than 100 cores. The maximum number of cores required by a system is determined by the number of applications running at any given time and the number of cores required by each of them. For systems with limited numbers of cores, a variation of our proposed scheme can be used by which the applications are grouped according to a pre-defined criteria and the applications belonging to the same group are allowed to share processor cores. For example, applications written by the same developer can belong to the same group. This allows our scheme to function even with limited number of processor cores.

A. Communication

One important aspect of application isolation is to mediate the communication between two applications. Forbidding all communications between two applications is not always practical since they may need to communicate in order to request and provide services, exchange data etc. For example, in our design, services such as file server may run isolated on a VM and will need to communicate with other applications that require these services. Another example would be an email client that needs to invoke other applications based on email content, such as a PDF viewer or a web browser. While such interactions enhance functionality, they also represent an attack vector that can be exploited by malicious or infected applications. Thus, it is crucial that these interactions be monitored according to a set of policies.

The NoHype architecture has been developed for the cloud settings in which guest OS do not need to communicate with each other as they belong to different customers. This allows one to disengage the hypervisor from guest OS at runtime and eliminate the hypervisor attack surface. However, this is not true in our setting in which each guest OS is dedicated to a single application and applications typically need to communicate. To ensure proper isolation, the communication between the various applications needs to be mediated, by only allowing the authorized communications and preventing unauthorized interactions. Communication mediation can be enforced by the hypervisor by intercepting the communication and enforcing appropriate policies. This means that, unlike NoHype, the hypervisor cannot be disengaged from the guest OS because we need it to mediate the communication between applications.

We propose using RPC-like mechanism to facilitate communication between applications running on different guest VMs. The hypervisor can serve as a middleware for RPC calls and the policy enforcement point that allows or blocks communication. The hypervisor must provide a RPC interface that uniquely identifies the application and the core it is running on and is able to route the RPC calls correctly. As an example, a text editor application running on one core may need to read a file, but the file server is running on another

IBM - Ex. 1002-Part 10 of 10, Page 3359
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:33:44 UTC from IEEE Xplore. Restrictions apply.

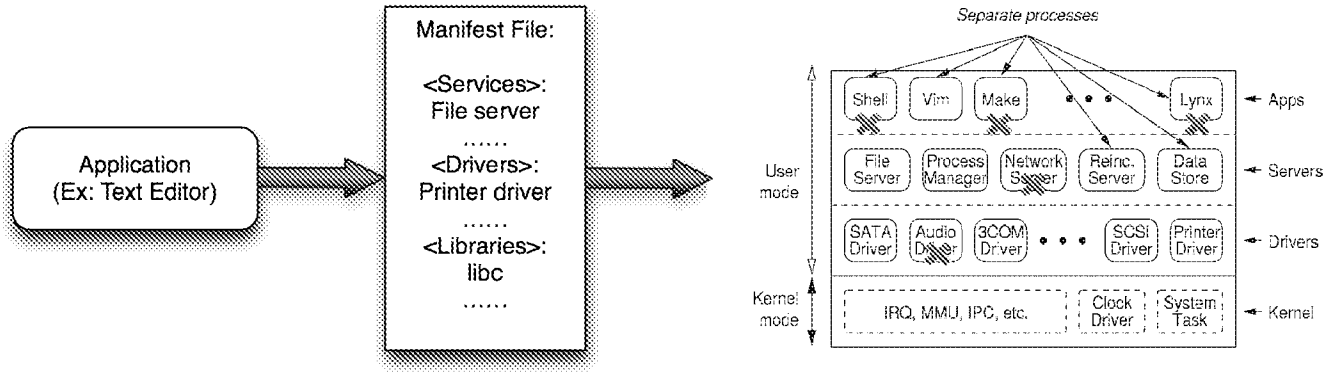


Fig. 2. Example: Minimizing MINIX 3 Operating system for a text editor

core. The call to read the file gets converted into a RPC like call that is transmitted via the hypervisor to the core that is running file server and the response from the file server will follow the same route back. The hypervisor will also check whether the text editor is allowed to ask for file system services and whether it is authorized to read the requested file.

B. Operating System Minimization

Traditional monolithic kernels violate the principle of least privilege by making several services and libraries available to an application even though the application does not require them. If the application is compromised, an attacker can misuse these system services for arbitrary code execution or privilege escalation attacks. To address this, the OS code must be minimized such that each guest OS only makes available the services required by the application executing on it.

To reduce the attack surface offered by the OS, one of the following approaches can be adopted. In the first approach, one can start with a monolithic kernel and remove the unwanted services, drivers and libraries from it. This would require removing a large chunk of code from the OS. The second approach involves starting from a microkernel and only adding the desired functionality that is required by an application. Instead of starting with a monolithic kernel and removing extra code, we adopted the microkernel approach which has only the basic functionality implemented in the kernel space.

We used MINIX 3 [11], a microkernel based OS, for our prototype implementation. Figure 3 shows the design of MINIX 3 where various services (such as file server, network server) and drivers are implemented in the user space. Given an application's requirements, all the services and drivers not required by the application are removed from the OS and the required dependencies of the application are added. This can be viewed as creating a minimal standalone customized OS image for each application.

Figure 2 shows an example use case of OS minimization according to application requirements. In this example, the application under consideration is a simple text editor that requires only few services provided by the MINIX 3 OS. For example, it does not need audio driver and network server and these should be removed from the OS to reduce attack surface. Also, MINIX 3 comes with some pre-installed applications

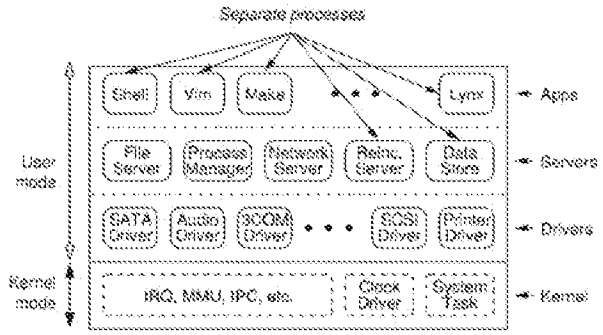


Fig. 3. Architecture of MINIX 3 operating system (Photo credit - MINIX 3 document [11])

such as Make, Lynx etc which should be removed if not required by the application (in this case, text editor).

To generate a minimized custom OS image for an application, we need to obtain the list of drivers, services and libraries required by the application. These should be specified by the application developer in a manifest file that is included in the application package. Since these services may have their own dependencies, a transitive closure of all the dependencies required by the application must be constructed. Correctly specifying all the application dependencies, especially at kernel level, may require significant effort from an application developer. Various techniques can be used to simplify this task for the developer by automatically inferring some or all of these dependencies from either the source code or the executable file. For instance, by analyzing the source code, we can see the libraries included and the system calls made by the application and make some inference about system services used by an application. Process tracing is another approach that can be used to automatically infer application dependencies. This can be done by tracing the process execution and logging the system calls and API calls made by this application. In this approach, it is crucial to execute the application with all possible inputs to build a complete dependency profile for the application. Techniques like those used for software testing can be used to ensure that a complete coverage of inputs is

IBM - Ex. 1002-Part 10 of 10, Page 3360
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:33:44 UTC from IEEE Xplore. Restrictions apply.

	LOC	Symbols
Total	276,970	2171
Min	131,653	360
Average	145,498	450
Std Dev	13,350	75

Fig. 4. Statistics concerning lines of assembly code and symbols usage in *libc*

achieved.

C. Library minimization

In addition to eliminating the libraries that are not needed by an application, a more fine grained minimization can be performed on the libraries themselves. In current systems, even if an application requires access to a single function in shared library, the entire library is mapped into the process memory image. This is a problem of over-provisioning of access. That is, the coarse granularity of shared library loading is inadequate for enforcing the *principle of least privilege* within the library. Consequently, an attacker exploiting an application vulnerability has more access to the library than is necessary. By minimizing the libraries so that they only include the required functionality, the principle of least privilege for code can be enforced more accurately.

To determine the library functionality that may be called by an application, we perform a control flow analysis to identify the reachable code in a library. This involves performing an any-path evaluation, beginning with the set of dynamically mapped symbols in the program's ELF symbol table. These can be retrieved using `objdump` utility. This any-path evaluation maintains a set of symbols to evaluate, where the set is initialized with the application's required symbols. Each symbol in this set is processed by traversing the corresponding binary instructions in a disassembled version of the shared library. Whenever any variation of `ret`, `jmp` or `call` instruction is encountered, the target symbol is added to the working set. This is executed until the least fixed point of symbol references is reached, indicating that all library instructions that could possibly be reached from the symbols in the application's symbol table have been traversed. The minimization technique only retains the code blocks of shared library corresponding to this set of symbols required by the application and removes rest of the code from the library.

D. Code Randomization

To increase robustness against various code reuse attacks, we have incorporated code randomization in the proposed secure architecture design. Randomizing an application's code breaks the underlying assumption for various code-reuse attacks that assume fixed address layout, thus rendering them ineffective. An application's code can be randomized at different stages, such as during compile time (so that every copy of distributed software has different address layout) or at runtime. Randomizing the application code at runtime ensures that each execution of this application uses a different randomized layout. Further, the randomization can be performed at various levels of granularity such as function level, basic block level or instruction level. The choice of granularity can be determined based on the need for security by different applications. That is, a critical application with higher security requirements can be randomized using a finer granularity of randomization such as basic block level randomization.

Our approach randomizes applications at runtime as the frequency of randomization is high and this significantly increases the brute force effort required from an attacker. Randomization is performed at the granularity of function block level as follows. First, the application binary is disassembled and the disassembly is parsed to extract the function blocks and relevant information (such as function address, length etc.). Next, a random permutation of function blocks is generated and a new randomized binary is generated according to this new order of functions. Finally, various jump and call instructions are patched to point to correct destination addresses. A detailed description of this randomization technique can be found in our previous work [12].

III. *libc* USAGE ANALYSIS

To better understand typical *libc* usage in a variety of settings, we performed an empirical analysis of the usage of the *libc* library by 51 popular applications, which are listed in the Appendix. We selected applications from a broad range of categories (*e.g.,* network applications, language interpreters, virtual machine monitors, media applications). Our analysis is based solely on the binary executable, so we considered both open- and closed-source software. All software packages were downloaded for Ubuntu 10.04 ("Lucid Lynx") for version 2.6.32 of the Linux kernel.

To analyze an application's *libc* usage, we performed a control flow analysis as explained in section II-C. In our static analysis, we discovered that there is a significant minimum threshold of *libc* code that must be retained for all applications. Specifically, all processes created from ELF executables jump to the `__libc_start_main` symbol, which indicates the start of the program. From this single symbol, every application reaches 360 of the 2171 function symbols in *libc*.

Figure 4 shows the total and minimum *libc* code base, as well as some basic statistics about the applications we surveyed. In Figures 5-6, the minimum values are labeled and marked with dotted line for reference. Figure 5 shows the number of *libc* symbols used by a subset of the applications we surveyed. For each application, the darker region on the left indicates the number of symbols explicitly identified by performing `objdump` on the executable; the lighter region on the right denotes the total number of symbols reached through our any-path analysis. Figure 6 shows the total number of *libc* lines of code that are required for each application. This figure shows that most applications only require about half the code that exists in *libc*. Furthermore, observe that most of this code is the minimum (*i.e.,* the code reachable from `__libc_start_main` entry point). From Figure 4, we observe that the average application requires only 90 additional symbols when compared with the minimum. That is, minimizing the *libc* code according to the application's needed functionality has a drastic effect on the attack surface. On average, the lines of code (*i.e.,* assembly instructions) and number of symbols used is significantly smaller than the full library size. In other words, mapping the full library gives the attacker access to a significant amount of unused code for attacking using attacks such as code-reuse attacks.

IBM - Ex. 1002-Part 10 of 10, Page 3361
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:33:44 UTC from IEEE Xplore. Restrictions apply.

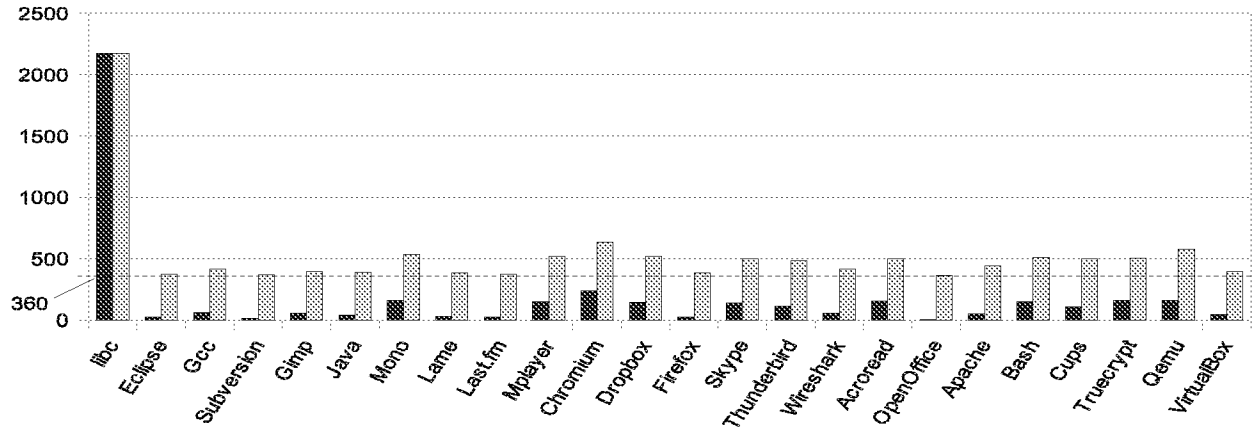


Fig. 5. Number of *libc* symbols used by popular applications

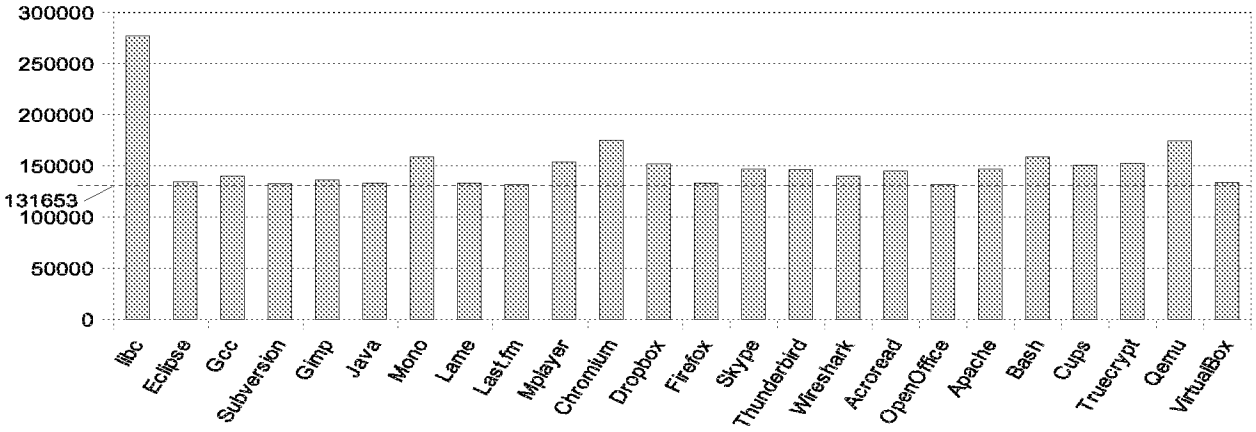


Fig. 6. Number of lines of *libc* assembly code used by popular applications

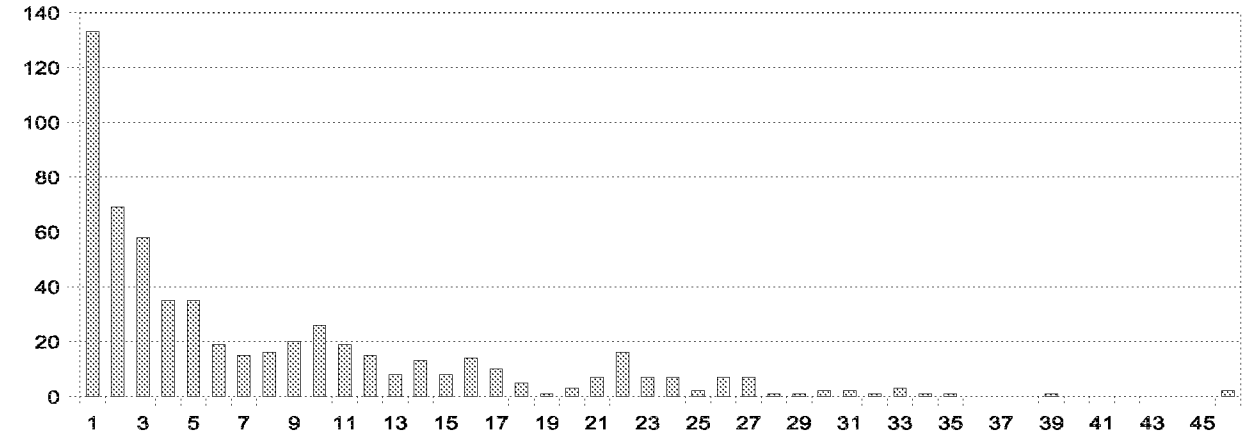


Fig. 7. Histogram of non-minimum symbols used by popular applications

IBM - Ex. 1002-Part 10 of 10, Page 3362
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:33:44 UTC from IEEE Xplore. Restrictions apply.

Finally, Figure 7 shows an interesting result concerning the frequency of non-minimal symbol usage among the 51 applications surveyed. That is, setting aside the 360 symbols derived from `__libc_start_main`, we calculated how many of the applications used each of the remaining symbols. For instance, there were 133 symbols that were used by only one application, while there was only one symbol used by 39 applications. Furthermore, 426 (72.2%) of the symbols in *libc* were used by 10 or fewer applications, while 522 (88.5%) of the symbols were used by 20 or fewer applications. In other words, there is significant variation in the non-minimal symbols that are used by applications. Consequently, the probability of any two applications using exactly the same set of symbols is very small; we observed only a single pair of applications (`gs` and OpenOffice) that used the exact same set of symbols, which happened to be the minimal set.

IV. Related work

Our basic architecture is very similar to the NoHype [10] architecture as both leverage multicore to achieve isolation. However, our architecture has some important differences from NoHype. First, the NoHype architecture has been designed specifically for cloud settings where the isolation is performed at VM level as opposed to application level as in our design. Second, NoHype uses the hypervisor only in the initial phase to allocate resources and in the end to terminate a VM. The hypervisor is disengaged from the VM when it starts the application execution. This is not the case in our design since in our design applications may need to communicate and the constant involvement of hypervisor is required to mediate this communication. Also, unlike our approach, NoHype does not use code minimization for the guest OS running in the VM.

Poly^2 [13], a secure platform for network services, isolates network services on different systems and proposes minimization of the OS to remove unused services. However, this work is limited to network services and uses different machines for isolation. Our approach significantly extends Poly^2 by isolating all applications in addition to just network services. Also, in our architecture, the isolated applications still run on the same machine and isolation is enforced by eliminating sharing of the processor cores and isolating memory.

Codejail [14] is a framework to isolate untrusted libraries so as to prevent a bug in the library from compromising the main program. This is achieved by creating separate execution contexts, that is trusted and untrusted contexts for executing main program and untrusted library code respectively. All accesses to system resources from untrusted context are sandboxed. Codejail approach can be integrated into our proposed design to achieve a higher degree of isolation within an isolated VM.

Dong et al. [15] quantify security benefits and performance costs of performing privilege separation in web browser designs. In web browsers, various browser components can be grouped into isolated partitions that are assigned minimum privileges as needed for runtime operation. However, excessive isolation between browser components may increase communication between components and incur higher performance costs. [15] provides empirical data on this trade-off while choosing various partitioning strategies in web browsers.

Tartler et al. [16] reduce the size of trusted computing base by reducing the kernel code using compile time kernel configuration options. This reduces the attack surface available to the attacker. They showed a decrease of 10% in CVE vulnerabilities by reducing the kernel code using configuration option. This reduction is based on the anticipated workload and is limited to the KCONFIG features. On the other hand, our minimization is specific to the application that will execute on the guest OS.

V. Discussion

We have presented a secure architecture design based on the security principles of isolation, minimization and randomization. This approach presents interesting research challenges to be addressed as future work.

One research challenge is achieving strong hardware isolation by eliminating all shared hardware between isolated applications. Currently, some hardware resources such as network card are shared between various applications. However, to achieve stronger isolation guarantees, it is desirable that this sharing of hardware resources is minimized. This can be achieved by virtualizing all the hardware resources. This is not trivial as not all hardware devices support virtualization currently.

Another research direction is how to achieve security goals similar to our design if the number of processor cores is limited. This will require that some applications share processor cores. To minimize the security risks, it is crucial that applications that share a core have similar trust level. For example, applications from the same developer can share a set of cores as these applications are unlikely to attack each other. One could also group applications based on their security requirement. For example, all finance related applications can be grouped to share a set of cores.

Long running processes also pose interesting research challenges, especially with respect to the diversification goal. This is because for long running processes, randomization is not performed frequently enough. This may lead to an attacker eventually learning some randomization parameters over a long period of time due to small leaks. To prevent this, even long running processes should be randomized frequently. Restarting the process may not be desirable as it leads to service outage and server state may be lost. Randomizing a running process without interrupting it is another research challenge to be addressed.

The current design can be extended to include various other diversification strategies. For example, in addition to supporting function level randomization of application code, a system can support multiple randomization techniques that operate at different granularities. This further increases the entropy of the system.

VI. Conclusion

In this work, we have proposed an initial secure architecture design that is based on the principles of application isolation, code minimization and code randomization. Application isolation is enforced at both software and hardware layers to prevent a malicious application from compromising

IBM - Ex. 1002-Part 10 of 10, Page 3363
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:33:44 UTC from IEEE Xplore. Restrictions apply.

other parts of the system. Code minimization for the operating system code and the library code is performed to reduce the software attack surface that can potentially be exploited by an attacker. We have also performed analysis of *libc* library to show that only a fraction of the entire library is used by any given application. For securing applications from attacks such as ROP attacks, our architecture incorporates application randomization techniques by which the application code is randomized at each execution [12]. As result, security will be achieved by the combination of four key approaches: isolation, mediated communication, minimization, and randomization.

VII. Acknowledgments

The work reported in this paper has been partially supported by a grant by Sypris Electronics and by NSF grant CNS-1111512.

References

[1] Xen Security Advisory CVE-2012-0217, "64-bit PV guest privilege escalation vulnerability," http://lists.xen.org/archives/html/xen-announce/2012-06/msg00001.html.

[2] National Vulnerability Database (NVD), "Vulnerability Summary for CVE-2009-1244," http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1244.

[3] Kostya Kortchinsky, "CLOUDBURST: A VMware Guest to Host Escape Story," http://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-SLIDES.pdf.

[4] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012, pp. 143–157.

[5] S. Forrest, A. Somayaji, and D. Ackley, "Building diverse computer systems," in *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, ser. HOTOS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 67–.

[6] F. B. Cohen, "Operating system protection through program evolution," *Comput. Secur.*, vol. 12, no. 6, pp. 565–584, Oct. 1993.

[7] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-variant systems: a secretless framework for security through diversity," in *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. Berkeley, CA, USA: USENIX Association, 2006.

[8] PaX Team, "PaX," http://pax.grsecurity.net/.

[9] Solar Designer, "Getting around non-executable stack (and fix)," August 1997.

[10] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 401–412.

[11] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Minix 3: a highly reliable, self-repairing operating system," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 3, pp. 80–89, Jul. 2006.

[12] A. Gupta, S. Kerr, M. Kirkpatrick, and E. Bertino, "Marlin: A fine grained randomization approach to defend against rop attacks," in *Network and System Security*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7873, pp. 293–306.

[13] E. Bryant, J. Early, R. Gopalakrishna, G. Roth, E. Spafford, K. Watson, P. William, and S. Yost, "Poly2 paradigm: a secure network service architecture," in *ACSAC '03*, 2003, pp. 342–351.

[14] Y. Wu, S. Sathyanarayan, R. H. C. Yap, and Z. Liang, "Codejail: Application-transparent isolation of libraries with tight program interactions," in *ESORICS*, 2012, pp. 859–876.

[15] X. Dong, H. Hu, P. Saxena, and Z. Liang, "A quantitative evaluation of privilege separation in web browser designs," in *ESORICS*, 2013.

[16] R. Tartler, A. Kurmus, B. Heinloth, V. Rothberg, A. Ruprecht, D. Dorneanu, R. Kapitza, W. Schröder-Preikschat, and D. Lohmann, "Automatic os kernel tcb reduction by leveraging compile-time configurability," in *Proceedings of the Eighth USENIX conference on Hot Topics in System Dependability*, ser. HotDep'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 3–3.

Appendix

Application	Version
Acroread	9.3.2
Amazon MP3 Downloader	1.0.9
Apache	2.2.14
Bash	4.1.5
Bluetooth	4.60
Brasero	2.30.2
Chromium	10.0.648.205
Cups	1.4.3
Dhclient	3.1.3
Dropbox	0.6.7
Eclipse	3.5.2
Empathy	2.30.3
Evolution	2.28.3
Firefox	3.6.16
Gcc	4.4.3
Gimp	2.6.8
Git	1.7.0.4
Gnome-terminal	2.30.2
Grip	3.3.1
Gs	8.71
Gtkpod	0.99.14
Gzip	1.3.12
Java	1.6.0_20
Lame	3.98.2
Last.fm	1.5.4
libc	2.11.01
Make	3.81
Mencoder	4.4.3
Mono	2.4.4
Mplayer	1.0
OpenOffice	3.2.0
OpenSSH	5.3p1
PdfTeX	1.40.10-2.2
Perl	5.10.1
Python	2.6.5
Qemu	0.12.3
Sha1sum	7.4
Skype	2.1.0.81
Smbclient	3.4.7
Subversion	1.6.6
Sudo	1.7.2p1
Tar	1.22
Thunderbird	3.1.8
Totem	2.30.2
Truecrypt	6.3a
Vim	7.2
VirtualBox	4.0.0
Vlc	1.0.6
Transmission	1.93
Wine	1.2.2
Wireshark	1.2.7
Xine	0.99.6

IBM - Ex. 1002-Part 10 of 10, Page 3364
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:33:44 UTC from IEEE Xplore. Restrictions apply.

Design and Implementation of Sandbox Technique for Isolated Applications

Muhammad Shams Ul haq[#], Lejian Liao[#], Ma Lerong[#]

[#]School of Computer Science and Technology, Beijing Institute of technology, Beijing, China

[1]shams_5@yahoo.com

[2]liaolj@bit.edu.cn

[3]malerong_bit@bit.edu.cn

Abstract— **In presence of known and unknown vulnerabilities in code and flow control of programs, virtual machine alike isolation to confine maliciousness of process is an effective strategy to contain the attack effects in isolated environment. But most of proposed isolation techniques does not offer execution sandbox. A process running in isolated environment with unrestricted access, without explicit mechanism for restriction on access for native system resources such as system call table, network and file system, can access unauthorized resources. In this paper, we propose a sandbox technique for applications running in Virtual Machine alike isolation. The proposed solution is a reference monitor that works without tampering with transitioning mechanism of process and does not require changes in program or kernel. We implemented prototype as executable shared library for dune that provides isolation to native Linux process. Reference monitor uses seccomp BPF filters, Linux Secure Module Apparmor and ptrace utility of native kernel to restrict access to system resources. Experimental results show that proposed technique provide security with acceptable overheads.**

Keywords—computer security; apparmor; seccomp filters; Reference monitor; isolation

I. INTRODUCTION

Operating system security has become a great concern due to prevalence of malicious binaries and exploitable vulnerabilities present in innocent binaries. The formal checking of these vulnerabilities seems impossible due to vast code base and different sources of vulnerabilities. In the same way end user expectations for ease of use and user carelessness exacerbate the security situation of system. For example users do not validate the source before downloading programs from internet or show carelessness in timely patching the system. Even if user patch system it is not guaranteed that patch itself is bug free and does not have vulnerabilities.

In this situation, security experts and attackers remain in war to find new vulnerabilities that can help in attacks. Rather being part of war between defenders and attacker for finding unknown vulnerabilities, confining malicious or anomalous behavior of application seems good solution. Virtual machines promise maximum isolation for system and user data on shared hardware. Virtual machines provide strong isolation between operating systems so that an application running in one virtual machine cannot effect other applications in other virtual machine or on host. But separate virtual machine for each application is not a feasible solution in real practice due to overheads incurred and deviance from traditional user experience. Even if applications run in separate VMs, due to poor integration with host and other VMs, it is not feasible to share resources and communicate among applications.

To overcome overheads of VM and exploit advantages provided by virtual machines, many solutions [1, 2, 3] are proposed to provide VM equivalent isolation for untrusted and malicious applications. Dune [4] is one of such solutions that isolate a native Linux process in its own VM by using hardware virtualization extensions. Even though dune provides sandbox by transitioning malicious binary into VMX ring 3 but dune sandbox only single process single threaded applications. Dune sandbox even does not support file system access monitoring and restriction. An isolated and sandboxed process has full access to file system. In the same way dune implemented network firewall to restrict access to network but this network firewall is a standalone application that cannot be integrated with file system or execution sandbox.

In the same way present era operating systems are general purpose systems to execute as many applications as possible. Each application has its own needs to access system resources, operating systems implement functionality to fulfil each application service requests. The most used method to request service from operating system is system call. Operating system provide a complete set of system calls available to application without knowing specific needs of an application. For example all Linux operating systems provide applications with more than 300 system calls to request resources. But an application merely use a very small subset of these system calls. Surprisingly, we note that Linux gzip utility only uses 24 different types of system calls. It means an application isolated in its own virtual machine must be explicitly restricted to invoke only those system calls that are needed to complete its tasks. Invocation of any other system call should result in termination of program.

In order to overcome these problems, we propose a generic and unified mechanism for sandboxing isolated process execution, file system access and network connectivity. The proposed mechanism is generic in sense that it does not tamper with the isolation mechanism so can easily be integrated with any isolation technique. The proposed method neither require modifications in kernel nor application. The proposed solution consists of an executable shared object (shared library) that

978-1-4673-9194-8/16/$31.00 ©2016 IEEE

IBM - Ex. 1002-Part 10 of 10, Page 3365
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:31:24 UTC from IEEE Xplore. Restrictions apply.

works as reference monitor for accesses to native system resources. Reference monitor uses seccomp BPF filters [5] to restrict system call table access and a manifest [6] consisting of Linux Secure Module (LSM) Apparmor [7] policies to restrict file system and network access.

II. Background and Preliminaries:

In this section we cover the basic technologies and concepts used in our proposed method. This effort will help reader not only to understand proposed method and its scope for security provision but also state of the art in operating system security.

Intel Virtualization extensions: Intel extended its x86 ISA with VTx to simplify virtualization mechanism and improve performance of operating systems running in virtual machine. VTx divides the hardware into two distinct mode VMX root and VMX non modes both having 4 traditional privilege rings. In VMX root mode, virtual machine monitor (VMM) or host operating system executes whereas non root mode is dedicated for guest operating systems. CPU transitions in (VM entry) and out of non root mode (VM exit) on specific configurable events. This transition between root and non root mode is called VMX transition. To facilitate VM entry and VM exit, VMM uses VM control structures to store and load the machine state during VMX transition and configure events for VM exit.

VM alike isolation: we differentiate VM alike isolation from sandboxing. The reason is that it is not necessary for former to facilitate execution sandbox and vice versa. We categorize those techniques under this definition that provide another level of abstraction or indirection by virtualizing one or more entities of operating system. For example Solaris zones [8], dune, Graphene [9] feather weight virtualization [10] belongs to this definition. Most of such methods use operating system level virtualization but it is also possible to use hardware to provide VM alike isolation. Graphene is one of solutions that provide applications with most of system calls with in isolated environment. Whereas system calls with external effects are sandboxed through reference monitor. The work present in this paper is inspired by graphene and share some code with it.

Sandboxing: techniques that restrict process access to kernel and system data such as firejail [11], seccomp, Apparmor, Solaris zones [8]. Sandboxing does not constitute strong isolation to contain maliciousness of a process. Instead sandboxing techniques are usually used to restrict process access to kernel structures, file system or network.

Dune: Dune uses hardware virtualization extensions Intel VTx and Extended Page Tables (EPT) to transition a process in VMX non root ring 0 to provide VM alike isolation. The main purpose of dune is to gain efficiency by intercepting system call in ring 0, forwarding to Virtual Machine Monitor (VMM) with VMCALL instruction of VTx technology and issue system call manually on host kernel. VMCALL performs explicit VMX transition into VMM (VM exit), VMM identifies reason for VM exit and perform system call manually on behalf of isolated process. A process running in VMX ring 0 is termed dune process and has complete direct access to system resources. Dune configure VMCS for each thread so it is possible that one thread of process is a dune process whereas other is a normal

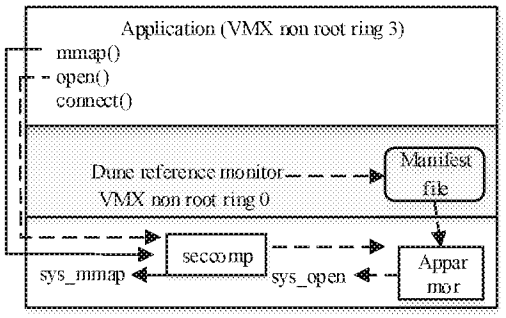


Fig. 1. Example for treatment of system calls invoked by sandboxed application

native host process. Dune implements three use cases sandbox, Wedge [12] and Garbage collector. Dune uses ring 3 to implement application sandbox, with limited functionality, for single process single threaded applications.

Seccomp filters: is a technique to restrict system call table. As discussed earlier, each application invoke a small number of system calls from a large number of system calls available in operating systems. Seccomp provides mechanism to profile system calls made by clean execution of an application and enforce security by sandboxing execution by allowing legal system calls either as whitelist or black list. Seccomp substantially reduce the risk of attack by reducing kernel entry points.

Apparmor: Apparmor is a famous Mandatory Access Control (MAC) technique, now, packaged with many Linux distributions such as Ubuntu. It is also a default mechanism used by Ubuntu to restrict process access to system managed resources. Like seccomp, Apparmor also facilitates user to profile application requirements before enforcing security policy. For example during trial execution, in complain mode, Apparmor logs system access whereas in enforce mode, process can only access allowed resources such as files with granted permissions. Apparmor has many tools to define and enforce policies. Apparmor policy parser Apparmor_parse converts text policies into machine instructions so that kernel can mediate resource access.

Manifest files: according to [6], manifest describes a program dependencies on other system, execution requirement and allowed resources. User usually invoke program execution through manifest file that facilitates the execution environment by getting configurations from system and code. But it is not always necessary to start execution with manifest because a default manifest can be loaded for program as Ubuntu click package does. Manifest based security solutions can range from very complex implementation [6] to somewhat simple implementation like Ubuntu click program for software development. The minimum requirement for manifest based solution is to extract application dependencies and configuration that needs substantial effort.

IBM - Ex. 1002-Part 10 of 10, Page 3366
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:31:24 UTC from IEEE Xplore. Restrictions apply.

III. SYSTEM OVERVIEW

The proposed reference monitor has multiple responsibilities: initialize and install seccomp filters, open and load manifest into kernel as Apparmor policy and bootstrap binary in dune with minimum required execution environment and configuration. User start execution of candidate application with reference monitor instead of directly starting execution or through program manifest. The reference monitor has one general manifest for each program that has been evaluated for this paper.

Reference monitor mediates all system accesses through manifest containing allowable actions defined by user. Whenever application invoke system call it is intercepted by host kernel resident seccomp for checking against manifest file. Seccomp filters consists of whitelist of system calls that can be vectored in host system call table. Any unallowable system call, not present in whitelist, results in termination of the process. But seccomp filters cannot mediate file system accesses so for restricting file system access, reference monitor uses Apparmor profiles. As a process accesses file system resources through system calls such as open, such system calls are first checked by seccomp and traced if system call is allowed and require access to network or file system. In order for tracing, reference monitor uses ptrace facility integrated with seccomp. Reference monitor consult with manifest file to determine the file system access validity against defined security policy. In the same way we extended manifest file with network access rules so that system calls related with network connectivity such as bind and connect are traced and checked against network policy. A process can connect with only those ports and networks that are allowed by user in network access policy.

Example: Fig. 1. shows working of reference monitor integrated with dune process. First dune process invokes mmap system call, it is allowed by seccomp due to local effect of the call. So this call is not directed to manifest file and directly executed. Whereas when process tries to open a file on system with open, seccomp forward this call to reference monitor for validity of accessed resource. Reference monitor decides on the basis of policies whether requested access is a legal access or not.

IV. SYSTEM DESIGN AND DETAILS

Proposed method involves many challenges and design choices such as how to make shared library an ELF executable, how and where (either ring 0 or ring 3) to transition reference monitor in dune, how to load binary with the help of reference monitor and how to forward system calls to reference monitor.

We identify following design goals for proposed sandboxing solution

Non-modification: The proposed method neither require changes in candidate applications nor in kernel.

Non-tampered: proposed method should not tamper with the entry process of application into isolated environment. This goal helps in devising general solution that can be integrated with any isolation technique that needs sandboxing, without or with minimal efforts.

Complete mediation: all accesses, with external effects, to system resources must pass through reference monitor.

We implemented reference monitor as executable shared object (shared library) that not only start loading of application but also sandbox process execution. In order to make shared object a dynamically linked executable, we note that both an ordinary ELF binary and ELF complaint shared library share same layout with some differences. For example, an ELF executable needs entry point and ELF interpreter. We trick to define own entry point by defining symbol entry and manually call native Linux interpreter ld-Linux.so. The function entry call a function named rm_main whose type is set as main with the directive .type. With these tricks we successfully make shared object as ordinary executable.

A. Integration with dune:

We decided to load reference monitor, instead binary, in dune mode (ring 0) by calling entry functions dune_init and dune_enter, provided by libdune, in function exported as main of reference monitor. The design choice of transitioning reference monitor fulfils three objectives. Firstly it does not need to compile and modify application to get transitioned in dune. Otherwise dune needs to call initialization and entry functions and hence requires recompilation of program. Secondly reference monitor remain a native process of host kernel so it is possible to use host kernel provided sandboxing techniques in our case seccomp and Apparmor. Thirdly manual transitioning of malicious binary into ring 3 becomes easy from within dune process without tampering with dune mechanism of transition native process into isolated environment. For these three above reasons proposed security framework as reference monitor can be ported with any isolation technique with little effort without modifications in applications or kernel.

The first task that reference monitor performed is to load ld-linux.so into ring 3. For loading ld-linux into ring 3, we need ELF loader that load ld-linux.so. We implemented ELF loader that read header information of ld-linux and map ld-linux in dune mode. In order to load interpreter in dune, ELF loader iterates over program headers of binary to load all sections with type PT_LOAD in conformance with dune. Now it is responsibility of loaded loader to map ELF executable binary in ring 3. For loading binary into ring 3, we also need to setup

initial execution environment- setting up stack, arguments and environment variables, determining name of application, setting initial stack and getting stack pointer and entry point for execution to get started. After setting up stack, all this execution environment is also loaded from auxiliary vectors and argument list of binary.

One problem with dune way of sandboxing is that Dune runtime sandbox loaded in ring 0 is not sandboxed and can subvert security mechanism from within privileged ring 0. Unlike dune, we address this problem by using two phase seccomp filtering policies: i) one for initial filtering to load binary into ring 3 ii) second for execution of sandboxed application. Former phase contain seccomp filters that mediate system calls made by ELF loader whereas latter phase seccomp policy consists of application specific filtering rules and override the former.

IBM - Ex. 1002-Part 10 of 10, Page 3367
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:31:24 UTC from IEEE Xplore. Restrictions apply.

TABLE I INDIVIDUAL SYSTEM CALL OVERHEADS

	getpid	read	write	Open+close
Linux	372	473	527	2890
dune	2269	577	653	5173
Dune w/ ref. monitor	2745	577	653	27210

TABLE II END TO END PERFORMANCE OVERHEADS FOR LINUX PROGRAMS

	Ls -R	gzip	gcc
Original Dune	4.2	0.89	0.43
Dune w/ ref. monitor	7.35	3.11	0.99

The use of two phase mechanism make sure that maliciousness and vulnerabilities in our reference monitor and boot loader does not affect the execution of application [13].

B. Security enforcement:

After loading untrusted binary in ring 3 and execution environment setup, malicious executable start execution. All system calls with outside effect such as open, connect and bind are forwarded to reference monitor whereas calls with local effects such as mmap are vectored directly to host system call table with the help of dune provided function for pass-through calls. System call mmap or other memory related calls cannot effect system resources or other applications due to isolation provided by EPT. The calls forwarded to seccomp are of two types: i) those that need further tracing such as open for file access or connect for networking ii) those that only require to determine their presence in whitelist and does not need tracing for further processing such as wait or settimer and pass-through calls. Former system calls are traced by using ptrace integrated with seccomp filters and forwarded to manifest file for further mediation. Whereas later system calls are allowed or restricted with seccomp filter rules.

A process can access kernel resources with the help of ioctl opcodes instead of using system calls. Kernel access through ioctl opcode is also determined by seccomp filters due to availability of label jumps embedded in filtering rules through helper functions. We at the moment only allow the ioctl opcodes defined by dune; any other ioctl code that try to access other driver results in termination of process. But flexible rules can be defined in case of requirement.

One major technical challenge with this design is generating Apparmor profiles and network connectivity policy enforcement from manifest file. For the reasons of simplicity and efficiency, we treat these file system access and network access in two different helper functions with different rules file for network. This network policy file contains only connectable network mask and bind-able port number. This sort of connectivity monitoring is simple and straightforward but file system access monitoring through Apparmor by manifest files, we have to enforce Apparmor policy beforehand. For generating Apparmor enforcement policies from manifest file, we tailored the code [12] according to our needs. More specifically there is need of parser to convert text written policies into machine instructions to be enforced by kernel. For example [12] has huge coding for sandboxing IPC due to its esoteric design for library OS and complex treatment of file system access as union file system [14]. We do not need such functionalities; we use their method to parse Apparmor policies, defined as manifest rules, to LSM Apparmor security module in kernel.

V. EVALUATION

We evaluated proposed solution for incurred performance overheads and provided security. For performance, we calculate execution times for individual system calls and that of applications by bootstrapping them with reference monitor in dune. We also performed experiments with lighttpd and manually written programs to test security provided by security policies through seccomp and manifest file.

For calculating individual system call overheads, we invoked system calls manually and calculated their execution time. Table I shows that most expensive system calls are open/close that are 4.2X slower than native execution. The reason for this overhead is that these calls trap with ptrace and checked against manifest file

Table II shows application execution overheads of dune and dune with reference monitor. For gzip we use gzip recursively on folder that contains 603 files. Gzip is one of the expensive utility used with reference monitor due to its frequent use of most expensive system calls open and close. Further profiling shows that during its execution gzip made total 9798 system calls of 24 different types. Among 9798 system calls, gzip invoked open and close in total 1235 times. Whereas ls on same folder invoked open and close 52 times that's why incurred overhead is far less than incurred by gzip.

In order to determine performance penalty on large scale application that exercise more computation, we perform experiments with lighttpd webserver. As table III shows that reference monitor introduced just 33% extra overheads as compared to original dune without reference monitor. This less overheads show that an application with diverse set of system calls especially that invokes more expensive calls with small ratio results in reasonable performance penalty.

In order to determine security added by our proposed method, we try to manually access file by a process that is not allowed in manifest. The access is successfully rejected by reference monitor. In the same way for lighttpd, we define manifest that allow access for one file public.txt whereas disallow for private.txt. User successfully accessed the public file whereas access to private file is thwarted by reference monitor.

In the same way, we invoke system call that is not present in whitelist of ls, gzip and lighttpd, seccomp successfully determine illegal system call and program is terminated. As for quantitative analysis, ls needs only 21 system calls while gzip needs only 24 system calls out of more than three hundred system calls in our system. It means only 7% of total system

IBM - Ex. 1002-Part 10 of 10, Page 3368
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:31:24 UTC from IEEE Xplore. Restrictions apply.

TABLE III PERFORMANCE OVERHEADS FOR LIGHTTPD WITH AND W/O REFERENCE MONITOR

	C=1	C=100	%decrease
Linux	5562	18172	
Original Dune	5494	17231	5.1
Dune w/ ref. monitor	3681	9305	33

calls is needed by an application. This reduction in system call table vectoring substantially reduces the trusted computing base of the system.

VI. CONCLUSION

In this paper, we presented method to sandbox execution of an isolated application. We implemented our prototype with dune isolation virtual machine. The experimental results showed that, despite individual system call performance overheads, end to end performance overheads are in acceptable range for achieved security. We are currently working on reducing these overheads by porting system calls with local effect in VMX non root ring 0.

ACKNOWLEDGMENT

This work is partially supported by National Program on

Key Basic Research Project under Grant No. 2013CB329605

REFERENCES

[1] Y. Yu, F. Guo, S. Nanda, L. C. Lam, and T. Chiueh. A feather-weight virtual machine for windows applications. In Proceedings of the 2nd ACM/USENIX Conference on Virtual Execution Environments (VEE'06), June 2006,

[2] J. R. Douceur, J. Elson, J. Howell, and J. R. Lorch. Leveraging legacy code to deploy desktop applications on the web. In OSDI, 2008.

[3] J. Dike. User Mode Linux. Prentice Hall, 2006.

[4] A. Belay, A. Bittau, A. Mashtizadeh, D. Terei, D. Mazi'eres, and C. Kozyrakis. Dune: safe user-level access to privileged CPU features. In OSDI, pages 335–348, 201

[5] SECure COMPuting Filters (seccomp). http://kernel.ubuntu.com/git?p=ubuntu/ubuntu-recise.git;a=blob;f=Documentation/prctl/seccomp_filter.txt;hb=HEAD

[6] a G. C. Hunt and J. R. Larus. Singularity: Rethinking the software stack. SIGOPS Oper. Syst. Rev., 41(2), 2007.

[7] Apparmor http://wiki.apparmor.net/index.php/Main_Page.

[8] D. Price and A. Tucker. Solaris Zones: Operating system support for consolidating commercial workloads. In LISA, pages 241–254, 2004.

[9] Chia-Che Tsai , Kumar Saurabh Arora , Nehal Bandi , Bhushan Jain , William Jannen , Jitin John , Harry A. Kalodner , Vrushali Kulkarni , Daniela Oliveira , Donald E. Porter, Cooperation and security isolation of library OSes for multi-process applications, Proceedings of the Ninth European Conference on Computer Systems, April 14-16, 2014, Amsterdam, The Netherlands

[10] Y. Yu, F. Guo, S. Nanda, L. C. Lam, and T. Chiueh. A feather-weight virtual machine for windows applications. In Proceedings of the 2nd ACM/USENIX Conference on Virtual Execution Environments (VEE'06), June 2006,

[11] http://sourceforge.net/projects/firejail/files/firejail/

[12] A. Bittau, P. Marchenko, M. Handley, and B. Karp. Wedge: Splitting Applications into Reduced-Privilege Compartments. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation

[13] https://github.com/oscarlab/graphene/wiki

[14] R. Pike, D. Presotto, K. Thompson, and H. Trickey. Plan9 from Bell Labs. In In Proceedings of the Summer 1990 UKUUG Conference, pages 1–9, 1990 NSDI'08, pages 309–322, 2008.Apparmor

IBM - Ex. 1002-Part 10 of 10, Page 3369
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on May 06,2020 at 12:31:24 UTC from IEEE Xplore. Restrictions apply.

Enforcing Least Privilege Memory Views for Multithreaded Applications

Terry Ching-Hsiang Hsu
Purdue University
terryhsu@purdue.edu

Kevin Hoffman
eFolder Inc.
khoffman@efolder.net

Patrick Eugster
Purdue University and TU Darmstadt
peugster@cs.purdue.edu

Mathias Payer
Purdue University
mathias.payer@nebelwelt.net

Abstract

Failing to properly isolate components in the same address space has resulted in a substantial amount of vulnerabilities. Enforcing the least privilege principle for memory accesses can selectively isolate software components to restrict attack surface and prevent unintended cross-component memory corruption. However, the boundaries and interactions between software components are hard to reason about and existing approaches have failed to stop attackers from exploiting vulnerabilities caused by poor isolation.

We present the *secure memory views* (SMV) model: a practical and efficient model for secure and selective memory isolation in monolithic multithreaded applications. SMV is a third generation privilege separation technique that offers explicit access control of memory and allows concurrent threads within the same process to partially share or fully isolate their memory space in a controlled and parallel manner following application requirements. An evaluation of our prototype in the Linux kernel (TCB $<$ 1,800 LOC) shows negligible runtime performance overhead in real-world applications including Cherokee web server ($<$ 0.69%), Apache `httpd` web server ($<$ 0.93%), and Mozilla Firefox web browser ($<$ 1.89%) with at most 12 LOC changes.

1. INTRODUCTION

Ideally, software components are separated logically into small fault compartments, so that a defect in one component cannot compromise the others. This concept of *privilege separation* [35, 36] protects confidentiality and integrity of data (and code) that should only be accessible from small trusted components. However, most applications use a *single address space*, shared among all components and threads. Redesigning all legacy multithreaded applications to use processes for isolation is impractical. Today's software, such as web servers and browsers, enhances its functionality through libraries, modules, and plugins that are developed independently by various *third-parties*. Failing to properly separate privileges in applications and confine software components in terms of their memory spaces leaves a system vulnerable to attacks such as privilege escalation, denial-of-service, buffer overflows, and control-flow hijacking, jeopardizing both the stability and the security of the system.

Many proposals exist for privilege separation in a monolithic application. The first generation privilege separation techniques focus on splitting a process into different single-process compartments. Provos et al. [33] presented an intra-process privilege separation case study by manually partitioning OpenSSH components into privileged master processes and unprivileged slave processes. Privtrans [9] automated the partitioning procedure. Wedge [8] then introduced capabilities to privilege separation and Salus [40] enabled a dynamic security policy. Unfortunately, all these techniques cannot support multithreaded compartments.

Second generation privilege separation techniques like Arbiter [47] aimed to support multithreaded applications by allowing concurrent thread execution. However, Arbiter's implementation for separating memory space and its serialized user-level memory management impose prohibitive runtime overhead (200% – 400% for memory operations). As a result, the thread execution is not fully concurrent since all threads must wait on a global barrier to tag memory pages for capabilities. In addition, the required retrofitting efforts for legacy software are non-trivial, as the case studies showed that at least hundreds of lines of code (LOC) changes are required to separate software components even for applications that have small code base sizes (8K LOC).

We postulate that a third generation privilege separation technique for achieving intra-process isolation in monolithic multithreaded applications such as the Cherokee web server, Apache `httpd`, and the Mozilla Firefox web browser, needs to fulfill the following requirements (which are only partially addressed by existing solutions) for wide adoption:

- *Genericity and flexibility* (GF): Implementing privilege separation in different types of applications requires programmers to employ completely different abstractions and concepts. A general model with a universal interface and isolation concept that supports *both* client- and server-side multithreaded applications is needed (e.g., compartments in the Firefox browser, worker buffers in Cherokee web server, worker pools in Apache `httpd` web server).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS'16, October 24-28, 2016, Vienna, Austria

© 2016 ACM. ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978327

IBM - Ex. 1002-Part 10 of 10, Page 3370
IPR2025-00379 (IBM v. Croga)

	1st gen technique				2nd gen technique	3rd gen technique
Problem tackled	Non-parallel privilege separation				Concurrent execution and dynamic security policy	Concurrent memory operations and high performance
Issue vs solution	**OpenSSH** [33]	**Privtrans** [9]	**Wedge** [8]	**Salus** [40]	**Arbiter** [47]	**SMV** (This paper)
Security principal	Process	Process	Single thread	Single thread	Multiple threads	Multiple threads
Parallel execution	Not handled	Not handled	Not handled	Not handled	Partially handled	Yes
Parallel tagging	Not handled	Not handled	Not handled	Not handled	Not handled	Yes
Security policy	Static	Static	Static	Dynamic	Dynamic	Dynamic
TCB	OS	Compiler, OS	OS	Library, OS	Library, OS	OS (<1800 LOC)
Refactoring efforts	Fully manual	Annotations	Tool assisted	Tool assisted	Fully manual(>100 Δ LOC)	Library assisted(<20 Δ LOC)
Use cases	OpenSSH	OpenSSH etc.	OpenSSH etc.	PolarSSL	FUSE (8K LOC) etc.	Firefox (13M LOC) etc.

Table 1: Issues and solutions for intra-process privilege separation techniques.

- *Ease of use* (EU): Programmers prefer to realize their desired security policy in a model with a high-level API rather than through low-level error-prone memory management tools without intra-process capabilities (e.g., `mmap`, `shmem`.). In particular, porting legacy software to a new model has to be easy despite the complexity of component interweaving and the underlying assumption of shared memory (e.g., Firefox, which contains 13M LOC), and should be possible with minimal code refactoring efforts.
- *No hardware modifications* (NH): Over-privileged multithreaded applications are pervasive. A model that is ready to run on today's commodity hardware (even regardless of the CPU brands/models) is necessary for wide deployment.
- *Low runtime overhead* (LO): Monitoring application memory accesses at high frequency is unrealistic for practical systems. A practical model must be implemented in a way that incurs only negligible runtime overheads. In particular, enhanced security should not sacrifice the parallelism in multithreaded applications. A model has to support selective memory isolation for multiple computing entities (i.e., multiple threads can exercise the same privilege to parallelize a given workload and perform highly parallel memory operations).

To address the above challenges, we propose a third generation privilege separation solution for monolithic applications: *secure memory views* (SMVs) – a model and architecture that efficiently enforces differential security and fault isolation policies in monolithic multithreaded applications at negligible overheads. SMV protects applications from negligent or malicious memory accesses between software components. In short, the intrinsically shared process address space is divided into a dynamic set of *memory protection domains*. A thread container SMV maintains a collection of memory protection domains that define the *memory view* for its associated threads. Access privileges to the memory protection domains are explicitly defined in the SMVs and the associated `SMVthreads` must strictly follow the defined security policies. The SMV model provides a well-defined interface for programmers to exercise the least privilege principle for arbitrary software objects "inside" a multithreaded process. For example, a server's worker thread can be configured to allow access to its thread stack and part of the global server configuration but not to the private key that resides within the same process address space.

With the SMV model, the programmer can enforce different access permissions for different components in a single address space (GF). New software can leverage the full API and can be designed to only share data along a well-defined API, and existing software can be retrofitted (with minimal code changes) by instrumenting calls across component boundaries to change the underlying memory view (EU). Moreover, the privilege enforcement relies on OS kernel level page table manipulation and standard hardware virtual memory protection mechanisms (NH). Therefore, the SMV model does not suffer from the performance overheads (LO) imposed by IPC (vs in-memory communication), user level memory management (vs kernel level), or per-instruction reference monitors (vs hardware trap). The SMV model's programmability and efficient privilege enforcement mechanism allow it to protect *both* client- and server-side multithreaded applications with low overhead.

We implemented a prototype of the SMV model in the Linux kernel. Our evaluation demonstrates (a) its negligible runtime overhead in the presence of high concurrency using multithreaded benchmarks that employ the general producer-consumer pattern, and (b) the immediate benefit of efficient software component isolation by compartmentalizing client connections for the popular Cherokee and Apache `httpd` web servers and the compartments in the Firefox web browser. SMVs incur only around 2% runtime overhead overall with 2 LOC changes for the multithreaded benchmark PARSEC, 0.69% throughput overhead with 2 LOC changes for Cherokee, 0.93% throughput overhead with 2 LOC changes for Apache `httpd`, and 1.89% runtime overhead with only 12 LOC changes for the Firefox web browser. Note that SMV focuses on restricting memory views for individual threads, access permissions for kernel APIs is an orthogonal problem that is well covered by, e.g., AppArmor [4], SELinux [39], or the seccomp framework [38].

In summary, this paper makes the following contributions:

Design of the SMV model which provides threads with fine-grained control over privileges for a shared address space.

Specification of an SMV API for programmers that facilitates porting existing `pthread` applications.

Implementation of the SMV model that consists of a trusted Linux kernel component (implementing enforcement) and the corresponding untrusted user-space library that implement the SMV API, which is publicly available along with our benchmarks and test suite[1].

Evaluation of our prototype implementation showing that SMVs achieve all four desired requirements as a practical and efficient model for enforcing least privilege memory views for multithreaded applications in practice.

[1] https://github.com/terry-hsu/smv

IBM - Ex. 1002-Part 10 of 10, Page 3371
IPR2025-00379 (IBM v. Croga)

2. THREAT MODEL AND OBJECTIVES

Threat model. We assume that the attacker, an unprivileged user without root permissions, can control a thread in a vulnerable multithreaded program, allocate memory, and fork more threads up to resource limits on a trusted kernel with sound hardware. The adversary will try to escalate privileges through the attacker-controlled threads or gain control of another thread, e.g., by reading or writing data of another module or executing code of another module. In this model, the adversary may read or write any data that the controlled thread has access to. The adversary may also attempt to bypass protection domains by exploiting race conditions between threads or by leveraging confused deputy attacks, e.g., through the API exported by other threads. We assume that the OS kernel is not compromised (OS kernel security is an orthogonal topic [24]) and user-space libraries installed by root users are trusted. We assume that the access permissions both of the memory views (enforced through SMV) and for the kernel (enforced through AppArmor, SELinux, or seccomp) are set correctly.

Objectives. The key objective of the SMV model is to efficiently protect memory references of threads to prevent unintentional or malicious accesses to privileged memory areas during the lifetime of a program. Threads may communicate with other threads through mutually shared memory areas set up by the programmer through SMVs. The SMV model restricts the memory boundaries and memory access permissions for each thread. Without SMVs, an untrusted thread (e.g., a compromised worker thread) may access *arbitrary* software objects (e.g., the private key) within its process (e.g., a web server). Existing programs assume a shared memory space for threads and SMVs must therefore validate that all threads follow the memory rules defined by the programmer (cf. Section 3.2). Threads that deviate from these memory reference rules are killed by the system.

The SMV model aims to strictly confine the memory access boundaries for multithreaded programs while *preserving all four desired requirements* for intra-process isolation. We argue in Section 3 that the SMV model along with the memory access enforcement can constrain threads within the programmer-defined memory boundaries.

3. SMV MODEL DESIGN

The SMV model consists of three abstractions: *memory protection domains*, *secure memory views*, and `SMVthreads`. The SMV model uses user-defined security policies to enforce the threads' privileges in accessing the shared memory space. The flexibility and programmability of the model allows a programmer to specify the protection domains using high-level abstractions while enforcing the security policy at the lowest level of the software stack (page tables) with acceptable runtime overhead.

3.1 Memory Protection Domains

We define a *memory protection domain* as a contiguous range of virtual memory. Any memory address can only belong to one memory protection domain. In this way, a large shared memory space such as the heap can be divided into several distinct sets of memory protection domains. For example, a process can create a private memory protection domain that is only accessible by *one* thread, or a *partially* shared memory protection domain such that only threads with explicit privileges can access it. In addition, an in-memory communication domain can be allocated with global access privileges so that all threads can exchange data without relying on expensive IPC. In general, an unprivileged thread cannot tamper with a memory protection domain even if there exists a defect in the code of the thread. We use the term *memory domain* to refer to the memory protection domain in the rest of the paper.

3.2 Secure Memory Views

We define a *secure memory view* (SMV) to be a thread container with a collection of memory domains. The memory blocks covered by a memory view can only be accessed by threads explicitly given permission to run in the corresponding privileged SMV. Therefore, we consider a memory view to be *secure*.

We define three abstract operations for defining the composition of an SMV:

- *Register(SMV,MD)*: registers memory domain *MD* as part of *SMV*'s memory view.
- *Grant(SMV,MD,P)*: grants *SMV* the capability to access memory domain *MD* with access privilege *P*.
- *Revoke(SMV,MD)*: revokes *SMV*'s capabilities to access memory domain *MD*.

We categorize the privileges P of an SMV to access a memory domain into four operations:

- *Read*: An SMV can read from the memory domain.
- *Write*: An SMV can write to the memory domain.
- *Execute*: An SMV can execute in the memory domain.
- *Allocate*: An SMV can allocate/deallocate memory space in the memory domain.

The access privileges to each of the memory domains for an SMV can be different. Two SMVs can reference the same memory domain but the access privileges can differ. The programmer can set up the SMV's privileges to access memory domains in the way needed for the application at hand. For example, multiple threads sharing the same security context can be assigned to the same SMV to parallelize the workload (LO). To minimize an application's attack surface, the programmer can assume the main parent thread to be the master thread of a program. All the permission modifications must be done by the master thread and are immutable by child threads. The SMV model considers any access to a memory domain without proper privileges to be an *SMV invalid* memory access. We implemented the privilege enforcement at the OS kernel level and detail the design in Section 4.5.

3.3 `SMVthread`

An `SMVthread` is a thread that strictly follows the privileges defined by an SMV to access memory domains. `SMVthreads` run in the memory view defined by an SMV and cannot change to other SMVs. While the popular `pthreads` have to trust all `pthreads` running in the same memory space, `SMVthreads` distrust other `SMVthreads` by default. `SMVthreads` – unlike `pthreads` – must *explicitly* share access to the intrinsically shared memory space with other `SMVthreads`. We designed `SMVthreads` to partially share the memory space with

IBM - Ex. 1002-Part 10 of 10, Page 3372
IPR2025-00379 (IBM v. Croga)

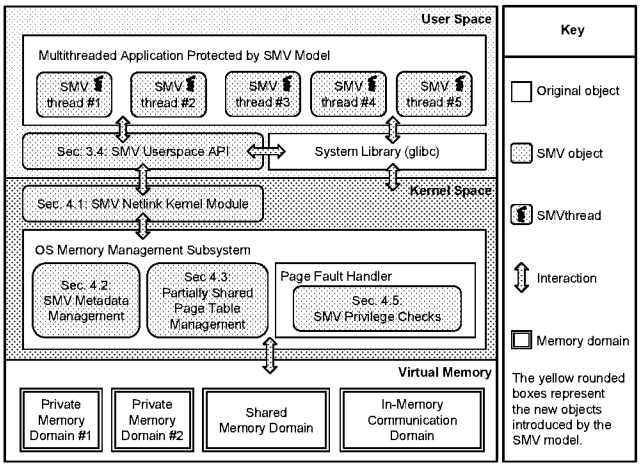


Figure 1: SMV architecture.

other `SMVthreads` according to the policy specified by the programmer through the API. Section 4.3 explains the implementation of the partially shared page tables for `SMVthreads`. `SMVthreads` are `glibc`-compatible, meaning that our `SMVthreads` can directly invoke the library functions in `glibc`. `SMVthreads` can cooperate with `pthreads` through all the synchronization primitives defined by the `pthreads` API. For SMV management, privileged `SMVthreads` have to invoke the SMV API to set up the memory boundaries for least privilege enforcement. `pthreads` can access the whole process address space. Changing such accesses would hamper the correctness of legacy programs that do not require any memory segregation (backward compatibility). While possible, programmers are advised *against* mixing `SMVthreads` and `pthreads` in one process when an application requires isolation as `pthreads` will have unrestricted access to all memory of the process.

3.4 SMV API: User Space Library

We implemented our SMV model as a user-space library that offers an API to support partially shared memory multithreading programming in C and C++. Table 2 summarizes the primary SMV API with descriptions of the main functions. For instance, a programmer can use `memdom_create` to create a memory domain and `memdom_alloc` to allocate memory blocks that are only accessible by `SMVthreads` running in the privileged SMVs. Each memory domain and SMV has a unique ID assigned by the SMV model in the system. `SMVthreads` are integrated with `pthreads` for easier synchronization and every `SMVthread` thus also has an associated `pthread_t` identifier. Note that casting an `SMVthread` to a `pthread` does not bypass the privilege checks. The SMV interface allows programmers to structure the process memory space into distinct memory domains with different privileges for `SMVthreads` and to manage the desired security policy. Furthermore, our library provides options for programmers to automatically override related function calls to significantly reduce the porting efforts. For example, `pthread_create` can be automatically replaced by `smvthread_create`, which internally allocates a private memory domain for the newly created `SMVthread`. Similarly, when an `SMVthread` calls `malloc`, the library allocates memory in the calling thread's private memory domain.

3.5 SMV Architecture

The SMV architecture consists of two parts: a user space programming interface and a kernel space privilege enforcement mechanism. Figure 1 gives an overview. In short, a user space application can call the SMV API to use the SMV model. In the OS kernel, the SMV kernel module is responsible for exchanging the messages between the user space component and the kernel memory management subsystem. We added SMV metadata management to the OS memory management subsystem to record the memory access privileges for the SMVs. We modified the page table management logic to support partially shared page tables and added the SMV privilege checks to the page fault handler that enforces the memory access control.

With the user space interface and the support from the OS kernel, applications can explicitly structure the intrinsically shared process memory space into distinct memory domains with different access privileges without any hardware modifications. Therefore, our approach can be run directly on today's commodity hardware (NH).

3.6 Application Examples

The SMV model allows privilege separation of individual components and data regions in an application. We present one example of the popular design model in general multithreaded applications and two concrete application examples of how the SMV model can protect applications by organizing the process address space with different privileges for threads (GF and EU).

3.6.1 Producer-Consumer Model

First, the SMV model can support the common producer-consumer model with strict memory isolation while maintaining efficient data sharing. Figure 2 illustrates how the SMV model can secure interacting components according to a *generic* producer-consumer model that is employed by all the applications in PARSEC we evaluated. In this example, the `SMVthreads` run in the process address space that contains four SMVs and six memory domains. The SMVs confine the memory access privileges of `SMVthreads` according to the security policy. In this case, the queue domain is the shared memory domain for *all* `SMVthreads` to cooperate with each

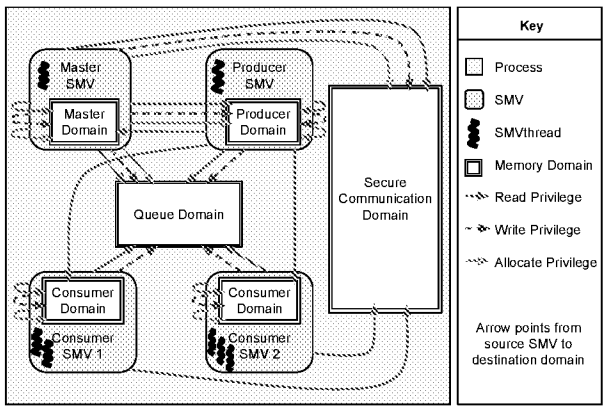


Figure 2: Security-enhanced producer/consumer model with fine-grained memory protection domains.

IBM - Ex. 1002-Part 10 of 10, Page 3373
IPR2025-00379 (IBM v. Croga)

Table 2: List of primary SMV API.

SMV API	Description
`int smv_main_init (bool allow_global)`	Initialize the main process to use the SMV model. If `allow_global` is true, allow child threads to access global memory domains. Otherwise distrust all threads by default.
`int memdom_create (void)`	Creates a new memory domain, initializes the memory region, and returns the kernel-assigned memory domain ID.
`int smv_create (void)`	Creates a new SMV and returns the kernel-assigned `smv_id`.
`pthread_t smvthread_create (int smv_id, (void*)func_ptr, struct smv_data* args)`	Creates an `SMVthread` to run in the SMV specified by `smv_id` and returns a `glibc`-compatible `pthread_t` identifier.
`void* memdom_alloc (int memdom_id, unsigned long size)`	Allocates a memory block of `size` bytes in memory domain `memdom_id`.
`void memdom_free (void* data)`	Deallocates a memory block previously allocated by `memdom_alloc`.
`int memdom_priv_grant (int memdom_id, int smv_id, int privs)`	Grants the privileges `privs` to access memory domain `memdom_id` for SMV `svm_id` and returns new privileges.
`int memdom_priv_revoke (int memdom_id, int smv_id, int privs)`	Revokes the privileges `privs` to access memory domain `memdom_id` from SMV `svm_id` and returns new privileges.
`int memdom_kill (int memdom_id)`	Deletes the memory domain with `memdom_id` from the process.
`int smv_kill (int smv_id)`	Deletes the SMV with `smv_id` from the process.

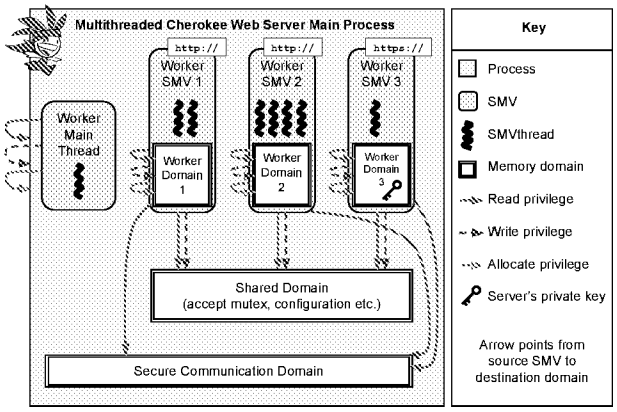


Figure 3: Security-enhanced Cherokee web server.

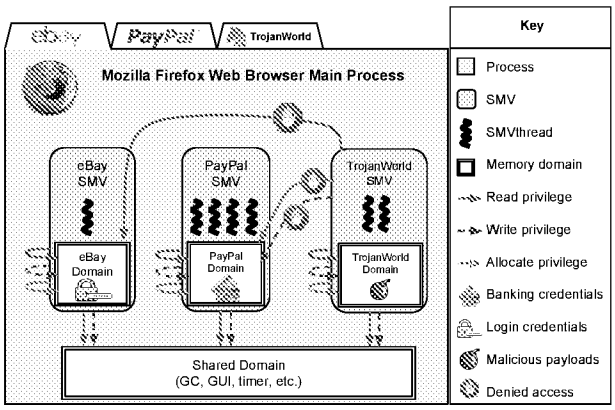


Figure 4: Security-enhanced Firefox.

other, but any write or allocate request from the producer SMV to the consumer domains is prohibited; only reads are permitted. The secure communication domain works as a one-way communication channel for the master `SMVthread` to transmit data to the consumer `SMVthreads` and is inaccessible to the producer `SMVthread` due to the restricted privileges of the producer SMV. In this case, the SMV model strictly enforces memory access boundaries, constraining memory safety bugs to the current component's memory view.

3.6.2 *Case Study: Cherokee Web Server*

Cherokee [12] is a high-performance and light-weight multithreaded web server. To isolate connections, Cherokee uses worker threads to handle incoming requests stored in per-thread connection queues. One worker thread handles all the requests coming from the same connection. However, only one worker thread on the server needs to be compromised to leak sensitive information. To provide an alternative for isolating server workers in different processes, we show how the SMV model can compartmentalize the process memory into memory domains and provide reasonable isolation for the multithreaded Cherokee web server. As shown in Figure 3, the SMV model defines the memory boundaries for worker `SMVthreads` and enforces the memory access privileges to protect the server. The SSL connections are handled only by the `SMVthreads` running in SMV 3 that have the privilege to access worker domain 3, which contains the server's private key. If `SMVthreads` in SMV 2 (handling only HTTP requests) make any attempt to access the private key, the SMV model will reject such invalid memory accesses because of insufficient privileges. In this way, when an exploited worker thread attempts to access memory in an invalid domain, the SMV model detects such invalid accesses and stops further attacks triggered by the memory bugs (e.g., CVE-2004-1097). The original `pthread` Cherokee server does not have this security guarantee since all the threads can access the complete process address space (with unanimously shared permission). We show how accessing invalid memory domains is prevented by the SMV model in Section 5.4.

3.6.3 *Case Study: Mozilla Firefox Web Browser*

The SMV model allows multithreaded web browsers such as Firefox and its JavaScript engine SpiderMonkey to achieve strict compartment isolation enforced by hardware protec-

IBM - Ex. 1002-Part 10 of 10, Page 3374
IPR2025-00379 (IBM v. Croga)

tion, preventing one malicious origin from accessing sensitive data such as bank accounts hosted by another origin. Figure 4 presents an example of how the SMV model can isolate browser tabs in SMVs based on the same-origin policy [37]. With SMVs, the malicious origin TrojanWorld cannot escape from its compartment to access the PayPal banking account (add recipient account by allocating memory or transfer money to attacker's account by writing to memory) or read the user credentials hosted by eBay. Such strong isolation guarantees inspired Google to design Chrome to use process isolation for its rendering process.

4. IMPLEMENTATION

This section details the OS kernel level implementation of the SMV model and discusses its security guarantees. We modified the Linux kernel version 4.4.5 for the x64 architectures to support the SMV model. Table 3 summarizes the component sizes in our prototype.

4.1 SMV Communication Channel

We developed an SMV loadable kernel module (LKM) that allows the user-space SMV API to communicate with our kernel using the Netlink socket family. Once loaded, the SMV LKM is effectively part of the kernel. The SMV LKM works as a dispatcher in the SMV model that sanitizes the messages from the user space SMV API and invokes SMV-related kernel functions.

Security guarantee. The attacker cannot replace our SMV LKM with a malicious SMV LKM to perform a man-in-the-middle attack and escalate permissions for a given `SMVthread`. Such a system-wide change requires the attacker to have root privilege on the system.

4.2 Metadata Management

To efficiently maintain the state of the processes that have `SMVthreads`, we added two major objects to the OS kernel. (1) `memdom_struct`: memory domain metadata for tracing the virtual memory area and the memory domains mappings. (2) `SMV_struct`: the SMV privilege metadata for accessing memory domains. These kernel objects cooperate with each other to maintain the fine-grained privilege information of each SMV in a process.

Security guarantee. The metadata is allocated in kernel memory space and is not mutable by any user space programs without proper privileges through our API. Memory bugs in user space programs cannot affect the integrity of the metadata stored in kernel memory. One of the main sources of kernel 0-day attacks is the use of uninitialized bytes in kernel memory (e.g., CVE-2010-4158) that allows local users to read sensitive information. The SMV model sanitizes the metadata by initializing objects to avoid any potential information leakage from this added attack surface. Our kernel inherits the original kernel's garbage collection system using reference counting to ensure that the additional metadata does not create any dangling pointers.

Table 3: Summary of component sizes.

	LOC†	Source files	Protection level
SMV API	781	6	user space
SMV LKM	443	2	kernel
SMV MM‡	1,717	24	kernel

† Lines of code computed by `cloc`.
‡ SMV MM stands for SMV memory management, which is integrated into the OS memory management subsystem as we show in Figure 1.

4.3 Partially Shared Memory Space

In the SMV model, `SMVthreads` can be perceived as *untrusted* tasks by default. Therefore, our kernel has to *partially* separate the kernel objects; it also maintains the consistent process address space for the SMV model. Overall, our kernel: (1) separates the memory space of SMVs by using a page global directory (`pgd_t`) for each SMV; (2) frees memory for all SMVs when one `SMVthread` frees the process memory; (3) loads thread-private `pgd_t` into the CR3 register during a context switch.

All `SMVthreads` in a process share the *same* `mm_struct` that describes the process address space. Our kernel allocates one `pgd_t` for each SMV in a process and stores all `pgd_ts` in a process's `mm_struct`. `SMVthreads` use their private page tables to locate memory pages, yet their permissions to the same page might differ. Note that we designed SMVs to protect thread stacks as well. To ensure the integrity of the process memory space, the page tables of all SMVs need to be updated when the kernel frees the process page tables or when `kswapd` reclaims page frames. The original kernel avoids reloading page tables during a context switch if two tasks belong to the same process (thus using the same `mm_struct`). We modified our kernel to reload page tables and flush all TLB entries if one of the switching threads is an `SMVthread`. Note that processors equipped with tagged TLBs could mitigate the flushing overhead. However, SMVs do not rely on this hardware optimization feature in order to function correctly (NH).

Based on our extensive experiments, we found that using different `mm_structs` to separate the address space for threads is overkill and could significantly impact the performance for practical applications (LO). This is because all the memory operations related to `mm_struct` need to be synchronized in an aggressive manner in order to maintain the consistent process address space for all threads (e.g., rotating the `vm_area_struct` red-black tree). Using the `clone` syscall without `CLONE_VM` flag to isolate a thread's address space from its parent is another approach. However, this approach has two main drawbacks. First, the kernel creates a new `mm_struct` for the new thread if `CLONE_VM` is not set. This leads to frequent synchronization and imposes overhead. Second, debugging (e.g., GDB [15]) and tracing memory activity (e.g., Valgrind [43]) become extremely difficult: GDB has to be constantly detached from one process and then attached to another in order to debug a parallel program; Valgrind does not support programs with `clone` calls. In contrast, using the same `mm_struct` preserves the system-wide process address space assumption and allows the kernel to separate process address space for threads efficiently.

Security guarantee. The security features of the partially shared memory space rely on the protection guaranteed by the original kernel. The memory management subsystem in the kernel space is completely unknown to user space programs. The attacker has to exploit the permission bits of the page table entries (PTEs) for a thread to break the security features provided by our kernel. We argue that this kind of exploit is highly unlikely without serious DRAM bugs such as rowhammer [22].

IBM - Ex. 1002-Part 10 of 10, Page 3375
IPR2025-00379 (IBM v. Croga)

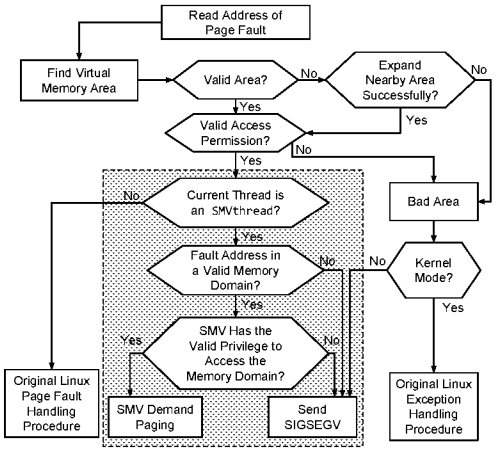


Figure 5: Page fault handler flow chart. The SMV kernel performs additional privilege checks (marked in the gray box).

4.4 Forking SMVthreads

The SMV API uses `pthread_create` to create a regular `pthread` and signals the kernel to convert the `pthread` to an `SMVthread` before the `SMVthread` starts execution. The kernel instructs the `SMVthread` to use the private page tables defined by the SMV that the `SMVthread` runs in. Once an `SMVthread` is created, the kernel turns on the `using_smv` flag stored in the process's `mm_struct` so that future memory operations must go through additional privilege checks.

To simplify porting efforts, the SMV API provides an option to override all `pthread_create` calls and automatically allocate private memory domains for each `SMVthread`.

Security guarantee. The `mm_struct` of a thread is allocated in kernel space and used solely by the kernel. There are no interfaces that allow user space programs to directly or indirectly modify the memory descriptor. This strong isolation between user and kernel space is guaranteed by the trusted OS kernel. In addition, the atomic fork procedure ensures that the attacker cannot intercept the fork procedure and steal the memory descriptor for the malicious thread.

4.5 Page Fault Handler

Figure 5 shows the flow chart of the page fault handler in our kernel. The additional checks are surrounded by the gray box with a dotted line. Our kernel kills the `SMVthread` that triggers an SMV invalid (cf. Section 3.2) page fault by sending a segmentation fault signal. For the privileged `SMVthreads`, our kernel performs *SMV demand paging* to efficiently handle the page faults.

Indeed, since the SMVs use private page tables to separate SMVs' memory views, using the original demand paging routine for `SMVthreads` is insufficient as the page fault handler only updates the page tables for the current SMV, which causes inconsistent process address space. To solve this problem, our kernel tracks all the faulted pages of a process in the *SMV shadow page tables*. The page fault handler deals with faults by using the SMV shadow page tables and then copies the page table entry of the fault from the shadow page tables to the running `SMVthread`'s page tables. Note that one process has only one set of shadow page tables, which only serve as quick reference with no permission implications when `SMVthreads` locate a memory page.

Security guarantee. The page fault handler cannot be accessed, changed, or abused by the attacker as it resides in the lowest level of the software stack. The PTE bits force invalid memory accesses to be trapped to the kernel for the additional privilege checks. To access a privileged memory region, the attacker must first get around the page fault handler. However, such a scenario is infeasible because the kernel memory management subsystem must intervene and prepare the data page before the attacker can access the privileged memory region.

5. EVALUATION

The goal of our evaluation is to demonstrate that *the SMV model has all four desired requirements* when enforcing least privilege memory views for multithreaded applications in practice. We show that the SMV model supports different types of multithreaded programs with flexible policies (GF), requires minimal code changes for legacy software (EU), requires no hardware modifications (NH), and incurs negligible runtime overheads while supporting complex thread interactions and extremely intensive memory allocation/free calls in parallel (LO).

5.1 Experiment Setup

Environment. We measured the performance of our SMV model on a system with Intel i7-4790 CPU with 4 cores clocked at 2.8GHz and 16GB of RAM for our modified x86 64-bit Linux kernel 4.4.5 Ubuntu 14.04.2 SMP (NH). The benchmarks are compiled into two versions: `pthread` and `SMVthread`.

Example policy. `SMVthreads` cannot access privileged memory domains without being explicitly granted the proper privilege. To test this security guarantee in all of our experiments, the number of domains was set to $N + 1$, where N is the number of worker threads and the additional domain serves as a global pool for threads to securely share data. Each worker has its own private memory domain that can only be accessed by itself. We do not claim that the proposed policy is optimal but instead focus on the mechanics to enforce the policy. Setting up alternative policies is possible (GF).

5.2 Robustness Test

To examine the robustness, we tested our modified Linux kernel with the Linux Test Project (LTP) [25] developed and maintained by IBM, Cisco, Fujitsu, SUSE Red Hat, Oracle and others. Specifically, we used the `runltp` script in the LTP package to test the memory management, filesystem, disk I/O, scheduler, and IPC. All stress tests completed without error. We did not observe any system crashes.

5.3 Inspecting Isolation

The SMV model treats invalid memory accesses as segmentation faults. Suppose an attacker's thread triggers a segmentation fault by accessing an invalid memory domain on purpose. The main process will crash to prevent further information leakage. Our SMV library provides detailed memory logs to the programmer. Listing 1 shows an example of the memory activity log. For crashes due to wrong isolation setup, the logs can help the programmer immediately identify the `SMVthread` that accessed the invalid

IBM - Ex. 1002-Part 10 of 10, Page 3376
IPR2025-00379 (IBM v. Croga)

Listing 1: Kernel log obtained by `dmesg` command.

```
1 [smv]Created memdom 2, start addr: 0x00f0f000, end addr: 0x00f10000
2 [smv]SMVthread pid 11157 attempt to access addr 0x00f0f0e0 in memdom 2
3 [smv]Addr 0x00f0f0e0 is protected by memdom 2
4 [smv]Read permission granted to SMVthread pid 11157 in SMV 2
5 [smv]SMVthread pid 11155 attempt to access addr 0x00f0f260 in memdom 2
6 [smv]SMV 1 is not in memdom 2
7 [smv]Detected INVALID memory reference to: 0x00f0f260
8 [smv]INVALID memory request issued by SMVthread pid 11155 in SMV 1
9 [smv]<6>chorekee[11155]: segfault at f0f260 ip 00007f09ba7d6656 error 4
```

protection domain and subsequently rectify the object compartmentalization. In addition, our library provides detailed stack traces for debugging. The logs and stack traces are unreadable by the attacker when debugging mode is disabled. A binary compiled without debugging option makes it impossible for an attacker to learn about memory activity.

5.4 Security Evaluation

To further understand how the SMV model offers strong intra-process isolation, we systematically discuss the security guarantees described in Section 4.

Trusted computing base. The TCB of the SMV model contains the SMV LKM and SMV MM with kernel level protection (cf. Table 3). The SMV API is untrusted and resides in user space as system library. The attacker may try to perform an SMV API call with a malicious intent to escalate permissions for an `SMVthread`. The SMV LKM sanitizes all user space messages sent into the kernel and verifies that the `SMVthread` executing the API call has the correct permissions for the requested change. The attacker may attempt to leverage the misuse of the SMV API to invalidate the memory isolation guarantee provided by the SMV model. Therefore, the security of the application relies on the correctness of the memory isolation setup. Once the memory boundaries are defined, all `SMVthreads` must follow the memory access rules defined by the programmer. Note that unprivileged users without root permission cannot compromise the SMV LKM (cf. Section 4.1 security guarantee).

The SMV model also relies on the privilege level enforcement imposed by the original Linux kernel to make sure that the attacker cannot tamper with the SMV model operating in the kernel space. To bypass the kernel protection, the attacker must hijack the page tables of a privileged thread or modify the metadata stored in the kernel space. The original Linux kernel ensures the integrity of the metadata and memory descriptors for all threads in the system. Using wrong page tables or metadata will cause a thread to be killed once the kernel detects the tainted kernel data structures. Thus, it is impossible for the attacker to exploit the metadata of any thread without kernel 0-day vulnerabilities (cf. Section 4.2 security guarantee).

In addition to the software TCB, the SMV model also relies on the hardware's correctness. The hardware vendors perform significant correctness validation. We believe that the security features offered by sound hardware are unlikely for the attacker to subvert (cf. Section 4.3 security guarantee). Given the extremely small source code base (less than 2000 LOC), we believe that the SMV's TCB could be formally verified.

TOCTTOU attack: stealing page tables. The attacker may attempt to steal the page tables of a privileged thread by hijacking its memory descriptor. We consider an oracle attacker who knows precisely when and how to launch a time of check to time of use (TOCTTOU) attack to steal the page tables of a privileged thread. If the attack succeeds, the attacker's malicious thread will use the hijacked page tables and read sensitive data in the privileged memory domain before the thread crashes. Assume the attacker can fork threads up to the system limit with the objective to hijack the page tables of an about-to-run privileged thread in the fork procedure, which is the only point for the attacker to exploit the `pgd_t` pointer. However, the malicious thread has to wait until the privileged `SMVthread` finishes the page tables setup in order to request the kernel to prepare its unprivileged page tables. Therefore, the attacker cannot intercept the fork procedure and steal the page tables. We conducted an experiment where 1,023 malicious `SMVthreads` tried to hijack the page tables of a privilege `SMVthread`. During the one million runs of the security test, every `SMVthread` used the correct page tables for its memory view (cf. Section 4.4 security guarantee).

Effectiveness of the SMV model. Listing 1 shows the kernel log when an invalid memory access is detected by the SMV model. In this example, the unprivileged `SMVthread` pid 11155 in SMV 1 tries to access memory in the privilege memory domain that stores the server's private key, which is only accessible by `SMVthread` pid 11157 in SMV 2. At line 5, the attempt to read the invalid memory domain triggers the page fault. The kernel rejects the invalid memory request by sending a segmentation fault signal to the unprivileged `SMVthread` pid 11155 at line 9, stopping the unprivileged `SMVthread` from accessing the server's private key. The privilege checks *cannot* be bypassed because the reference monitor is implemented entirely in the page fault handler, and arbitrary page table manipulation is beyond the attacker's scope (cf. Section 4.5 security guarantee).

5.5 PARSEC 3.0 Benchmarks

Overview. The multithreaded PARSEC benchmarks include several emerging workloads with non-trivial thread interaction. Both data-parallel and pipeline parallelization models are covered in the benchmarks with coarse to fine granularity. We used all benchmarks in [7], covering all application domains that were originally multithreaded using the standard `pthreads`. The evaluated benchmarks all employ the producer-consumer pattern (cf. Section 3.6.1) that is pervasive in systems programs and parallelization models. We used the *parsecmgnt* tool in the PARSEC package to run the benchmarks with minimum number of threads set to four for the large inputs as defined by the benchmarks.

Assessment of porting effort. We ported the PARSEC benchmarks by replacing each `pthread` with an `SMVthread` running in its own SMV with a private memory domain. In each program, the main program allocates a shared memory domain to store the working set for `SMVthreads`. The porting procedure consisted of three parts: (1) including the header files to use the SMV API, (2) setting up memory domains and SMVs in the main program, and (3) replacing the `pthreads` with `SMVthreads`. All these changes required *only 2 LOC* changes as the SMV API eliminates the refactoring burden (EU). We needed to add only 1 line to include the header file and another to initialize the main process to use the SMV model. The SMV API automatically intercepts `pthread_create` and `malloc` and replaces them with `smvthread_create` and `memdom_alloc` calls. Therefore, each `SMVthread` could automatically allocate memory in its private memory domain (cf. Section 3.4).

IBM - Ex. 1002-Part 10 of 10, Page 3377
IPR2025-00379 (IBM v. Croga)

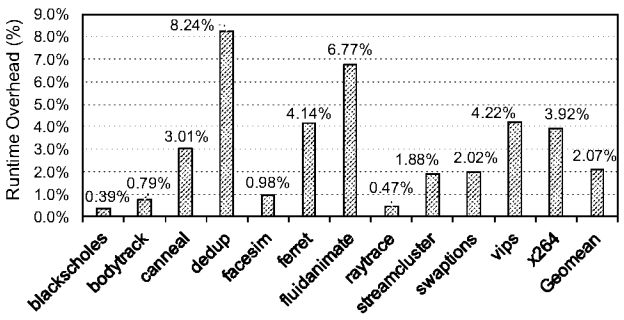


Figure 6: Runtime overhead of the SMV model for the multithreaded applications in the PARSEC benchmark suite.

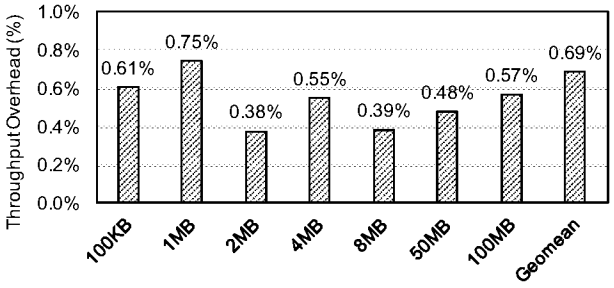


Figure 7: Throughput overhead of Cherokee server.

The security-enhanced PARSEC benchmarks demonstrate a general case that could be applied to any multi-threaded programs written in C/C++ (GF), even with the presence of extremely intensive memory allocation/free calls in parallel. In addition, the intra-process isolation can help prevent attacks that arbitrarily modify data on the stack using malicious threads, e.g., ROP-based attacks. One study has shown that an attacker can perform ROP and use gadgets (16 payloads is enough for > 80% of GNU coreutils [16]) to achieve Turing completeness. Note that ASLR/stack canaries have been proven ineffective to protect against information leakage [20]. With SMV, programmers can secure the system with few changed lines of source code while also handling nontrivial thread interaction, if needed.

Performance. Figure 6 shows the runtime overhead of the ported PARSEC benchmarks with 10 runs for each program. The results show that the SMV model incurs negligible runtime overhead. The overall geometric mean of the runtime overhead is only 2.07% (LO) and the maximum of the runtime overhead occurs for `dedup` due to the huge amount of page faults and the highly intensive parallel memory operations.

5.6 Cherokee Web Server

Overview. The original Cherokee server uses a per-thread memory buffer system for resource management to isolate threads from remote connections. We leveraged the SMV model to provide Cherokee with the OS level privilege enforcement for different server components. Then we compared the throughput of our security-enhanced Cherokee with the original Cherokee.

Assessment of porting effort. We enhanced the security of the Cherokee version 1.2.104 server as illustrated in Section 3.6.2. The user-space SMV library automatically replaced `pthread_create` with `smvthread_create` to create `SMVthreads` for the workers to handle client requests. Each `SMVthread` worker ran in its own SMV with a private memory domain which is inaccessible by other workers. All other shared objects such as the mutex are allocated in a shared memory domain and accessible by all workers. We modified *only 2 LOC* of Cherokee to enforce the least privilege memory access with the SMV model (EU). We believe that the negligible porting effort demonstrates the practicability of the SMV model to protect real-world applications.

Performance. We used ApacheBench to measure the server throughput for the original and security-enhanced Cherokee. Both versions of Cherokee hosted two kinds of web content: (a) social networking web pages, and (b) large streaming files. Based on the total transfer size per page reported in Alexa top one million websites [19], we tested web page sizes from moderate amount of content to abundant media objects (100KB to 8MB). We also evaluated the performance of both servers hosting large streaming files (50MB and 100MB) to show the practicability of our security-enhanced Cherokee. The Cherokee server process created 40 worker threads by default to handle client requests. The client initiated the ApacheBench for 100,000 requests with concurrency level set to four (matches the number of cores). We conducted the experiment 20 times for each object size and present the results in Figure 7. Overall, the SMV model reduced throughput by only 0.69% in exchange for strictly enforcing a least privilege security policy (LO).

We also ported the popular Apache `httpd-2.4.18` with only 2 LOC (EU). Using Apache as a file sharing server (GF) to host large objects with size of 10MB, 50MB, 100MB, and 1GB we conducted the same experiment. Overall, SMVs reduced the throughput of the `httpd` server by only 0.93% (LO). As Cherokee already presents the case for web servers, we exclude the details for Apache `httpd` due to space limitation.

5.7 Mozilla Firefox Web Browser

Overview. The developers of modern web browsers have made tremendous efforts to ensure resource isolation. In 2011, Firefox introduced an abstraction called "compartments" for its JavaScript engine SpiderMonkey to manage JavaScript heaps with security in mind [45]. However, the isolation is not enforced by any mechanism stronger than the compartments' logical boundaries. As a result, any memory corruption can still lead to serious attacks. Here we demonstrate that the SMV model can be easily deployed to protect Firefox's JavaScript engine from memory corruption by confining each compartment to access only its private and the system compartments.

Assessment of porting effort. Firefox uses threads for UI rendering, processing network packets, monitoring browser status, handling JavaScript jobs, etc. In our evaluation, we replaced Firefox 45.0 SpiderMonkey's NSPR (Netscape Portable Runtime) threads with `SMVthreads` running in a private memory domain by adding a new thread type named `PR_SMV_THREAD` to the NSPR library. SpiderMonkey creates 8 threads (1 thread per core + 4 excess threads) in total. We

IBM - Ex. 1002-Part 10 of 10, Page 3378
IPR2025-00379 (IBM v. Croga)

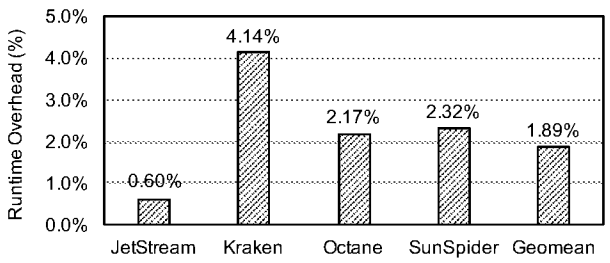


Figure 8: Runtime overhead of security-enhanced Mozilla Firefox web browser.

modified *only 12 LOC* of the entire Firefox source to use the SMV model. Although we designed a per-tab isolation policy, the workloads of individual JavaScript benchmark suites are run in the same tab. For example, JetStream executes 40 benchmark programs in the same tab. The performance numbers faithfully report the overhead for privilege checks as each memory page reference is monitored by our page fault handler.

Performance. We evaluated our security-enhanced Firefox with four popular JavaScript benchmarks and report the numbers in Figure 8. The overall geometric mean for all benchmarks is only 1.89%. The performance numbers report the overhead when Firefox performs additional privilege checks for SpiderMonkey's helper threads. We believe that such negligible overhead numbers allow efficient and strong isolation for multithreaded browsers to be used in practice and provide the Mozilla team an alternative to the ongoing multi-process Firefox *e10s* project [14].

5.8 Limitations

The low performance overhead for checking privileges in the SMV model builds on the virtual memory protection at page granularity. At this point, the SMV model does not guard against unprivileged memory references within the same page as the kernel relies on page table entry (PTE) permission bits. However, poorly organized data structures mixing privileged and non-privileged data within a region are intrinsically insecure and avoided by real-world software (e.g., Hoard [6] memory allocator, connection buffers in Cherokee, worker pools in Apache `httpd`, compartments in Firefox). Therefore, SMVs can be seamlessly integrated into modern software, eliminating the chances for threads to unintentionally access the same page while enforcing memory boundaries at kernel level. Software monitors for byte-granularity protection has inevitably high overhead (e.g., decentralized information flow control systems) since the memory boundary is neither supported by hardware nor the kernel subsystem, making every memory load/store instruction a candidate for a privilege check. In contrast, page-granularity offers strong memory isolation and superior performance with hardware/kernel support.

Although SMVs cannot protect against malicious library threads once they are installed on the system (requires root privilege, which is out of scope), a user can compile any third-party threading libraries to use SMVs, as we demonstrated in the PARSEC benchmark with GThread in `vips` and RTThread in `raytrace`.

6. RELATED WORK

Techniques for achieving intra-process isolation have been studied for decades. In this section, we summarize and compare the related work following a more detailed breakdown.

Memory safety is the goal of many proposals, as memory corruption is the root cause of various well-known software vulnerabilities. We refer the reader to Nagarakatte et al. [32] and Szekeres et al. [42] for two surveys on memory safety. In short, solutions for complete memory safety do not handle intra-process privilege separation problem and impose significant cost for practical systems (cf. LO).

The first generation privilege separation techniques focus on partitioning a process into single-process components. Provos et al. [33] were the first to manually partition OpenSSH by running components in different processes and coordinating them through inter-process communication (IPC). Privtrans [9] automated the retrofitting procedure for legacy software by partitioning one program into a privileged monitor process and an unprivileged slave process with just few programmer-added annotations. Wedge [8] extended the idea of privilege separation to provide fine-grained privilege separation with static capabilities, which was improved by Dune [5] through the Intel VT-x technology (cf. NH) for better performance and by Salus [40] for dynamic security policy. The disadvantage of these first generation techniques is that they lack support for multiple computing entities within the same compartment (cf. LO). This limitation hurts performance of multithreaded programs and restricts the usability of these solutions in practice.

The second generation privilege separation technique Arbiter [47] allowed multiple threads to run in the same compartment. However, Arbiter still faces similar limitations on parallel memory operations and their evaluation does not use multithreaded benchmarks that have intensive memory operations to demonstrate the system's parallelism, even though the design aims at concurrent execution for threads (cf. LO). We identify two major causes of the limitation on Arbiter's parallelism. First, the highly serialized memory management in their user-space library incurs inevitable runtime overhead of up to 400%. Second, the design choice of separating `mm_structs` forces their kernel to aggressively synchronize the global process address space for every thread's memory descriptors. The synchronization costs increase when an application performs intensive memory operations or generates a huge amount of page faults. These limitations on parallelism manifest themselves when running real-world applications (e.g., Firefox) with large inputs (e.g., web server hosting a 100MB file). However, the largest input in the authors' evaluation is only 1MB. In addition, programmers are on their own to partition applications as the solution does not provide assistance in retrofitting applications (cf. EU). As we showed in this paper, the SMV model addresses the limitations of the first and second generation privilege separation techniques without sacrificing security or parallelism.

OS-level abstraction mandatory access control solutions such as SELinux [39], AppArmor [4], and Capsicum [48] protect sensitive data at process/thread granularity. However, fine-grained privilege separation for software objects (e.g., arrays) within a process is not supported in these techniques. On the other hand, SMVs tackle issues for intra-process data protection with capabilities. PerspicuOS [13] separates privileges between trusted and untrusted kernel

IBM - Ex. 1002-Part 10 of 10, Page 3379
IPR2025-00379 (IBM v. Croga)

components defined by kernel developers using an additional layer of MMU. Such an intra-kernel design does not facilitate intra-process privilege separation as SMV does for user-space applications (cf. GF and EU). These security policies are orthogonal to the SMV memory policies and SMVs can be used in conjunction with these techniques to gain additional inter-process protection.

Decentralized information flow control (DIFC) systems allow programmers to associate secrecy labels with data and enforce information flow to follow security policies. HiStar [51] is an OS that fundamentally enforces DIFC that could likely address intra-process isolation. However, HiStar is not based on a general OS kernel such as Linux and thus cannot be incrementally deployed to commodity systems. Moreover, the applications have to be completely rewritten in order to use HiStar (cf. GF and LO). Thus, the solution is infeasible for legacy software (e.g., Firefox) in practice. Flume [23] focuses on process-level DIFC for OS-level abstractions (e.g., files, processes) in UNIX but it does not handle intra-process privilege separation within a multithreaded application. Laminar [34] supports multithreaded application running in its specialized Java virtual machine (cf. GF). However, the additional layer in the software stack and its dynamic checker incur significant runtime overhead (cf. LO). As noted in Section 5.8, byte-granularity checkers in DIFC systems incur high performance overhead in practical applications that have intensive memory operations (cf. LO).

Software-based fault isolation (SFI) [26, 46] isolates software components within the same address space by constructing distinct fault domains for code and data. SFI prevents code from modifying data or jumping to code outside of a fault domain. Native client [3, 50] utilizes SFI with x86 hardware segmentation for efficient reads, writes, control-flow integrity [1], and component isolation. However, the untrusted code is statically associated with a specific fault domain as the approach does not provide simple means of implementing a dynamic and flexible security policy for practical multithreaded applications (cf. EU and LO). In contrast, SMVs offers solutions for programmers to structure the protected memory regions in a dynamic and non-hierarchical manner.

Language-based techniques utilize safe language semantics to provide isolation for applications written in type-safe languages (e.g. [10, 21]) and implement information flow control for objects within a process (e.g. [11, 30, 31]). However, the vast majority of legacy software are still written in an unsafe language for efficiency. As a result, programmers need to completely rewrite their legacy software using safe languages (cf. GF). Ribbons [18] is a programming model developed entirely for user space that provides fine-grained heap isolation for multithreaded applications. While the access privileges of threads are tracked pair-wise between domains hierarchically in user space in Ribbons, the SMV model leverages the OS memory management subsystem to organize the access privileges of threads systematically in kernel space at negligible overhead (cf. EU and LO).

Special hardware support and virtualization technologies is another line of research that seeks for strong isolation of program secrets. Flicker's [28] significant overhead due to its intensive use of the TPM chip (cf. NH) makes it impractical for performance-critical applications (cf. LO). Although TrustVisor [27] mitigates the overhead by a hypervisor and a software-based TPM chip, the system is impractical for applications that require multiple compartments with different capabilities (cf. GF). Fides [41] points out the limitations in TrustVisor and improves it by supporting more flexible secure modules with a dual VM architecture on top of its special hypervisor. Hypervisors can be used for guest OSs (e.g. SMV OS kernel) on a shared host while SMVs (providing a richer API) directly run on bare metal at full speed (cf. GF and LO). The additional software level in the hypervisor introduces overheads as the VMM intervenes for the guest OSs page tables, causing TLB cache misses. Recent studies [2, 17, 29] by Intel indicate that hardware support for secure computing will become available on mainstream X86 environments in the near future. Intel Software Guard Extensions (SGX) is a mechanism to ensure confidentiality and integrity (but not availability) of a trusted unprivileged software module under an untrusted OS with limited trusted hardware. SGX protects one component from possible interaction using an "enclave" enforced by hardware. Although the goal of Intel SGX is similar to SMVs, our pure-software solution allows SMVs to be adopted by any OSs that have MMU subsystems with commodity hardware (cf. NH). Loki's [52] tagged memory architecture, CODOMs' [44] tagged pages, and CHERI's [49] capability registers can isolate modules into separate domains with efficient access protection check logic. But these approaches require hypothetical hardware support which make them incompatible with commodity systems (cf. NH).

7. CONCLUSIONS

We have presented the design, implementation, and evaluation of SMVs, a programming abstraction that allows efficient memory compartmentalization across concurrent threads. SMVs yield a comprehensive architecture with all four desired requirements – *genericity and flexibility*, *ease of use*, *no hardware modifications*, and *low runtime overhead* – for efficient fine-grained intra-process memory separation in multithreaded applications. Our performance evaluation demonstrates that the SMV model imposes negligible overhead in exchange for greatly improved security guarantees, enforcing intra-process isolation for concurrent threads. The runtime overhead of the multithreaded benchmark PARSEC for using the SMV model is only 2.07% overall with only 2 LOC changes. For popular web servers, the reduction in throughput is only 0.69% overall for Cherokee and 0.93% overall for Apache `httpd`. Both applications required only 2 LOC changes. We also showed that the real-world web browser Firefox can be easily ported to the SMV model with only 1.89% runtime overhead overall, requiring only 12 LOC modifications to the large code base (13M LOC). The simplicity of the porting effort allows legacy software to be quickly adapted to the SMV model. In summary, we believe that the SMV model can greatly reduce the vulnerabilities caused by improper software component isolation and encourages more research on the efficient and practical intra-process isolation for general multithreaded applications.

8. ACKNOWLEDGEMENTS

This work was supported by NSF TC-1117065, NSF TWC-1421910, and NSF CNS-1464155. P. Eugster was partly supported by European Research Council under grant FP7-617805 "LiVeSoft – Lightweight Verification of Software".

IBM - Ex. 1002-Part 10 of 10, Page 3380
IPR2025-00379 (IBM v. Croga)

References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow Integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 340–353, New York, NY, USA, 2005. ACM.

[2] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative Technology for CPU based Attestation and Sealing. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, New York, NY, USA, 2013. ACM.

[3] J. Ansel, P. Marchenko, U. Erlingsson, E. Taylor, B. Chen, D. L. Schuff, D. Sehr, C. L. Biffle, and B. Yee. Language-independent Sandboxing of Just-in-time Compilation and Self-modifying Code. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 355–366, New York, NY, USA, 2011. ACM.

[4] AppArmor. https://wiki.ubuntu.com/AppArmor.

[5] A. Belay, A. Bittau, A. Mashtizadeh, D. Terei, D. Mazières, and C. Kozyrakis. Dune: Safe User-level Access to Privileged CPU Features. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 335–348, Berkeley, CA, USA, 2012. USENIX Association.

[6] E. D. Berger, K. S. McKinley, R. D. Blumofe, and P. R. Wilson. Hoard: A Scalable Memory Allocator for Multithreaded Applications. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX, pages 117–128, New York, NY, USA, 2000. ACM.

[7] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[8] A. Bittau, P. Marchenko, M. Handley, and B. Karp. Wedge: Splitting Applications into Reduced-privilege Compartments. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 309–322, Berkeley, CA, USA, 2008. USENIX Association.

[9] D. Brumley and D. Song. Privtrans: Automatically Partitioning Programs for Privilege Separation. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 5–5, Berkeley, CA, USA, 2004. USENIX Association.

[10] C. Bryce and C. Razafimahefa. An Approach to Safe Object Sharing. In *Proceedings of the 15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '00, pages 367–381, New York, NY, USA, 2000. ACM.

[11] W. Cheng, D. R. K. Ports, D. Schultz, V. Popic, A. Blankstein, J. Cowling, D. Curtis, L. Shrira, and B. Liskov. Abstractions for Usable Information Flow Control in Aeolus. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 12–12, Berkeley, CA, USA, 2012. USENIX Association.

[12] Cherokee Web Server. http://cherokee-project.com/.

[13] N. Dautenhahn, T. Kasampalis, W. Dietz, J. Criswell, and V. Adve. Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 191–206, New York, NY, USA, 2015. ACM.

[14] Multiprocess Firefox. https://developer.mozilla.org/en-US/Firefox/Multiprocess_Firefox.

[15] GDB: The GNU Project Debugger. https://www.gnu.org/software/gdb/.

[16] C. Giuffrida, A. Kuijsten, and A. S. Tanenbaum. Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 40–40, Berkeley, CA, USA, 2012. USENIX Association.

[17] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. Using Innovative Instructions to Create Trustworthy Software Solutions. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 11:1–11:1, New York, NY, USA, 2013. ACM.

[18] K. J. Hoffman, H. Metzger, and P. Eugster. Ribbons: A Partially Shared Memory Programming Model. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '11, pages 289–306, New York, NY, USA, 2011. ACM.

[19] Interesting stats based on Alexa Top 1,000,000 Sites. http://httparchive.org/interesting.php.

[20] H. Hu, Z. L. Chua, S. Adrian, P. Saxena, and Z. Liang. Automatic Generation of Data-Oriented Exploits. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 177–192, Washington, D.C., Aug. 2015. USENIX Association.

[21] K. Kawachiya, K. Ogata, D. Silva, T. Onodera, H. Komatsu, and T. Nakatani. Cloneable JVM: A New Approach to Start Isolated Java Applications Faster. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, VEE '07, pages 1–11, New York, NY, USA, 2007. ACM.

[22] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 361–372, Piscataway, NJ, USA, 2014. IEEE Press.

[23] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information Flow Control for Standard OS Abstractions. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 321–334, New York, NY, USA, 2007. ACM.

[24] A. Kurmus and R. Zippel. A Tale of Two Kernels: Towards Ending Kernel Hardening Wars with Split Kernel. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1366–1377, New York, NY, USA, 2014. ACM.

[25] Linux Test Project. http://sourceforge.net/projects/ltp/.

IBM - Ex. 1002-Part 10 of 10, Page 3381
IPR2025-00379 (IBM v. Croga)

[26] S. McCamant and G. Morrisett. Evaluating SFI for a CISC Architecture. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[27] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 143–158, Washington, DC, USA, 2010. IEEE Computer Society.

[28] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 315–328, New York, NY, USA, 2008. ACM.

[29] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1, New York, NY, USA, 2013. ACM.

[30] A. Mettler, D. Wagner, and T. Close. Joe-E: A Security-Oriented Subset of Java. In *Network and Distributed Systems Symposium*, NDSS 2010. Internet Society, 2010.

[31] A. C. Myers. JFlow: Practical Mostly-static Information Flow Control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 228–241, New York, NY, USA, 1999. ACM.

[32] S. Nagarakatte, M. M. K. Martin, and S. Zdancewic. Everything You Want to Know About Pointer-Based Checking. In *1st Summit on Advances in Programming Languages (SNAPL 2015)*, volume 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 190–208, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[33] N. Provos, M. Friedl, and P. Honeyman. Preventing Privilege Escalation. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 16–16, Berkeley, CA, USA, 2003. USENIX Association.

[34] I. Roy, D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel. Laminar: Practical Fine-grained Decentralized Information Flow Control. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 63–74, New York, NY, USA, 2009. ACM.

[35] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sept 1975.

[36] J. H. Saltzer. Protection and the Control of Information Sharing in Multics. *Commun. ACM*, 17(7):388–402, July 1974.

[37] Same-origin Policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.

[38] SECure COMPuting with filters. https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt.

[39] SELinux. https://wiki.centos.org/HowTos/SELinux.

[40] R. Strackx, P. Agten, N. Avonds, and F. Piessens. Salus: Kernel Support for Secure Process Compartments. *EAI Endorsed Transactions on Security and Safety*, 15(3), 1 2015.

[41] R. Strackx and F. Piessens. Fides: Selectively Hardening Software Application Components Against Kernel-level or Process-level Malware. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 2–13, New York, NY, USA, 2012. ACM.

[42] L. Szekeres, M. Payer, T. Wei, and D. Song. SoK: Eternal War in Memory. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 48–62, Washington, DC, USA, 2013. IEEE Computer Society.

[43] Valgrind. http://valgrind.org/.

[44] L. Vilanova, M. Ben-Yehuda, N. Navarro, Y. Etsion, and M. Valero. CODOMs: Protecting Software with Code-centric Memory Domains. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 469–480, Piscataway, NJ, USA, 2014. IEEE Press.

[45] G. Wagner, A. Gal, C. Wimmer, B. Eich, and M. Franz. Compartmental Memory Management in a Modern Web Browser. In *Proceedings of the International Symposium on Memory Management*, ISMM '11, pages 119–128, New York, NY, USA, 2011.

[46] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient Software-based Fault Isolation. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, SOSP '93, pages 203–216, New York, NY, USA, 1993. ACM.

[47] J. Wang, X. Xiong, and P. Liu. Between Mutual Trust and Mutual Distrust: Practical Fine-grained Privilege Separation in Multithreaded Applications. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 361–373, Santa Clara, CA, July 2015. USENIX Association.

[48] R. N. Watson, J. Anderson, B. Laurie, and K. Kennaway. Capsicum: Practical Capabilities for UNIX. In *USENIX Security 2010*, pages 29–46, 2010.

[49] J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe. The CHERI Capability Model: Revisiting RISC in an Age of Risk. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 457–468, Piscataway, USA, 2014. IEEE Press.

[50] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *2009 30th IEEE Symposium on Security and Privacy*, pages 79–93, May 2009.

[51] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making Information Flow Explicit in HiStar. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 19–19, Berkeley, CA, USA, 2006. USENIX Association.

[52] N. Zeldovich, H. Kannan, M. Dalton, and C. Kozyrakis. Hardware Enforcement of Application Security Policies Using Tagged Memory. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 225–240, Berkeley, CA, USA, 2008. USENIX Association.

IBM - Ex. 1002-Part 10 of 10, Page 3382
IPR2025-00379 (IBM v. Croga)

Editors: Mohamed Kaâniche, mohamed.kaaniche@laas.fr | Richard Kuhn, d.kuhn@nist.gov

Trusted Execution Environments: Properties, Applications, and Challenges

Patrick Jauernig, Ahmad-Reza Sadeghi, Emmanuel Stapf | TU Darmstadt

Software attacks on modern computer systems have been a persisting challenge for several decades, leading to a continuous arms race between attacks and defenses. As a first line of defense, operating system kernels enforce process isolation to limit potential attacks to only the code containing the vulnerabilities. However, vulnerabilities in the kernel itself (for example, various vulnerabilities found by Google Project Zero), side-channel attacks,[1] or even physical attacks[2] can be used to undermine process isolation.

To provide strong isolation between untrusted software components, including the operating systems itself, a number of hardware-assisted security solutions have been proposed that aim to significantly increase the protection level that defenses against software exploitation have failed to provide. Prominent examples are trusted platform modules (TPMs), hardware security modules (HSMs), secure elements (SEs), and trusted execution environments (TEEs).

TPMs are dedicated, secure cryptoprocessors tied to the individual device, offering services like key generation, encryption, or authenticated integrity measurements called *attestation*. Yet, they do not offer isolation for sensitive applications. In contrast, HSMs also allow the running of sensitive code and are not bound to a device because they are typically implemented as plug-in cards (like peripheral component interconnect cards) or attachable external devices. SEs instead implement the HSM functionality as a coprocessor directly on the board, which offers only low performance due to size and energy constraints. Moreover, third-party development on SEs is heavily restricted by manufacturers. An alternative to these solutions is TEEs, which are commonly integrated tightly into the system on chip (SoC) and leverage existing SoC resources, enabling them to provide cryptographic primitives and isolated execution with much higher flexibility than that provided by any other solution. Thus, industry is pushing TEEs on all fronts, from cloud servers and desktop computers over smartphones to low-energy embedded devices.

However, after years of hardware security research, we can conclude that existing solutions are still insufficient: Deployed hardware solutions like TPMs or TEEs, are not used widely or have been attacked through various side-channels or through recently emerging cross-layer attacks that exploit hardware vulnerabilities from software, as demonstrated by attacks like CLKScrew or Foreshadow. The Hack@DAC hardware security contest[3] revealed a systematic protection gap in current chip designs: Existing verification approaches may fail to detect certain classes of vulnerabilities in the Register Transfer Level (RTL) of hardware description code.

A promising recent approach is to manage hardware via a small software trusted computer base (TCB) to create strongly isolated unprivileged TEEs, (e.g., by building user-space enclaves for mobile devices based on ARM TrustZone[4]), or to leverage a completely open source design (for example, for the RISC-V platform[5]).

In this article, we review selected existing industrial and promising academic TEE solutions and briefly discuss the impact of deployed solutions, their strengths and shortcomings, and new research directions. The results of this survey are summarized in Table 1.

TEEs Proposed by Industry

The high demand for hardware-assisted security architectures led to the development of TEE architectures across many platforms. In this section, we present a subset of these TEE architectures of major processor brands, namely, Intel, AMD, and ARM.

Digital Object Identifier 10.1109/MSEC.2019.2947124
Date of current version: 19 March 2020

Copublished by the IEEE Computer and Reliability Societies
1540-7993/20©2020IEEE

IBM - Ex. 1002-Part 10 of 10, Page 3383
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:26:57 UTC from IEEE Xplore. Restrictions apply.

Intel SGX: Protecting Apps in User Space

Intel introduced Software Guard Extensions (SGX)[6] with their Skylake microarchitecture in 2015. From the adversary model perspective, any software, including the OS, (and even some hardware components) may be considered untrusted. Hence, SGX's TCB only comprises the CPU hardware and its microcode.

In SGX, instances of a TEE, called *enclaves* [as shown in Figure 1(a)], are used to execute sensitive program code in user space, isolated from a potentially malicious OS or hypervisor. Each enclave is bundled with a regular nonsensitive application (App) that invokes the enclave as a child process. During the enclave setup, the integrity of the enclave code is verified (attested); i.e., an authenticated measurement, typically a binary hash, of the code loaded into the enclave is reported either locally or remotely. When an enclave is executed, it shares its virtual address space with its host process. The enclave memory management is entirely performed by the untrusted OS. Moreover, the OS provides exception handling and input–output services to the enclave. Hardware features protect the enclave code and data (e.g., its page tables) from an unauthorized access by the host process, the OS, or even the hypervisor.

SGX is implemented inside the CPU through microcode and minimal

Table 1. A comparison of presented TEE architectures.

	Intel SGX	AMD SEV	ARM TrustZone	Sanctum	Sanctuary
Commercial/academic	Commercial	Commercial	Commercial	Academic	Academic
Target devices	Client PCs	Servers	Mobile devices	Undefined	Mobile devices
Trust anchor	CPU hardware and microcode	Platform security processor	TZ hardware and ARM trusted firmware	CPU hardware and security monitor	TZ hardware and ARM trusted firmware
Cache side-channel protection	No	No	No	Yes	Yes (for sanctuary instances)
Multiple security domains	Yes	Yes	No	Yes	Yes
Secure peripherals	No	No	Yes	No	Yes

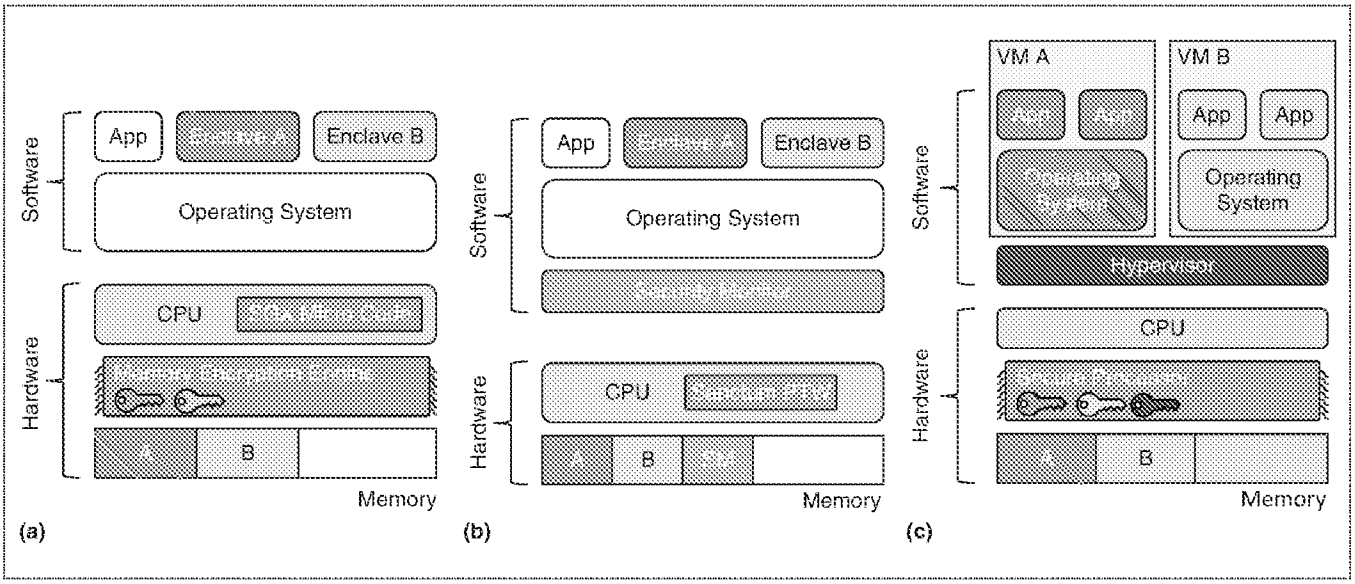


Figure 1. High-level architectural views. The (a) Intel SGX, (b) Sanctum, and (c) AMD SEV. PTW: page table walker; VM: virtual machine.

IBM - Ex. 1002-Part 10 of 10, Page 3384
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:26:57 UTC from IEEE Xplore. Restrictions apply.

hardware changes that are made at the page table walker (PTW). All enclave code or data leaving the CPU is encrypted, using a new hardware component called the *Memory Encryption Engine* (*MEE*) that allows SGX to protect enclaves from direct memory access (DMA) attacks from malicious peripherals. The MEE is also used to persistently store the enclave states after their execution.

In the first and only commercially available version of SGX, software cache side-channel attacks are not considered in the adversary model. However, a broad spectrum of recent attacks showed that side-channel attacks are a much more crucial threat for SGX than expected.[1,7]

AMD SEV: Moving to the Cloud

In 2017, AMD introduced its own TEE, Secure Encrypted Virtualization (SEV)[8] [shown in Figure 1(c)], which follows an entirely different approach than Intel SGX. Whereas SGX focuses on micro services (e.g., DRM, or cryptographic functionality), AMD designed a TEE for the cloud: It offers better performance for intensive workloads and is transparent to the software running in an SEV-enabled virtual machine (VM)[8]. Thus, its adversary model is also centered around the cloud (i.e., after setting up a VM, its memory is isolated even from the hypervisor).

To isolate a VM, Secure Encrypted Virtualization (SEV) encrypts each VM transparently with an individually generated encryption key. Access to these keys is limited to hardware; thus, the hypervisor or any other software component outside the VM cannot interfere with the encryption. Although SGX enclaves' isolation is enforced by the memory management unit, SEV leverages Secure Memory Encryption (SME) to encrypt VM memory to protect against physical adversaries and to protect against privileged software or other VMs. SME is a transparent hardware memory encryption feature that encrypts data before storing it in memory and decrypts the data before loading it to the cache. SME stores encryption keys in a dedicated platform security processor based on the ARM architecture, and only allows operations on memory that has the matching tag to a key. This tag is automatically assigned to all memory that belongs to a VM (using the VM ASID identifier); thus, external accesses (for example, by the hypervisor) will see only encrypted data. A bit flag set indicates that a page should be encrypted; hence, VMs can also have unencrypted pages for shared memory.

Later, SEV-ES (Encrypted State) is introduced to add protection for CPU register contents which prevents information leakage to the hypervisor when a VM is suspended. As in SGX, cache side-channels are not considered in the adversary model of AMD SEV. Recently, security researchers from Google found a vulnerability in SEV's elliptic-curve implementation, allowing them to recover the private key that is used to derive individual VM keys, effectively breaking the encryption. This issue has been fixed by a firmware update.[9]

ARM TrustZone: Protecting Mobile Devices

Protection of sensitive program code on mobile devices is also of vital importance and will be even more vital due to emerging use cases like mobile ID or the digital car key. Since 2004, ARM TrustZone[10] has provided the hardware primitives to implement TEE security architectures on mobile devices. In Figure 2(a), the typical design of a security architecture that uses TrustZone is shown.

TrustZone assumes a strong software attacker able to compromise the untrusted commodity operating system running regular apps. However, physical attackers are out of scope. TrustZone separates the system into two worlds: the normal world, containing the untrusted commodity OS and all nonsensitive applications, and the so-called secure world. The secure world comprises the sensitive applications [Trusted Apps (TAs)] and the Trusted OS (TOS). TAs can come either standalone or bundled with a nonsensitive application running in normal world, whereas the TOS provides usual process isolation and system services to the TAs. The secure world represents the single TEE of the system, unlike Intel SGX or AMD SEV, which can all provide multiple separated TEE instances. By including device drivers into the TOS, TrustZone can establish secure communication channels from peripherals (e.g., fingerprint sensors) to sensitive applications. TrustZone even allows secure DMA by temporarily assigning memory regions exclusively to one or multiple SoC components (for example, the CPU, GPU, or LCD controller).

TrustZone's core idea, the world switch, is implemented in the most privileged software component, called *Trusted Firmware* (*TF*), which also handles the communication between both worlds. The TF represents the software TCB of the system together with all code running in the secure world. The separation between the normal and the secure world is achieved in hardware by a set of security enhancements of the CPU, the system bus, and additional components on the SoC such as the memory controller.

TrustZone does not provide cache partitioning and, therefore, cannot protect against cache side-channel attacks in general. Yet, the main weakness of classic TrustZone-based security architectures is their single-TEE nature. As a result, trust relationships have to be established between the device vendor and every developer. Establishing this trust generates high costs for security assessments and management overhead, which restrains development of new secure mobile services.

IBM - Ex. 1002-Part 10 of 10, Page 3385
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:26:57 UTC from IEEE Xplore. Restrictions apply.

TEEs Proposed by Academia

Limitations of existing TEEs motivated academic developers to propose their own hardware-assisted security architectures. In this section, we present Sanctum and Sanctuary which, on an abstract level, aim to improve Intel SGX and ARM TrustZone, respectively.

Sanctum: Extending Enclave Protection

The Sanctum security architecture was proposed in 2016 by Costan et al.[4] for the open RISC-V architecture [shown in Figure 1(b)]. Sanctum resembles Intel SGX regarding its adversary model and high-level concept of user-space enclaves (or TEE instances) but provides resilience to cache side-channel attacks. Like SGX, a Sanctum enclave comes bundled with a nonsensitive host application that invokes the enclave. The untrusted OS still provides exception handling and input–output services to the enclaves. However, in Sanctum, each enclave manages its own page tables.

In contrast to SGX, the enclave setup and other security critical functionalities are not implemented in microcode but in a software component called the *Security Monitor* (*SM*), which represents the software TCB of the system. The SM runs in the most privileged software level of a RISC-V processor, known as the *machine level*, and enables the enclave integrity verification locally or remotely before the enclave is executed. Sanctum enforces the isolation of the enclave code and data by introducing small hardware changes at the PTW of the CPU. The hardware changes guarantee on one hand that the OS cannot access enclave memory and, on the other hand, that an enclave cannot access the OS memory by changing its page tables. The circuitry added around the PTW prevents a successful address translation of virtual memory addresses that would map to physical memory addresses that the current execution context is not allowed to access. The protection from cache side-channel attacks is achieved with cache partitioning, which is implemented through memory page coloring. Cache partitioning allows assignment of cache lines of the last-level cache, which is shared between cores, exclusively to single enclaves.

In contrast to SGX, Sanctum does not encrypt the enclave code and data in the main memory. However, Sanctum provides a basic DMA attack protection by making small changes to the memory controller that allow Sanctum to restrict DMA to a certain region of the memory.

Sanctuary: Providing Multidomain Security

The limitations of TrustZone inspired the TEE architecture Sanctuary, which was proposed by Brasser et al. in 2019.[3] In contrast to currently deployed TrustZone-based security architectures, Sanctuary, which is shown in Figure 2(b), provides an arbitrary number of TEEs (or enclaves) on ARM-based devices. Moreover, Sanctuary tolerates malicious code in the TEEs, as well as cache side-channel attacks.

Sanctuary relies on TrustZone and, therefore, inherits many of its security features, such as world separation, secure boot, DMA access control, or secure communication to peripherals. In contrast to TrustZone, Sanctuary removes sensitive third-party applications from the secure world, which contains only security primitives provided by the device vendor. Instead, single sensitive apps are temporarily isolated on physical cores in a Sanctuary Instance. The remaining CPU cores execute the normal world and secure world code. Thus, no trust relationship has to be established between the device vendor and the sensitive app developer, which

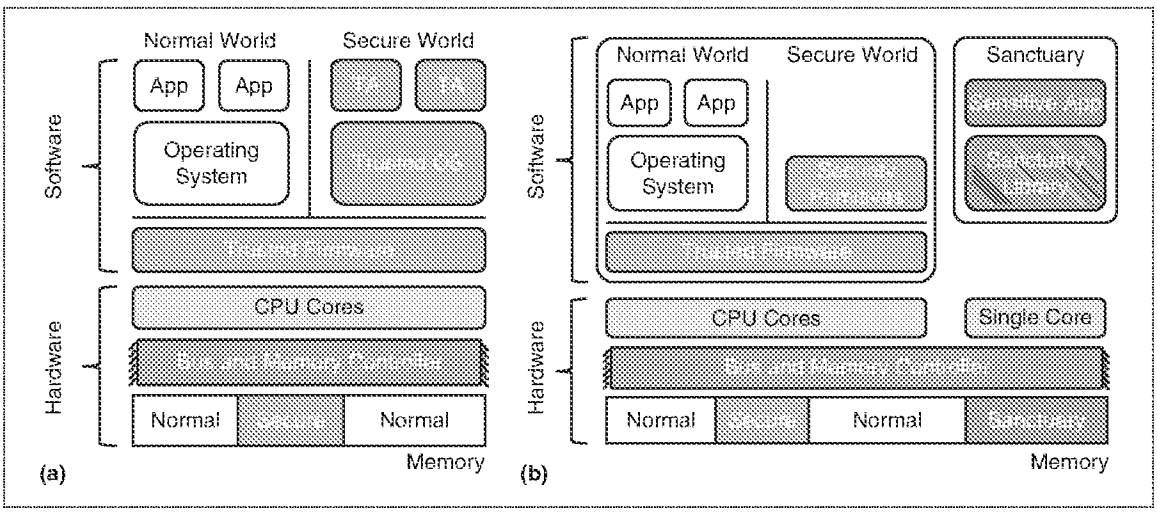


Figure 2. High-level architectural views. (a) The ARM TrustZone and (b) the Sanctuary.

IBM - Ex. 1002-Part 10 of 10, Page 3386
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:26:57 UTC from IEEE Xplore. Restrictions apply.

enables an open use of TrustZone. The isolated sensitive apps can communicate with the untrusted OS in the normal world and with the security primitives in the secure world. The communication is facilitated by a software component, called *Sanctuary Library (SL)*, which provides OS services to the SA. The TF and secure world code represent the software TCB of the system. However, in Sanctuary, no third-party code is included in the TCB. The SL, which is also provided by the device vendor, is not part of the system TCB because it is running in the normal world.

The isolation of the physical CPU cores is enforced using a security feature of the TrustZone-enabled memory controller provided by ARM. In Sanctuary, this feature is exploited to temporarily assign memory regions exclusively to physical CPU cores when they are selected to execute a sensitive app. The configuration of the memory controller, as well as the setup of the enclaves, is performed and verified by the security primitives in the secure world.

Sanctuary cannot provide hardware cache partitioning. However, software cache side-channel attacks on sensitive apps are prevented by flushing the core-exclusive caches when setting up an enclave and by excluding its code and data from the shared last-level cache.

One of the biggest challenges of existing TEE architectures is the threat of microarchitectural attacks. The most prominent attacks in recent years were cache side-channel attacks (see Brasser et al.[1]) and transient execution attacks like Spectre and Meltdown.[7] Most known microarchitectural attacks involve resources that are shared on the system and responsible for optimizing the performance of the system (for example, the data caches[1] or the Branch Target Buffer).[7]

A modern TEE architecture must provide strong isolation among different security domains. However, preventing the sharing of resources might lead to a drastic decrease in system performance, which would make a TEE architecture impractical. At the same time, increasing the number of resources and partitioning them between the security domains might lead to a high hardware overhead. Therefore, designers of TEE architectures need to find a valuable tradeoff between domain isolation and resource sharing.

Moreover, designers must keep in mind that the known microarchitectural attacks might not be the end of the story. Cross-layer attacks based on hardware bugs can be diverse, as shown at the hardware security contests Hack@DAC in 2018 and 2019.[3]

References

1. F. Brasser, U. Muller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software Grand Exposure: SGX cache attacks are practical," in *Proc. 11th USENIX Workshop Offensive Technologies*, Vancouver, British Columbia, 2017. [Online]. Available: https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser
2. "Fault attacks on secure chips: From glitch to flash," in *Proc. Design and Security of Cryptographic Algorithms and Devices (ECRYPT II)*, Albena, Bulgaria, May 29–June 3, 2011. [Online]. Available: https://studylib.net/doc/18400951/fault-attacks-on-secure-chips–from-glitch-to-flash
3. G. Dessouky, D. et al., "Hardfails: Insights into software-exploitable hardware bugs," in *Proc. 28th USENIX Security Symp.*, Santa Clara, CA, 2019. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky
4. F. Brasser, D. Gens, P. Jauernig, A. Sadeghi, and E. Stapf, "Sanctuary: Arming TrustZone with user-space enclaves," in *Network and Distributed System Security Symp.*, San Diego, CA, 2019. doi: 10.14722/ndss.2019.23448.
5. V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Security Symp.*, Austin, TX, 2016, pp. 213–230.
6. Intel. "Intel® Software Guard Extensions programming reference," 2014. Accessed on: Aug. 9, 2019. [Online]. Available: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf
7. C. Canella, J. V. et al., "A systematic evaluation of transient execution attacks and defenses," in *Proc. 28th USENIX Security Symp.*, Santa Clara, CA, 2019, pp. 249–266.
8. S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of Intel SGX and AMD Memory Encryption technology," in *Proc. 7th Int. Workshop Hardware and Architectural Support Security and Privacy*, New York, 2018.
9. Seclists.org, CVE-2019-9836, Accessed on: Aug. 8, 2019. [Online]. Available: https://seclists.org/fulldisclosure/2019/Jun/46, 2019.
10. ARM Limited. "Security technology: Building a secure system using TrustZone® technology," 2008. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html

Patrick Jauernig is a researcher with TU Darmstadt. Contact him at patrick.jauernig@trust.tu-darmstadt.de.

Ahmad-Reza Sadeghi is a professor with TU Darmstadt. Contact him at ahmad.sadeghi@trust.tu-darmstadt.de.

Emmanuel Stapf is a researcher with TU Darmstadt. Contact him at emmanuel.stapf@trust.tu-darmstadt.de.

IBM - Ex. 1002-Part 10 of 10, Page 3387
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:26:57 UTC from IEEE Xplore. Restrictions apply.

2014 IEEE 11th Intl Conf on Ubiquitous Intelligence & Computing and 2014 IEEE 11th Intl Conf on Autonomic & Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Associated Symposia/Workshops

An Architecture for Virtualization-Based Trusted Execution Environment on Mobile Devices

Young-Woo Jung, Hag-Young Kim
Cloud Computing Department
Electronics and Telecommunications Research Institute
Daejeon, Republic of Korea
{jungyw, h0kim}@etri.re.kr

Sang-Wook Kim
Mobile Media Laboratory
Kyungpook National University
Daegu, Republic of Korea
swkim@knu.ac.kr

***Abstract*—In this paper, we present an architecture for a trusted execution environment on mobile devices that allows applications with a wide range of security requirements to run safely in an isolated environment, by using a mobile virtualization technology. These applications can take advantage of the semantics of running on secure area which is isolated from non-secure area that suffers from hacking, malignant code, or the like, while retaining the ability to run side-by-side with normal applications on a general execution environment. We achieve this synthesis by use of a *mobile virtual machine monitor* (mVMM) that partitions single mobile hardware platform into the separated and isolated virtual machines (VMs), providing the trusted execution environment and the trusted paths. In VM on which the secure OS runs, authentication credentials (e.g. private keys) for electronic transactions and security sensitive data are stored and security sensitive processing is executed with no external network interfaces provided and only with a secure communication channel provided by mVMM. We explore the strengths and limitations of this architecture by describing and analyzing our prototype implementation and a simple mobile payment service that can be one of the important applications for the trusted execution euvironment. Through the architecture analysis, the proposed architecture can provide a reasonably trustworthy execution environment to a user in the run-time execution point of view.**

Keywords-trusted execution environment; virtualization; virtual machine monitor; security; isolated execution; trusted path; m-payment

I. INTRODUCTION

With the recent development of mobile technologies, smart phones, tablets and other smart devices have been released and they have provided mobile commerce to reach customers anywhere and at any time. Also, we have been able to access our bank account, personal networks and confidential documents wherever and whenever we need. Nowadays, they are at a stage of mobile convergence which accommodates an area of new services as well as their own traditional area. On the other hand, these added richness and connectivity also poses a problem such as increased malwares, because of untrustworthy operating systems (OSes) for mobile devices. Therefore, in order to take advantage of the potential of these devices and prevent mobile device users from putting their financial and personal security at risk, we must be able to lessen those risks and offer a safe execution environment for the authorized security software, known as trusted applications, providing end-to-end security by enforcing protection, confidentiality, integrity and data access rights.

Today's mobile device ecosystem with hardware and software consists of multiple stake-holders, mainly comprising hardware manufacturers, telecommunications providers, application developers, and the device owner. While hardware manufacturers and telecommunication providers have led to their own trust models including add-on hardware and corresponded software, application developers and users such as banks, credit card companies, or credential data service providers, who really require security properties, mostly cannot benefit from the security features without the support of hardware manufacturers and telecommunications providers. The only way they have attempted for security has been just to install additional software as much as possible, just like various antivirus software, keyboard protection software, intrusion detection systems, a complex password, and so on. However, it results in making the mobile OS much more complex and difficult to use, but still can't prevent attacks from inside caused by using implementation error or user unawareness [6]. In this circumstance, enabling "secure execution" on mobile devices can be interpreted as the alternative solution for security-conscious application developers and users. There is a set of features intended to enable the secure execution on mobile devices, including the isolated execution, the secure storage, the remote attestation, the secure provisioning, and the trusted path [1].

A Trusted Execution Environment (TEE) is an environment where any code executed inside the TEE is trusted in authenticity and integrity and the other assets inside the TEE are also protected in confidentiality [24]. This is closely related to the features of the secure execution. In this paper, we focus on the isolated execution and the partially trusted path, because we believe that they can be the most critical security features for the secure execution to provide TEE to users, although other features need to be also considered as important research topics.

For the isolated execution of trusted applications on a mobile device, multiple hardware primitives exist on ARM architecture devices today. ARM first introduced their TrustZone Security Extensions in 2013 [16, 17], enabling a

978-1-4799-7646-1/14 $31.00 © 2014 IEEE
DOI 10.1109/UIC-ATC-ScalCom.2014.21

IEEE computer society

IBM - Ex. 1002-Part 10 of 10, Page 3388
IPR2025-00379 (IBM v. Croga)

"two-world" model, whereby both secure and non-secure software can coexist on the same processor. One of the related solutions is Samsung Knox which provides an isolated environment for corporate applications [25]. ARM also announced support for Hardware Virtualization Extensions since ARM Cortex-A15 CPU family [18]. These extensions enable the traditional virtualization solutions, such as OKL4 [20], XenARM [19], and KVMARM [21, 22], more adjustable to mobile devices.

In this paper, we propose the virtualization-based TEE to provide the isolated execution and the trusted path. However, we don't want to insist that the proposed architecture is the only and best way to support the TEE on mobile devices, but the recent development in the virtualization technology has created new opportunities in the security feature of mobile devices and this technology can be more applicable to application developers and users. Although the proposed architecture is designed and implemented especially for mobile devices, it also can be adjustable for desktop or laptop. However, we developed our own VMM for mobile devices and current desktop virtualization vendors such as VMware, Citrix, and Microsoft have not targeted their solutions to TEE.

This paper will be organized as follows. Firstly, we give basic information about the virtualization technology on mobile devices, mVMM, which is basic architecture to provide the isolated execution environment, and the mobile payment service that is an important application to verify the proposed concept. The detailed architecture for our solution is introduced in Section 3 and one of the possible applications (i.e., mobile payment) is explained in the enhanced security point of view in Section 4. Section 5 briefly analyzes the strengths and limitations of the proposed architecture. Finally, we conclude this paper and suggest some future work.

II. BACKGROUD

A. *Mobile Virtual Machin Monitor (mVMM)*

Virtual Machine Monitor (VMM), also called by hypervisor, is small system software that runs a number of virtual machines on a single physical host system. This concept first appeared on mainframe computers in 1970s, and becomes widely used on desktop computers and in server farms. For consumers, this technology makes it possible to consolidate multiple servers on single server and run Windows, Mac OS, Linux applications simultaneously on a desktop PC.

As modern mobile devices clearly have sufficient resources to support virtualization, we can leverage it for improving mobile device security, resulted by isolation feature of virtualization. As illustrated in Fig. 1, we can consider two types of VMM on mobile device. One is type-1 VMM, known as native or bare-metal VMM, which runs directly on the host's hardware to control all host hardware and to manage multiple guest OSes such as Android. A guest OS thus runs on another level above VMM. The other is type-2 VMM, also called by the hosted VMM, which runs within a conventional OS environment such as Android. With the VMM layer as a distinct second software level, the guest OSs run at the third level above the hardware [23].

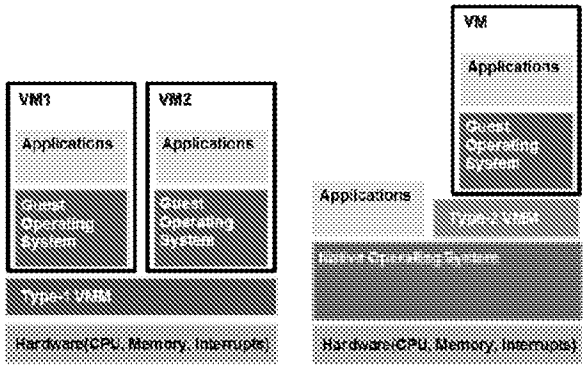


Figure 1. Two types of VMM for mobile devices.

Although type-1 VMM is more efficient and secure than type-2 VMM, it may be very difficult to provide the type-1 VMM to the commercial market without any help of mobile device manufactures, because mobile systems such as smartphone and tablets have not been standardized like desktop systems. From the security point of view, however, we need to focus on the TEE based on type-1 VMM, but the proposed concept can be also applicable to type-2 VMM, although it may be less secure.

B. *Mobile Payment and Vulunerabilites*

Mobile payment or m-payment can be considered as payment services which are operated under financial regulation and performed from or via mobile devices. A consumer can use a mobile device to pay for services and goods, instead of paying with cash, check, or credit cards. Therefore, it can be defined as the process of two parties such as a customer and a merchant, or a customer and a bank, exchanging financial value using a mobile device in return for goods or services [7, 8]. This ecosystem involves a number of participants, such as banks, network operators, device manufactures, service providers, application providers, merchants, trusted service managers, and so on. Therefore, there have been numerous m-payment systems and approaches suggested by the different concepts and technology researches [9, 10, 11, 12, 13, 14].

Although there can be a number of participants involved in m-payment service, however, we assume a simple m-payment model involving banks, a merchant, a customer as shown in Fig. 2, because our objective is just to verify the security and reliability of the proposed architecture on a customer's mobile device through this service, not to verify the security of the whole m-payment architecture.

In our simple model, the customer's bank loads the m-payment applications on a customer's mobile device and the customer processes mutual authentication with the merchant and the customer's bank to purchase goods or services and complete payment, using the mobile device. For authentication key establishment, there are many schemes

IBM - Ex. 1002-Part 10 of 10, Page 3389
IPR2025-00379 (IBM v. Croga)

that provide authentication of session keys. Authenticated key-establishment protocols serve as basic building blocks for constructing secure, complex, and higher-level cryptographic protocols. Some paper suggested compilers to convert a key-establishment protocol secure in a specific security model into a key-establishment protocol secure in a more restricted security model [15]. As an authentication scheme, we assume Public Key Infrastructures (PKIs) which enables users to securely communication on an insecure public network by encrypting and decrypting plaintext, and reliably verifies the identity of a user via digital signature. It can bind public keys with the respective user identities by means of a certificate authority (CA).

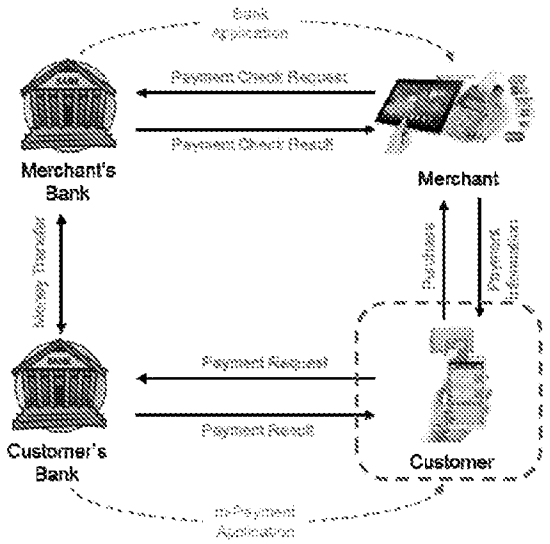


Figure 2. Simple mobile payment scenario.

From the customer's device point of view, if there is no support of the secure execution, three vulnerabilities can be identified as follows:

- **Keeping security sensitive data in non-secure area**: Private keys, credential certificates, and other security sensitive data require to be strongly secured, and should not be stolen by any malwares. If they are kept in non-secure area, however, it is not safe anymore.
- **Processing encryption and decryption in non-secure area**: Just before encryption or just after decryption, secure data from the customer, the merchant, or the bank can be converted to the plaintext in non-secure area.
- **User input/output in non-secure area**: To ensure m-payment security, we also need to make sure that input data from a user such as password or nonce are executed in the secure area. And, the payment details and the transaction details should not be displayed in non-secure area as well.

Therefore, the security sensitive functions and data need to be isolated in the secure execution environment in order to avoid these potential security vulnerabilities. In the following section, we introduce the architecture to avoid the identified vulnerabilities.

III. VIRTUALIZATION-BASED TRUSTED EXECUTION ENVIRONMENT

The proposed architecture is based on the mVMM, allowing two virtual machines to be multiplexed on a single mobile device. Therefore, it can enable simultaneous execution of two different OSes, offering a rich set of security features and execution environments. And, the proposed mVMM can be considered to be trusted, because there is no administrator domain to be able to threat the other guest domains.

In this section, we present the isolated execution and the trusted path on the trusted mVMM with the detailed explanation of its structure and the secure communication mechanism between two OSes with the functionalities provided by the trusted mVMM

A. Trusted mVMM

As outlined in the block architecture below Fig. 3, the proposed architecture identifies two distinct execution environments: General Execution Environment (GEE) and Trusted Execution Environment (TEE). The trusted mVMM can provide two separated execution environments that run simultaneously, allowing each execution environment to run in its own hardware protection domain and providing the strong isolation between two different execution environments. It provides only virtual hardware resources such virtual CPU, virtual memory and virtual IO to each virtual machine, so one virtual machine cannot access the resources of the other virtual machine and the trusted VMM, and communicate with the other virtual machine without the help of the trusted VMM. Within the GEE, a traditional OS runs, providing a hugely extensive operating environment, which brings flexibility and capability, but at the same time leaves a mobile device vulnerable to a wide range of security threads. On the other hand, any application in the TEE is trustworthy in authenticity and integrity and also any asset is protected in confidentiality.

In this architecture, the secure OS in the TEE only needs to present the relatively simple file systems and the several trusted device drivers dedicated to the TEE, without any path directly to external networks, while the host OS in the GEE must support the file systems, the network stacks, the public device drivers, and so on. In addition to the isolated execution capability that is provided by the trusted mVMM, two additional capabilities are provided to enhance security features as follows:

Trusted Human Interface Device (HID) Path: HID resources on a mobile device, such as a touchscreen, a graphic display, and a keypad, should be dedicated only to the secure OS, when the secure OS runs on the foreground.

IBM - Ex. 1002-Part 10 of 10, Page 3390
IPR2025-00379 (IBM v. Croga)

For the events from a touchscreen and a keypad, the trusted mVMM firstly receives and forwards them to the secure OS on the foreground. In the case of a graphic display, the trusted mVMM switches the memory area for a graphic resource to the area that the secure OS manages. The management of HID resources in the trusted mVMM is also related to efficiency capability, i.e. response time.

Trusted Network Communication Path: The secure OS has no external network interfaces, thereby preventing snooping or tampering by malicious programs through the Internet. In the trusted mVMM, a trusted communication path allows the trusted applications in the secure OS to make use of external networks in safety. The following section presents this capability in detail.

B. *Secure Communication between Host OS and Secure OS*

We separated the trusted environment from the untrusted part, using the trusted mVMM. This requires some form of interaction or collaboration between two separated domains. To facilitate this, the trusted mVMM provides the communication method, message passing over shared memory and event channel.

1) Shared Memory: This feature is ability to enable the secure OS and the host OS to communicate the large amounts of data quickly, efficiently, and securely via access to a memory area accessbile to both OSes. As outlined in Fig.4, after the host OS starts to boot, it allocates a memory area to be shared with the secure OS and requests the setting-up of the shared memory to the trusted mVMM. The trusted mVMM maps the physical memory which is provided by the host OS with the virtual memory of the secure OS. After this memory mapping by the trusted mVMM, the secure OS can directly access to the shared memory which is physically allocated in the host OS. In this way, the host OS cannot have access to the physical memory area of the secure OS.

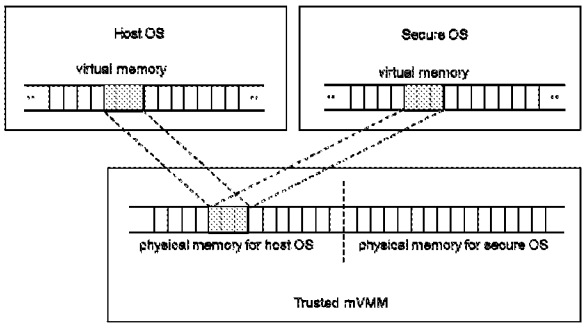


Figure 4. Memory mapping for the shared memory in the trusted mVMM.

2) Event Channel: The trusted mVMM provides the event channels up to 32 (1 byte: 1 bit per a channel) that have been pre-defined. From now on, we have defined a total of four events of the touch screen, the keypad, and send/receive events of message passing. Each OS identifies the event channel using the event number. As illustrating by a simple example, when the secure OS starts to send an event to the host OS, it sends an interrupt to the trusted mVMM by specifying the event number. The host OS can receive the event from the trusted mVMM only after it registers an event handler to the corresponded event channel. If the event handler is not registered, events will not be sent. Regardless of the running state of each OS, the event handler is executed asynchronously, but it is possible only in a state that the OS can receive interrupts.

3) Communication Agents: The kernel API in the secure OS offers trusted applications a set of trusted communication APIs. It is a low level communication interface designed to enable trusted applications running inside the TEE to exchange data with external service providers. This interface calls the communication server agent running in the secure OS kernel and exchaning data with the communication client agent via message passing. The communication client agent sends or receives data through the public network interface. The host OS kernel provides no interface with which user applications in the GEE can access to the communication client agent.

IV. Enhanced Security Model in M-Payment Service

In this section, we present the enhanced security model based the proposed architecture on a mobile device, referring to the m-payment application that we introduced in section 2.

A. *Participants*

In this model, a customer is a human being who is assumed to use a mobile device to pay for services and goods. A mobile device has two execution environments:

- When it is in the GEE, the host OS is scheduled to run. Although it provides the full functionalities, it is untrusted and prone to be snooped or tampered by malwares
- In order to carry out the security sensitive applications, the secure OS in the TEE is scheduled to run, where user input/output and encryption/decryption of data can be executed in security.

The customer, employing his own mobile device, is required by the merchant and the bank to prove his credential. The credential contains cryptographic keys and further attributes, signed by a certificate authority (CA), which enables the customer to construct the expected proof. This procedure is also applied to the merchant, the customer's bank, and the merchant's bank.

For PKI authentication mechanism, the participants of the customer, the merchant, and the banks keep their own private keys and distribute the corresponded public keys to the other parts. In Table 1, all the participants with brief explanation are given.

IBM - Ex. 1002-Part 10 of 10, Page 3391
IPR2025-00379 (IBM v. Croga)

TABLE I. PARTICIPANTS

Customer	Mobile device user
Secure OS	Operating system in the TEE on a mobile device
Host OS	Operating system in the GEE on a mobile device
Merchant	Merchant terminal to provide the information of goods and services and issue payment
Customer's bank	Issuing bank which have the relationship with customers
Merchant's bank	(implicitly) Issuing bank which have the relationship with merchants
CA	(implicitly) Certificate authority issuing credentials

B. *m-Payment Service Flow*

From a customer's point of view, m-payment is carried out with two-factor authentication, payment approval, and transaction approval, collaborating with the merchant and the customer's bank. We only focus on the security in the customer's mobile device, assuming that the bank server and the merchant's device are trusted enough.

A payment procedure based on the proposed architecture in the m-payment application is shown in Fig. 5. In the customer's point of view, it works as follows:

- *Step 1:* The customer, the merchant, and the customer's bank keep their own private key. After the customer decide to purchase the services or the goods, he starts to process payment, inputting and verifying the certificate password on the secure OS.
- *Step 2:* After the credentials are exchanged to verify each partner by CA, the merchant sends the approval request such as the customer's consent and all payment details *pd*, which is encrypted using the customer's public key. The encrypted data are communicated to the host OS through the Internet and to the secure OS through the communication channel between two OSes.
- *Step 3:* The encrypted data from the merchant are decrypted in the secure OS, the customers gets the payment information such as the merchant's bank name, the merchant's account number, and amount of money. When the customer approves the payment, the credentials of the customer and the customer's bank are exchanged to verify each other by CA.
- *Step 4:* After the customer prepares a money transfer, encrypts payment details *pd*, and sends it to the host OS through the communication channel and to the bank through the Internet, the bank sends the encrypted request of an authorized approval, that is, the customer's consent and all transaction details td. The encrypted data are communicated to the host OS through the Internet and to the secure OS through the communication channel.
- *Step 5:* After the encrypted data from the bank are decrypted in the secure OS, the customer gets the relevant information about the transfer on the mobile device, such as bank names, account numbers and amount of money. To approve the transaction, the customer enters the authentication password and the nonce, which is encrypted in the secure OS and communicated to the host OS through the communication channel and to the bank through the Internet.
- *Step 6:* The customer's bank processes the money transfer with the merchant's bank and it sends the payment result to the customer. The secure OS encrypts the proof of payment details and send to the merchant through the communication channel and the Internet. After the merchant verifies the payment completion with the merchant's bank, all payment process is completed.

Payment details *pd* and transaction details *td* are highly important in this application. They show the identities of the merchant and the bank and the descriptions of the payment and the transaction. They should be used to display information to the customer and to receive the customer's input. In the proposed architecture, all these security sensitive data and processing are kept and executed on the secure OS in the TEE, while all unauthorized applications on the host OS can see only the encrypted data and cannot intercept the customer's inputs and displayed information.

The proposed architecture is highly secure, yet simple and intuitive for the user without an additional hardware device. In the same way, other applications of mobile banking services, streaming services, eGovernment applications, and etc. can be devised.

V. SECURITY ANAYSIS FOR PROPOSED ARCHITECTURE

For desired security features on mobile devices, the proposed architecture, which is the virtualization-based trusted execution environment, gives the trusted application in the secure OS the ability of isolated execution from other code. It provides secrecy and integrity of that application's code and data at run-time. Because the isolated execution environment is not under control of the host OS, any malicious program outside the secure OS is not directly accessible to that code and data, even though the host OS itself is compromised.

The proposed mVMM also supports two kinds of trusted path (i.e., trusted HID path and trusted network communication path), which is one of the desired security features. The trusted HID path prevents the malicious programs from confusing the user about which the secure OS is in use, while the trusted network communication path guarantees the secure usage of the external networks. We achieved this by forwarding the HID events from the trusted VMM directly to the secure OS and supporting the secure communication channel between the host OS and the secure OS. They protect secrecy and availability of communication between the trusted application and a peripheral (e.g., touchscreen, graphic display, or network). This trustworthy property allows a human user to ascertain secrecy of the application with which she is interacting.

IBM - Ex. 1002-Part 10 of 10, Page 3392
IPR2025-00379 (IBM v. Croga)

Although device manufacturers and platform providers should support them, we need to consider two kinds of other security features to make the proposed architecture more trustworthy:

- **Secure storage**: It provides secrecy, integrity for the trusted application's data *at rest* (when the mobile device is powered off or the trusted application goes to a sleep mode). These sensitive data should be stored in the secure storage and should not be leaked to an attacker. Most mobile OSes (such as Android, iOS, and etc.) provide this property at least using file system permissions. Although it may not be enough, we adopted an encrypted file system in Android which can be enabled to protect data in the secure OS. In addition to the simple encrypted file system, however, the implementation of a device key and necessary cryptographic mechanisms should be provided with the support of device manufacturers. Such a device key may be initialized during manufacturing and stored in a protected memory area on the processor chip.
- **Integrity-checking on boot time**: The boot time refers to the time frame from the reset/power-up of the mobile device hardware to the time when the trusted VMM, the secure OS, and the host OS have completed their initialization and loading. The integrity of at least the trusted mVMM code and the secure OS code should be verified during the boot time to prevent or detect their usages that have been modified without authorization. In the boot sequence, after boot starts from ROM and first initializations are done, the trusted VMM will be validated and be executed. And then, the secure OS will be validated and be executed, containing initialization, key import, and so on. However, it also requires the support of hardware manufacturers.

In this paper, we only get the software perspectives, such as the isolated execution and the trusted path, only with hardware virtualization extensions support in ARM architecture. Considering the run-time environment, the proposed architecture can provide a reasonably trustworthy execution environment to the user.

VI. Conclusions and Future Work

We presented a flexible architecture for the virtualization-based trusted execution environment on mobile devices. The proposed architecture allows the trusted applications to run securely in the trusted execution environment (TEE). The key primitive of this architecture is the trusted mobile virtual machine monitor (trusted mVMM). The trusted mVMM enables users to partition the mobile device platform into the multiple and isolated execution environments. The applications in the secure OS can keep security sensitive data, process security sensitive operations, and handle security sensitive user input/output in the securely isolated environment. We also presented the primitives provided by the trusted mVMM to build the trusted paths such as trusted HID path and trusted network communication path, in particular those required to support trustworthy execution of applications.

We implemented these primitives in the prototype implementation and built the simple service application, m-payment service, which demonstrates its security capabilities. From the run-time execution point of view, we believe the proposed architecture is sufficiently secure, yet simple and intuitive for the user, and also can have benefit to be extended to other applications such as mobile banking services, streaming services, eGovernment applications, and so on.

For future work, it would be interesting to incorporate a near field communication (NFC) module and Secure Element (SE) that is one proprietary solution for the secure storage. And, we will extend our implementation to support platform integrity during system boot or at device run-time, if possible. As an interesting research topic, we are developing "Virtualization of Smart Cards" concept which is a holistic approach and centralized management for a secure (enterprise) single-sign-on (SSO) system. In this concept, the authentication procedure will be executed in the secure OS completely isolated with the host OS by virtualization layer, and a trusted virtual machine for smart card in the cloud will be instantiated at the connected mobile device through secure communication channels only when the user has been authenticated. We are targeting at e-Commerce services, telemedicine services, and public/private services.

Acknowledgment

This work was supported by the ICT R&D program of MSIP/IITP [10041753, Development of General-Purpose OS and Virtualization Technology to Reduce 30% of Energy for High-density Servers based on Low-power Processors].

References

[1] Vasudevan, Amit, et al. *Trustworthy Execution on Mobile Devices: What security properties can my mobile platform give me?*. Springer Berlin Heidelberg, 2012.

[2] Marforio, Claudio, et al. "Secure enrollment and practical migration for mobile trusted execution environments." *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*. ACM, 2013.

[3] Gebhardt, Carl, Chris I. Dalton, and Allan Tomlinson. "Separating hypervisor trusted computing base supported by hardware." *Proceedings of the fifth ACM workshop on scalable trusted computing*. ACM, 2010.

[4] Stumpf, Frederic, et al. "An approach to a trustworthy system architecture using virtualization." *Autonomic and trusted computing*. Springer Berlin Heidelberg, 2007. 191-202.

[5] McCune, Jonathan M., et al. "TrustVisor: Efficient TCB reduction and attestation." *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010.

[6] Jeon, Woongryul, et al. "A practical analysis of smartphone security." *Human Interface and the Management of Information. Interacting with Information*. Springer Berlin Heidelberg, 2011. 311-320.

[7] Ramezani, Elham. "Mobile Payment." 2008

[8] Schierz, Paul Gerhardt, Oliver Schilke, and Bernd W. Wirtz. "Understanding consumer acceptance of mobile payment services: An

IBM - Ex. 1002-Part 10 of 10, Page 3393
IPR2025-00379 (IBM v. Croga)

empirical analysis."*Electronic Commerce Research and Applications* 9.3 (2010): 209-216.

[9] Chen, Lei-da. "A model of consumer acceptance of mobile payment."*International Journal of Mobile Communications* 6.1 (2008): 32-52.

[10] Harb, Hany, Hassan Farahat, and Mohamed Ezz. "SecureSMSPay: secure SMS mobile payment model." *Anti-counterfeiting, Security and Identification, 2008. ASID 2008. 2nd International Conference on*. IEEE, 2008.

[11] Kungpisdan, Supakorn, Bala Srinivasan, and Phu Dung Le. "A secure account-based mobile payment protocol." *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*. Vol. 1. IEEE, 2004.

[12] Liu, Jun, Jianxin Liao, and Xiaomin Zhu. "A system model and protocol for mobile payment." *e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on*. IEEE, 2005.

[13] Karnouskos, Stamatis. "Mobile payment: a journey through existing procedures and standardization initiatives." *Communications Surveys & Tutorials, IEEE* 6.4 (2004): 44-66.

[14] Tiwari, Ayu, et al. "A multi-factor security protocol for wireless payment-secure web authentication using mobile devices." *arXiv preprint arXiv:1111.3010*(2011).

[15] Jeong Ok Kwon and Ik Rae Jeong, "Relations among Security Models for Authenticated Key Exchange," ETRI Journal, vol. 36, no. 5, Oct. 2014, pp. 856-864, http://dx.doi.org/10.4218/etrij.14.0113.1071.

[16] ARM Limited. ARM build security foundation for future wireless and consumer devices. ARM Press Release, May 2003.

[17] ARM Limited. ARM security technology: Building a secure system using TrustZone technology. WhitePaper PRD29-GENC-009492C, April 2009.

[18] ARM Limited. Virtualization extensions architecture specification, October 2010, *http://infocenter.arm.com*

[19] Xen.org. Xen ARM project, wiki.xen.org/wiki/XenARM

[20] Heiser, Gernot, and Ben Leslie. "The OKL4 Microvisor: Convergence point of microkernels and hypervisors." *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*. ACM, 2010.

[21] Dall, Christoffer, and Jason Nieh. "KVM/ARM: the design and implementation of the linux ARM hypervisor." *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014.

[22] http://systems.cs.columbia.edu/projects/kvm-arm

[23] Gerald J. Popek and Robert P. Goldberg, "Formal requirements for virtualizable third generation architecture", In Proc. of the Fourth Symposium on Operating System Principles, YorkTown Heights, New York, October 1973.

[24] Global Platorm™. GlobalPlatform device technology TEE system architecture v1.0, November 2011, *http://www.globalplatform.org*

[25] http://www.samsung.com/global/business/mobile/platform/mobile-platform/knox/

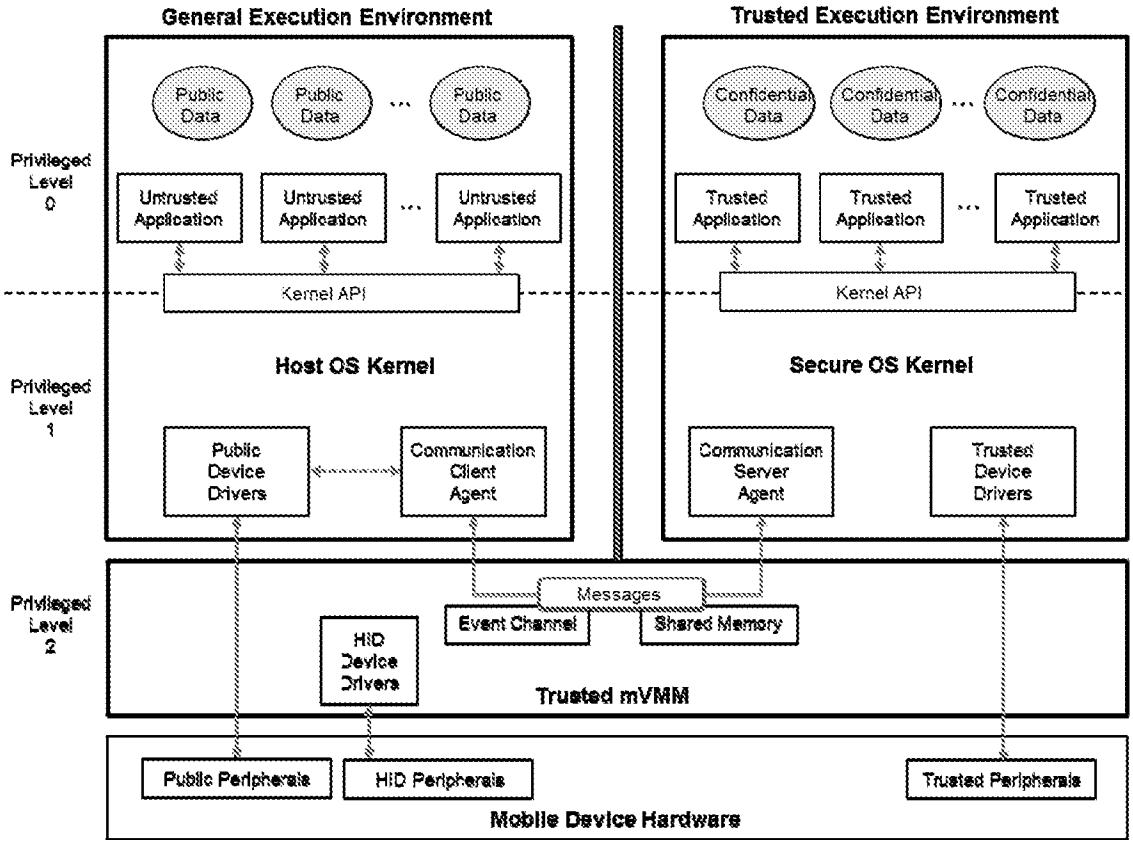


Figure 3. Security-enhanced m-payment service flow by the proposed architecture.

IBM - Ex. 1002-Part 10 of 10, Page 3394
IPR2025-00379 (IBM v. Croga)

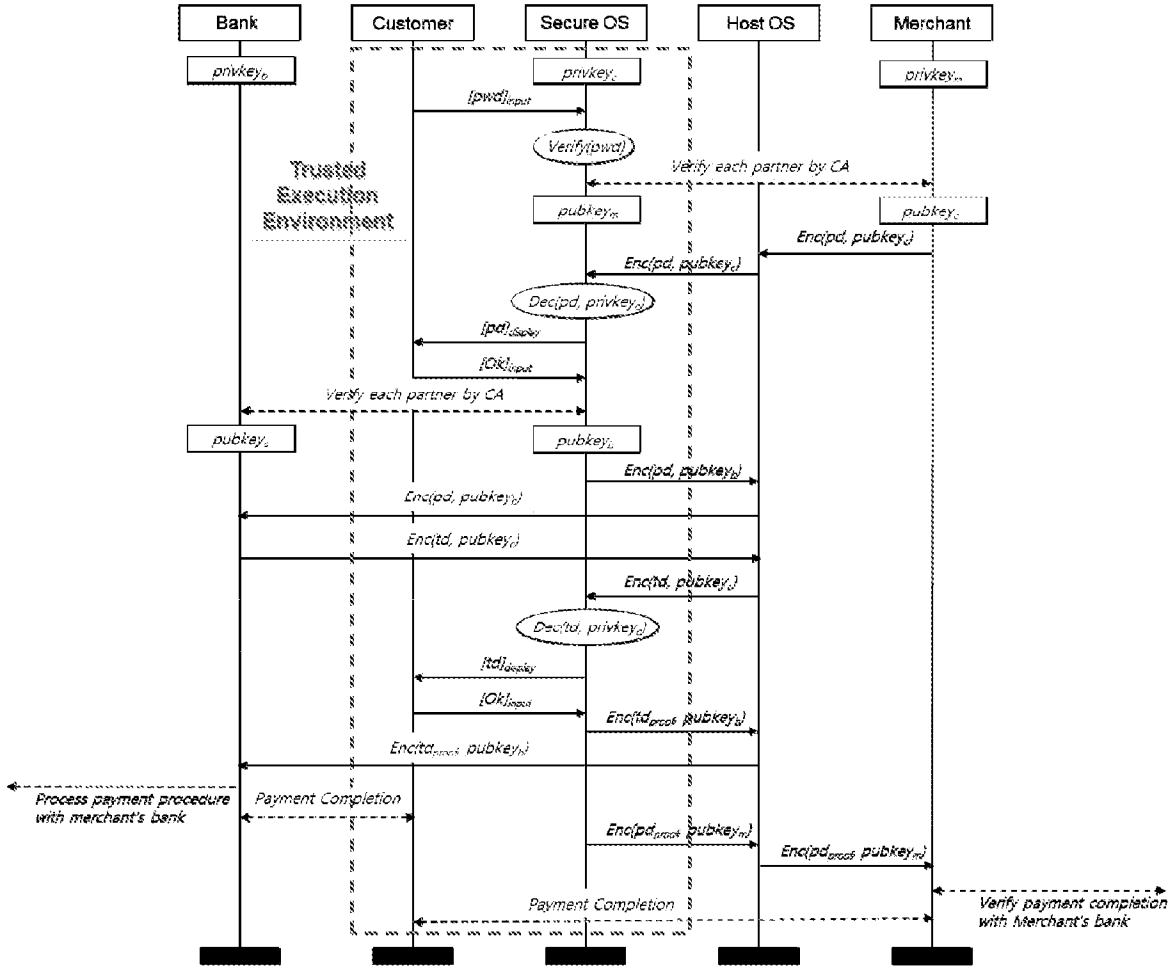
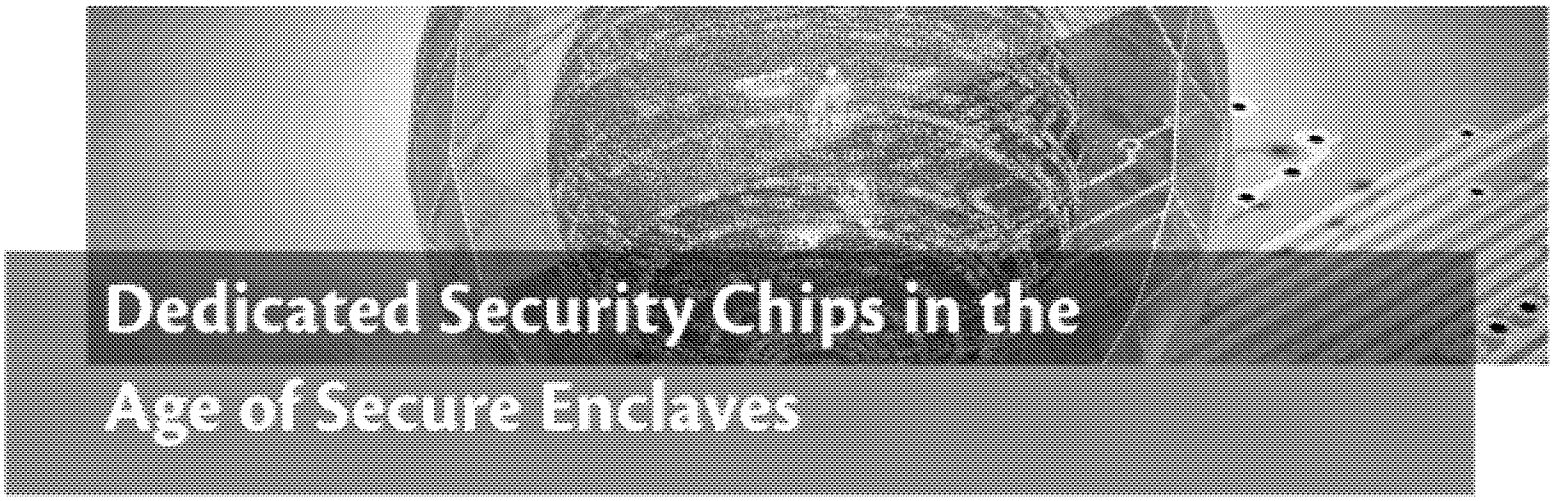


Figure 5. Security-enhanced m-payment service flow by the proposed architecture.

IBM - Ex. 1002-Part 10 of 10, Page 3395
IPR2025-00379 (IBM v. Croga)



Dedicated Security Chips in the Age of Secure Enclaves

Kari Kostiainen, Aritra Dhar, and Srdjan Capkun | ETH Zurich

Secure enclave architectures have become prevalent in modern CPUs. Enclaves provide a flexible way to implement various hardware-assisted security services. But special-purpose security chips can still have advantages. Interestingly, dedicated security chips can also assist enclaves and improve their security.

Trusted computing base (TCB) minimization is one of the most fundamental computer security principles. The main idea is to reduce the amount of software and hardware that needs to be trusted for the secure operation of a particular application. A common technique to achieve TCB minimization is to run the application inside a trusted execution environment (TEE). The TEE protects the application's execution despite any other compromised software on the same system.

One TEE implementation approach that has recently gained significant popularity is to realize the TEE by enhancing the main CPU of the computing platform with new features, such as special instructions and access-control checks. Intel's software guard extension (SGX), designed for the x86 architecture, is a prime example of such a TEE. In SGX, the CPU ensures that no other process can access the memory of the protected application, which is called an *enclave*. By doing this, SGX guarantees that enclaves enjoy execution integrity and that their data remains confidential. Several other TEE designs exist, too. ARM TrustZone is a popular TEE architecture that is used in many commercial mobile devices, while Sanctum[1] serves as a good example of a research TEE system. For simplicity, we focus on Intel's SGX and use it as a case study to discuss the strengths and limitations of enclaves.

SGX-style enclaves are a powerful security primitive. They are programmable, and thus developers can implement almost arbitrary hardware-protected security services by using them. This is in contrast to previous secure elements, such as trusted platform modules (TPMs), that support only a fixed set of operations. Enclaves are also fast, as they run on the main CPU of the computing platform, compared to significantly slower security elements, including smart cards. Furthermore, enclaves are cheap since they require no additional hardware, in contrast to expensive separate coprocessors, such as hardware security modules. This combination of programmability, high performance, and low cost makes enclaves an attractive way to deploy various hardware-assisted security services. Indeed, after a decade of research and development into secure enclaves, the first large-scale commercial deployments are now starting. For example, Microsoft's Azure Confidential Computing service uses SGX enclaves to protect customer data in the cloud.

The wide adoption of enclave architectures in modern CPUs is probably the most prominent trend in

Digital Object Identifier 10.1109/MSEC.2020.2990230
Date of current version: 15 May 2020

 Copublished by the IEEE Computer and Reliability Societies 1540-7993/20©2020IEEE

IBM - Ex. 1002-Part 10 of 10, Page 3396
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

hardware-assisted security during the past decade. However, there is also another, more subtle trend appearing. Recently, computing service providers, such as Google, and computer manufacturers, including Apple, have started to enhance their systems with special-purpose security chips. Google's cloud servers have a security chip called *Titan* in them,[2] while Apple's computers come with the T2 security chip.[3]

At first glance, these two trends seem almost contradictory. If enclaves enable arbitrary hardware-protected security services, why do we still need dedicated security chips? In this article, we discuss the rationale behind this situation. We explain the benefits of dedicated security chips and outline two of our research projects where we designed such chips. These projects showcase an interesting new pattern, one where special-purpose security chips assist enclaves and thus improve their security.

Dedicated Security Chips

Computing platform providers have recently added new security chips to their systems. We look at two examples.

Google Titan

Titan[2] is a security chip implemented as a low-power microcontroller on Google's purpose-built server platforms. The Titan chip communicates with the main CPU via the Serial Peripheral Interface, and it interposes between the boot firmware flash and Platform Controller Hub (PCH).

One of the main functionalities that Titan implements is secure boot. When the server machine is powered up, Titan executes code, known as *boot read-only memory* (*bootROM*), from its embedded ROM. This code is immutable and thus implicitly trusted. The boot ROM code loads Titan's firmware from the embedded flash and verifies its integrity using a digital signature. Once Titan's firmware is securely verified and running, it can authenticate the boot process of its host. Titan blocks the PCH's access to the firmware flash until it has cryptographically confirmed the content of the flash, and then it releases the lock and allows the verified boot firmware to configure the machine and activate the boot loader, which subsequently verifies and loads the operating system (OS). Such an iterative process enables precise control over which system software is booted.

Apple T2

Apple's latest PCs come with a security chip called *T2*[3] that also supports secure boot. When a machine with the T2 chip is turned on, T2 executes code from its ROM. This code verifies the next step of T2's own boot process. Once T2 is fully running, it can verify the Unified Extensible Firmware Interface, which will ensure that only an authorized kernel will be booted on the host CPU. Besides secure boot, T2 provides other security features, such as protecting users' fingerprint values and making sure that the microphone is disconnected from the main CPU when the laptop's lid is closed.

Specific Security Objectives

Both Titan and T2 implement secure boot. Secure boot is also a good example of a security mechanism that is outside the security objectives of SGX, which was designed to provide a specific set of protections,[4] including detecting an integrity violation of an enclave instance, safeguarding the confidentiality of an enclave's data, isolating enclaves from each other, and ensuring that an enclave's execution always starts from an authorized location. The overall goal of these protections can be loosely summarized as enabling secure computation on untrusted computing platforms. Because these safeguards do not include OS integrity verification, platform providers have added dedicated security chips, including Titan and T2, to implement such functionality. Disconnecting the microphone from the main CPU is another example of a security feature that is not

Secure Boot in TrustZone

ARM TrustZone is a processor-based trusted execution environment architecture that is commonly used on smartphones. The main idea of TrustZone is to implement two separate execution modes on the main CPU. All untrusted software, such as the operating system (OS) and third-party apps, is executed in the *normal world*, while applications that need protection are run in a separate execution mode called the *secure world*. The processor and memory controllers ensure that any process in the normal world cannot access the secure world.

TrustZone can enable secure boot.[5] A mobile device can be configured such that when the device is powered up, the main CPU starts executing implicitly trusted code that is loaded from the ROM in the secure world. This code can then verify the normal-world boot loader before the CPU starts executing the main boot sequence of the normal-world OS. Many smartphone manufacturers implement this approach.

IBM - Ex. 1002-Part 10 of 10, Page 3397
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

provided by enclaves. (Other TEEs, such as the ARM TrustZone architecture, can accommodate secure boot. See "Secure Boot in TrustZone.")

Security Weaknesses

Besides limited objectives, enclaves have security weaknesses. Since enclaves and untrusted code share the same CPU, they can be susceptible to side-channel leakage and microarchitectural attacks. The recently discovered Spectre and Meltdown vulnerabilities showed how transient execution could leak information across isolation boundaries. The same idea was successfully applied to extract secret keys from SGX enclaves in the Foreshadow attack.[6] While specific attacks can be, and have been, mitigated (e.g., Intel's microcode updates include Spectre and Meltdown patches), side-channel and microarchitectural attacks continue to be a concern for enclaves. The root cause is that modern processors are extremely complex systems that have been optimized through decades. Enclave support was added on top of many layers of performance optimizations, and now, in hindsight, one can easily say that this approach was not the ideal foundation for strong isolation. In this regard, dedicated security chips have a clear advantage over enclaves.

Another security challenge is the rich interface between the untrusted OS and the enclave. Enclaves must interact with the OS in many ways. For example, enclaves communicate by sending and receiving messages through the OS. Enclaves also need to safely pause their execution for interrupts. While enclave architectures provide a coarse-grained memory isolation primitive at the hardware level, developers need to ensure that the interface is protected on the software level. Common implementation tasks include sanitization of buffers and safety checks for pointers. Because such checks are tedious, several enclave runtimes, such as the Open Enclave software development kit, have been designed to assist enclave creators. However, recent research has shown that many such enclave runtimes have classical memory-safety vulnerabilities.[7] Since dedicated security chips do not need equally extensive interaction with the OS, the interface toward the untrusted OS is easier to protect tightly.

Other Security Services?

To summarize our discussion so far, enclaves do not implement all useful security mechanisms, and they also have significant security issues. Dedicated security chips can address both of these concerns. Obviously, the security services that can be implemented as dedicated security chips are not limited to the previously discussed examples. Which other services could be implemented as special-purpose security chips? Titan and T2 are integrated security chips that are permanently attached to the computing platform. Are there also use cases that would benefit from plug-and-play security tokens? In the rest of this article, we explore these questions by examining two examples from our recent research. Our first example is trusted path,[8] and after that, we focus on platform identification and remote attestation.[9]

Trusted Path

The main goal of SGX is to enable secure computation for enclaves. Such enclaves do not easily lend themselves to secure user interaction. The main reason for this is that, in an architecture such as SGX, enclaves communicate with input and output (IO) devices through the untrusted OS. When an enclave needs to receive user input, the OS must pass data from an input device, such as a keyboard, to the enclave. When an enclave creates user output, the OS must forward data to an output device, such as the display. Such a TEE design means that a compromised OS can easily modify any user inputs and outputs. Indeed, trusted path—a secure channel from the human user to a trusted application, such as the enclave—is outside the security objectives of SGX.[4]

User input manipulation can have severe consequences. If a malicious OS modifies the user input that is provided to a financial enclave, the enclave can be tricked to perform unauthorized payments. If enclaves are used to implement hardened medical devices and industrial systems, user input modifications may cause serious safety and health risks. Also, any enclave that needs user passwords or similar credentials is difficult to implement securely.

Design Principles

Several projects have explored the idea of complementing computing platforms with dedicated hardware modules for building a trusted path. While proposing extra hardware may be easy, the more difficult and interesting question is what exactly should the added hardware do? To answer this question, we look at previous approaches, examine their limitations, and identify design principles for trusted path implementation.

The first approach that we look at is transaction confirmation.[10] In such a solution, the user first completes the interaction, such as a payment, by interacting with the user interface (UI) of the untrusted platform that may be manipulated by the OS or browser. Once this is done, the user is expected to confirm his or her input, such as a payment value and account number, through a separate hardware token to detect and prevent any possible modifications. This approach has two main problems. The first is the fact that the user now has to interact with two separate devices and two UIs, which reduces usability. The second and more severe problem is that the extra confirmation step is vulnerable to user habituation. *Habituation* refers to behavior where the

IBM - Ex. 1002-Part 10 of 10, Page 3398
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

user begins, after a few successful transactions, confirming new payments without verifying their correctness. These observations lead us to the first design principle.

Principle 1. Out-of-context security confirmations have a high cognitive load and risk of user habituation. Thus, user interaction should be protected in the context of the normal UI.

The next approach that we examine is input signing.[11] Here, the main idea is to use a simple hardware device that sits between a user input device, such as a keyboard, and the untrusted computing platform. The device intercepts every keypress, or similar user input event, and sends a signed trace of each one to the trusted application. Input signing conforms to our first principle because the user does not have to interact outside the main UI. However, such solutions are vulnerable to attacks, where the adversary manipulates the user's input by showing false information on the output channel. That is, output manipulation leads to user input integrity violations. One example is a fake-typo attack. Assume that the user types in the value "10," but the adversary shows "1" on the screen. The user is likely to think he or she mistyped and may press "0" again. As a result, a signed trace of "100" will be sent to the trusted application. This simple attack leads us to our next design principle.

Principle 2. User input and output integrity cannot be considered in isolation. Both must be protected simultaneously.

The last approach that we examine is secure overlays. Fidelius[12] is an example research system that follows this approach. In Fidelius, one device intercepts keyboard presses and signs them for the trusted application, while another device intercepts the High-Definition Multimedia Interface (HDMI) output signal and modifies it with secure overlays of security-critical UIs, such as payment web forms. Fidelius addresses the previously mentioned problems but is still vulnerable to a different type of user input manipulation that we call *early submission attack*. This attack is possible because Fidelius protects input from a keyboard but not a mouse. While this may seem to be only a functional limitation, it turns out to be a security problem. Assume again that the user intends to submit the value "10." Once the user has typed "1," the untrusted OS or browser generates a fake mouse event that submits the web form. Mouse inputs could be disabled altogether, but such a naive solution hurts usability. Now we can state our last design principle.

Principle 3. All user input modalities must to be protected simultaneously.

ProtectIOn System

Given these principles, we designed a new trusted-path system called *ProtectIOn*.[8] In the following, we focus on the use case where the trusted path is established between the user and a remote web server. That is, we want to protect interactions where the user completes and submits a security-critical web form. The same solution could be easily modified to create a trusted path between the user and a local enclave, as well.

Figure 1 provides an overview of the ProtectIOn system. The central component of the solution is a low-complexity embedded device called *IOHub* that intercepts keypresses from a keyboard and movement events from a mouse, tracks mouse movements, and draws secure overlays. When the user visits a webpage that contains a protected web form, the remote server sends a QR code to the untrusted browser, as shown in Figure 2(a). The QR code contains a specification of the protected web form signed by the server to prevent its modification in the browser. By using QR codes, we enable communication from the web server to the IOHub device via an unmodified browser, which significantly simplifies deployment. By periodically examining the HDMI signal, IOHub detects the QR code on the screen, decodes it, and verifies its signature. After that, as illustrated in Figure 2(b), IOHub renders the protected web form as an overlay on top of the HDMI frames that it receives from the untrusted OS. This step ensures that the security-critical UI elements are correctly presented to the user, and thus the output integrity of the protected web form is preserved.

IOHub tracks mouse movement events, and when the mouse pointer enters the secure overlay, as pictured in Figure 2(c), IOHub dims the rest of the screen to focus the user's attention on the secure overlay. Such protection is needed to prevent the user from following a possible fake mouse cursor, drawn by the untrusted browser or OS, elsewhere on the screen. Dimming parts of the screen has been shown to be an effective way to focus the user's attention to the correct cursor.[13]

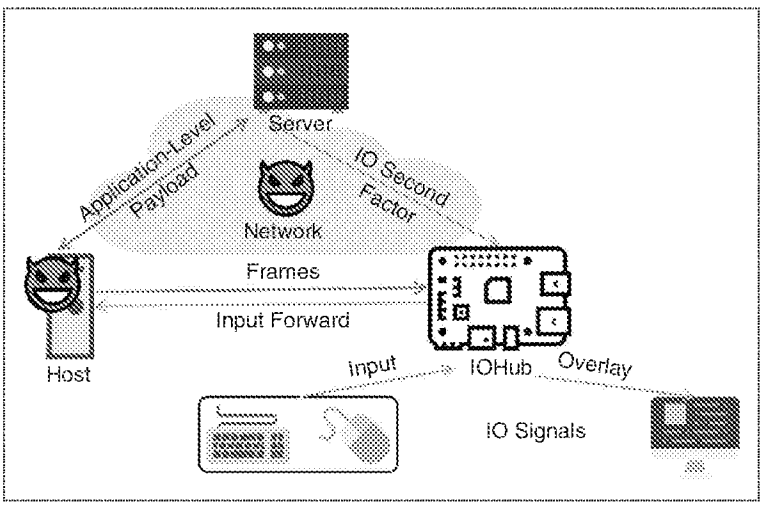


Figure 1. An overview of the ProtectIOn system and its attacker model.

IBM - Ex. 1002-Part 10 of 10, Page 3399
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

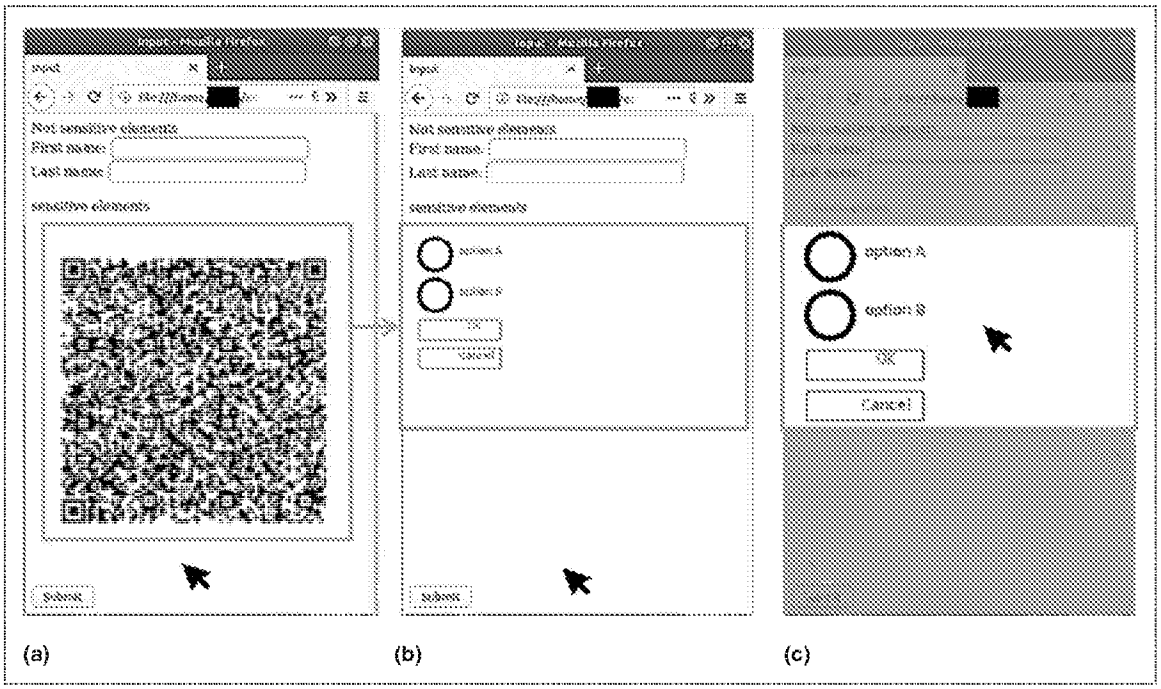


Figure 2. The ProtectIOn UI, including (a) the attacker's view, (b) the user's view on the monitor, and (c) focusing the user's attention.

While the user interacts with the protected web form, IOHub intercepts all input events, and when the user clicks on the "OK" button, IOHub signs all inputs and sends them to the server via the untrusted browser (e.g., by encoding them to a requested URL). Since the "OK" button is part of the protected overlay and all mouse clicks are intercepted and signed by the IOHub device, early submission attacks are not possible. The server verifies the signed user inputs it receives. If input confidentiality is needed, the user can be required to trigger a secure attention sequence, such as Control-Alt-Delete, before entering any secrets. The untrusted OS and the browser cannot observe sensitive data on the secure overlay since the information is rendered by IOHub and never accessible to them. Such a design complies with our three principles. Principle 1 is met because the user interacts only with the main UI. Secure overlays support Principle 2, and mouse tracking combined with input trace signing ensures that all input modalities are protected (Principle 3).

ProtectIOn Prototype

We implemented a prototype of the IOHub device as a combination of Raspberry Pi and Arduino boards and a simple HDMI interceptor. Our prototype shows that the proposed functionalities are feasible to implement with a small TCB on low-cost hardware. Full details of the ProtectIOn system are explained in our recent paper.[8]

Attestation and Platform Identification

Remote attestation is a key feature of enclave architectures such as SGX. In remote attestation, an external verifier checks that an enclave was constructed as expected. To do this, the verifier sends a challenge to the attested CPU. The CPU signs the challenge together with a previously recorded measurement of the enclave's code by using a processor-specific attestation key. The signature can also include application-specific data, such as the public key of the enclave, that enables secure communication with the attested enclave. The attestation key is part of a group signature scheme managed by Intel. The signed attestation statement is then sent back to the verifier. If the signature can be authenticated correctly, the remote verifier knows that the attested enclave was correctly constructed and runs the expected code inside a legitimate SGX processor. The remote attestation process enables remote verifiers to detect enclave integrity violations before provisioning secrets to, or accepting signed messages from, the enclaves.

Relay Attacks

The previously outlined remote attestation protocol is a useful security mechanism, but it also has a well-known problem. An adversary that controls the OS on the attested platform can easily redirect the attestation challenge to another platform that computes the response. The verifier cannot notice such relay attacks since the SGX attestation mechanism is based on a group

IBM - Ex. 1002-Part 10 of 10, Page 3400
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

signature scheme, and all processors from the same group produce indistinguishable signatures. (Even if attestation used traditional digital signatures, it would be very difficult, in practice, for the verifier to know which signing key corresponds to which CPU.) In other words, SGX guarantees detection of an enclave's integrity violation, but identification of the computing platform on which the attested enclave is running is not part of its security objectives.

Relay attacks have been known for a long time. Parno identified them in the context of TPM attestation more than a decade ago and called them *cuckoo attacks*.[14] However, the full implications of relay attacks have not been well understood. Since attestation can be redirected only to another legitimate processor that executes exactly the same attested enclave code, it may appear that such relay attacks do not have noteworthy negative implications. Our analysis shows that such a belief is misguided.

Relay attack implications. Many computer systems, such as servers at data centers, use multiple layers of protection. These safeguards may include using TEEs to protect certain applications and other defenses, such as running software components at different privilege levels, physically protecting access to the computing platform, and frequent patching of discovered vulnerabilities. The main implication of relay attacks is that they increase the adversary's ability to attack the attested enclave by circumventing many such protections.

Our first observation is that by redirecting the attestation to the adversary's platform, the enemy enables physical side-channel attacks, such as acoustic, electric, and electromagnetic monitoring, that otherwise would not be possible to mount on the victim's platform. Such side channels have been shown to be effective and inexpensive means to extract secrets,[15] and hardening enclaves against all possible physical side channels is difficult. Relay attacks can also enable privilege escalation. In cases where the adversary has compromised only the user space application that manages the enclave on the victim platform, the application can redirect the attestation to the attacker's platform, where the opponent controls the OS, as well. In such cases, the relay enables digital side-channel attacks that require system privileges.

The third and perhaps most subtle implication of relay is that it can enable software-based side-channel attacks that otherwise would not be possible due to the timing of certain events. One example is a scenario where the victim platform OS is compromised at the time of attestation and secret provisioning, and the attested enclave is hardened against known digital side-channel attacks (e.g., using tools like Raccoon[16]). After secret provisioning, the OS compromise is detected, and the platform is cleaned and patched. Later, a new side-channel attack vector (that is not prevented by the used tools) is discovered. If the adversary performed redirection during attestation and the secret was provisioned to the attacker's machine, the new side channel is exploitable. Without the relay, the attack is not possible. In this case, the attestation relay eliminates the security benefit of good platform maintenance. Finally, we note that attacks based on leaked attestation keys (e.g., ones obtained through the Foreshadow attack[6]) are independent of relaying. If the adversary has obtained a valid attestation key, he or she can emulate an SGX processor on the target platform and steal any secrets that are provisioned to it.

Trust on first use. A commonly suggested solution to relay attacks is the principle of trust on first use (TOFU). In one example solution, a platform-specific enclave generates a key pair and exports the public part of the key for certification right after a fresh OS installation. When remote verifiers need to attest to other enclaves on that platform, they can first authenticate the certified enclave, which, in turn, performs local attestation of the target enclave (SGX provides a local attestation primitive that cannot be relayed). Such a TOFU solution has several problems. The first is a large temporary TCB, as a complete general-purpose OS needs to be trusted during the first boot. The second is difficulty in deployment because a fresh OS reinstallation is not always possible. The third is the need for online certification authorities, which increases their attack surface, assuming that the certification process is automated. Further problems are discussed in Dhar et al.[9]

ProximiTEE System

Parno identified relay attacks more than a decade ago and, at the same time, suggested proximity verification as a solution.[14] The main idea was to verify that the attested CPU was in the proximity of the verifier device, which should prevent attestation redirection to remote platforms. Proximity verification using an external device overcomes the previously listed main problems of TOFU approaches. First, the TCB is small, as the verifier device can be a single-purpose one and thus very simple. Second, the adoption of such a solution is easy since there is no need to fully reset the target platform. And third, the authority that certifies the verifier device can remain entirely offline.

Although the idea of proximity verification sounds simple to realize, the research efforts that followed failed to implement secure proximity verification. The main reason for the failure was that TPM, the secure element commonly available at the time, supported only fixed identification operations, such as digital

IBM - Ex. 1002-Part 10 of 10, Page 3401
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

signatures. TPM signatures may take up to one second, which enables the redirected attestation request to travel a long distance, and thus TPMs were simply too slow for secure proximity verification.

Because SGX enclaves are programmable, it is possible to implement proximity verification protocols that leverage simple operations, such as exclusive-OR, that enable fast challenge-response rounds. Based on this observation, we designed a hardened SGX attestation scheme called *ProximiTEE*.[9] Our solution uses a simple embedded device called *ProximiKey* that is attached to the target platform through a local communication interface, such as a USB. In hardened remote attestation, the verifier first establishes a secure channel, a ProximiKey, whose public key the verifier learns from its issuer. ProximiKey performs standard remote attestation on the local enclave, establishes a secure Transport Layer Security channel to the enclave, and verifies its proximity by using a simple distance-bounding protocol that consists of repeated and fast challenge-response rounds. If such proximity verification succeeds, ProximiKey facilitates the creation of a secure channel between the remote verifier and attested enclave.

Proximity verification security. While the preceding design is mostly straightforward, the more interesting aspect is whether such a system prevents relay attacks, in practice. To answer this question, we implemented our solution using a USB prototyping board and simulated a strong attack scenario where the adversary performs a relay to another SGX platform that is connected to the target platform through a 1-m-long Ethernet wire. We also assumed that the adversary was able to perform all protocol computation instantaneously. Figure 3 shows the results from our experiments, where we measured both the legitimate and relayed challenge-response latencies. The vast majority of the benign latencies range from 145 to 250 μs, while the attack round trips take from 200 to 750 μs. The average delay of our adversary is only 80 μs. (To put this in perspective, even the highly optimized network connections between major data centers in the same region exhibit latencies from 1 ms upward.)

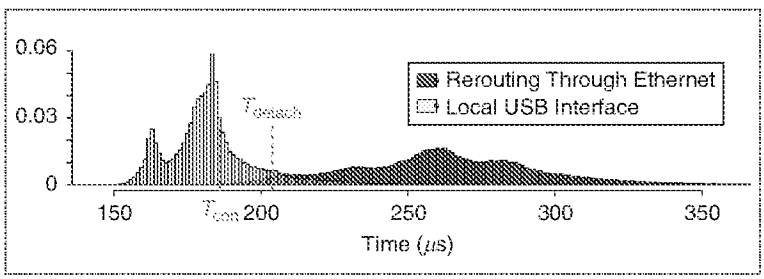


Figure 3. The ProximiKey latency distributions for legitimate challenge-response rounds and a simulated relay attack.

As can be seen from Figure 3, these two latency distributions are distinguishable. Our analysis confirms that it is possible to set protocol parameters (the number of challenge-response rounds, latency threshold, and so on) such that very fast relay attacks can be detected with high probability, and legitimate attestations fail only with a negligible likelihood. The full details of the ProximiTEE system and our analysis can be found in our recent paper.[9]

Discussion

Now that we have seen two commercial integrated security chips (Titan and T2) and two user-attachable research tokens (ProtectIOn and ProximiTEE), we can discuss their deployment aspects, security benefits, and previous comparable solutions.

Deployment Options

Integrated security chips, such as Titan and T2 are, obviously, limited to deployments by major service and platform providers that have the ability to design and build their own systems. ProximiTEE is an example of a plug-and-play security token that can be attached to the target platform through a standard interface, such as a USB. Deployment of security solutions is a feasible option for a larger set of service providers. For example, cloud-computing providers can enhance off-the-shelf servers with ProximiKey tokens and communicate their public keys to their clients to support more secure attestation. Another interesting use case is the setup of a permissioned blockchain, where every consensus node is hardened with SGX enclaves. The trusted authority that appoints the consensus nodes can issue a ProximiKey token to each organization that operates one node.

Also, ProtectIOn could be deployed as a plug-and-play security module. Such deployment enables service providers, such as voting authorities and banks, to increase the security of their services without restricting users' choice of client platform. In medical and industrial domains, an externally attached ProtectIOn module can improve the security of safety-critical systems, even when modifications to the computing platform itself are prohibited due to strict regulations. Alternatively, ProtectIOn could be deployed as an integrated security chip such that its functionality is implemented as part of the integrated keyboard, mouse, and display controllers.

Security Benefits

Titan and T2 are chips that enable security functionality, such as a secure boot, that is not provided by enclaves.

IBM - Ex. 1002-Part 10 of 10, Page 3402
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

Thus, the operation of Titan and T2 is largely orthogonal to the operation of enclaves. In comparison, ProximiTEE is a security chip that is designed to work in collaboration with enclaves and improve their security guarantees by enabling secure TEE identification for hardened remote attestation. ProtectIOn is a solution that can either assist enclaves or operate independently of them. One possible usage for ProtectIOn is to enable a trusted path from the user to a local enclave, which can communicate securely with remote servers. In such a deployment, ProtectIOn works together with an enclave. Alternatively, ProtectIOn could be employed to create a trusted path from the user to a remote server without the use of enclaves. Such a deployment is beneficial when the risk of microarchitectural attacks on enclaves is considered too high, for example.

Dedicated Chips and Early TEEs

Some previous TEE designs have leveraged dedicated security chips to implement a TEE that can be attested. AMD's Secure Virtual Machine technology is one such example, where the TEE is created as follows: the CPU measure code in a specific memory region enables direct memory access protections for that region, disables interrupts, records the measurement into a platform configuration register of a TPM chip, and, finally, begins executing the measured code. Essentially, this sequence of events provides a clean execution of the measured code without restarting the whole platform, and therefore this technique is often called *late launch*. The main role of the dedicated chip (a TPM, in this case) is to securely record the code that was launched so that an external verifier can check its integrity by using attestation.

Future computing platforms are likely to combine various units, including CPUs, GPUs, tensor processing units, field-programmable gate arrays (FPGAs), and more. Similar to the current enclave architectures that enhance CPUs with secure execution capabilities, other processing units, such as GPUs and FPGAs, will need secure computation, too. There are already several ongoing research efforts that explore the design of such TEEs.[17]

Our research on trusted path (the ProtectIOn system[8]) highlights that IO devices need secure communication with enclaves. Similarly, other peripherals, such as GPS units and fingerprint sensors, would benefit from secure communication with enclaves. Protected communication between TEEs and other platform components requires authentication, enclave identification, and access-control mechanisms. The ARM TrustZone architecture has limited support toward this direction. In TrustZone, hardware components, such as memory controllers, can make coarse-grained access-control decisions based on the CPU's execution mode.[5] Extending this paradigm for more fine-grained access control and secure intercomponent communication is one promising direction. We envision future computing platforms where enclaves, peripherals, and special-purpose security chips can communicate and work together to provide a rich set of hardware-assisted platform security services. ⊠

References

1. V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 857–874. doi: 10.5555/3241094.3241161.
2. "Titan in depth: Security in plaintext," Google, Mountain View, CA. Accessed on: Mar. 2020. [Online]. Available: https://cloud.google.com/blog/products/gcp/titan-in-depth-security-in-plaintext
3. "About the Apple T2 security chip," Apple Inc., Palo Alto, CA. Accessed on: Mar. 2020. [Online]. Available: https://support.apple.com/en-us/HT208862
4. F. McKeen et al., "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware and Architectural Support Security and Privacy*, 2013. doi: 10.1145/2487726.2488368.
5. J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security Privacy*, vol. 12, no. 4, pp. 29–37, July–Aug. 2014. doi: 10.1109/MSP.2014.38.
6. J. Van Bulck et al., "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proc. 27th USENIX Conf. Security Symp.*, 2018, pp. 991–1008. doi: 10.5555/3277203.3277277.
7. J. Van Bulck et al., "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *Proc. 2019 ACM SIGSAC Conf. Computer and Communications Security*, pp. 1741–1758. doi: 10.5555/3277203.3277277.
8. A. Dhar, E. Ulqinaku, K. Kostiainen, and S. Capkun, "ProtectIOn: Root-of-trust for IO in compromised platforms," in *Proc. Network and Distributed System Security Symp.*, 2020. doi: 10.14722/ndss.2020.24112.
9. A. Dhar, I. Puddu, K. Kostiainen, and S. Čapkun, "ProximiTEE: Hardened SGX attestation by proximity verification," in *Proc. 10th ACM Conf. Data and Application Security and Privacy*, 2020, pp. 5–16. doi: 10.1145/3374664.3375726.
10. A. Filyanov, J. M. McCuney, A. Sadeghiz, and M. Winandy, "Uni-directional trusted path: Transaction confirmation on just one device," in *Proc. 2011 IEEE/IFIP 41st Int. Conf. Dependable Systems & Networks*, pp. 1–12. doi: 10.1109/DSN.2011.5958202.
11. A. Dhar, D-Y. Yu, K. Kostiainen, and S. Capkun, "Integrikey: End-to-end integrity protection of user

IBM - Ex. 1002-Part 10 of 10, Page 3403
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

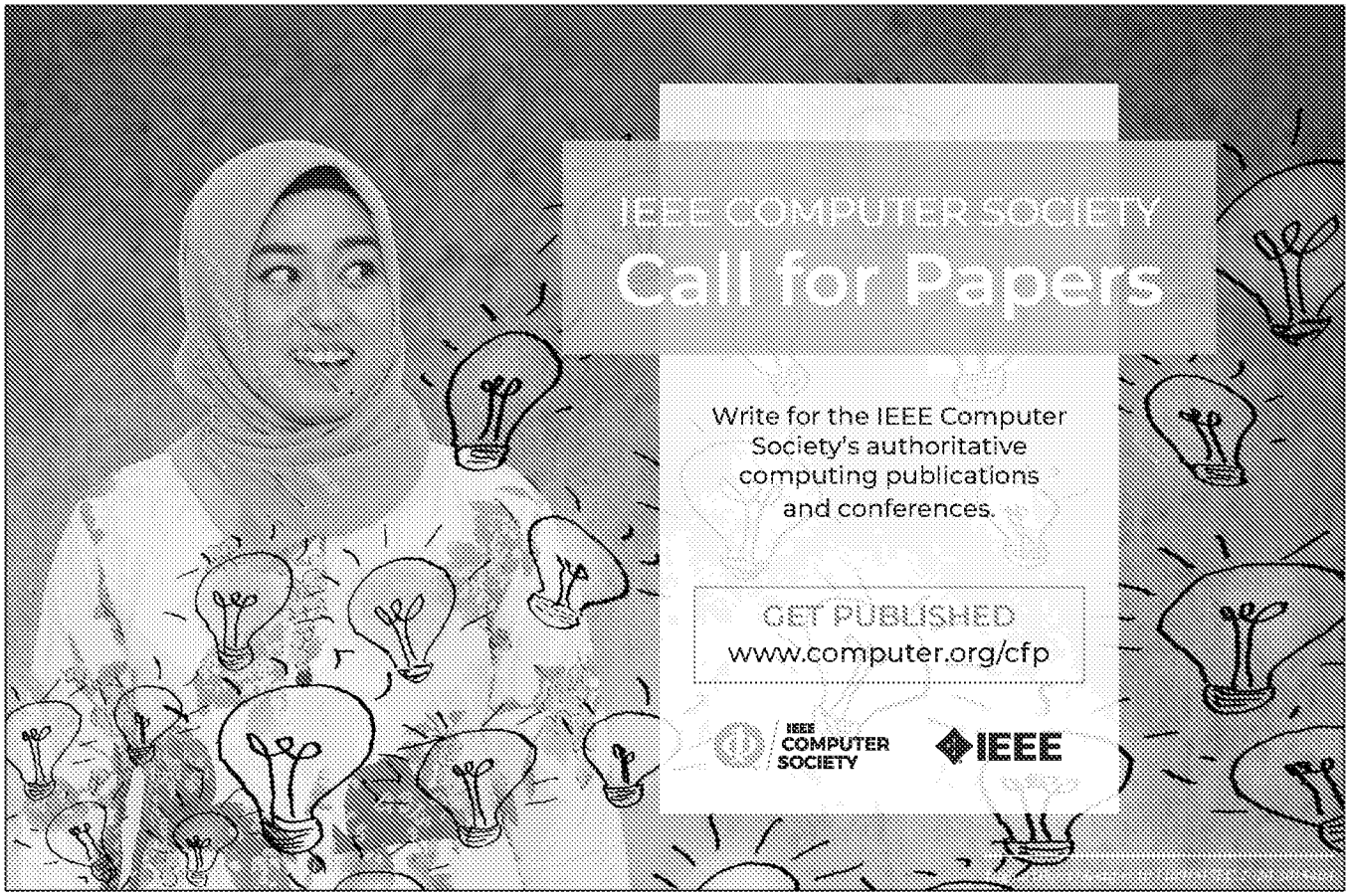
input," 2017. [Online]. Available: https://eprint.iacr.org/2017/1245

12. S. Eskandarian et al., "Fidelius: Protecting user secrets from compromised browsers," in *Proc. IEEE Symp. Security and Privacy*, 2019, pp. 264–280. doi: 10.1109/SP.2019.00036.
13. L.-S. Huang, A. Moshchuk, H. Wang, S. Schechter, and C. Jackson, "Clickjacking: Attacks and defenses," in *Proc. 21st USENIX Conf. Security Symp.*, 2012, pp. 413–428. doi: 10.5555/2362793.2362815.
14. B. Parno, "Bootstrapping trust in a trusted platform," in *Proc. 3rd Conf. Hot Topics Security*, 2008, pp. 1–6. doi: 10.5555/1496671.1496680.
15. D. Genkin, L. Pachmanov, I. Pipman, A. Shamir, and E. Tromer, "Physical key extraction attacks on PCs," *Commun. ACM*, vol. 59, no. 6, pp. 70–79, 2016. doi: 10.1145/2851486.
16. A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *Proc. 24th USENIX Conf. Security Symp.*, pp. 431–446. doi: 10.5555/2831143.2831171.
17. S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *Proc. 12th USENIX Conf. Operating Systems Design and Implementation*, 2018, pp. 681–696. doi: 10.5555/3291168.3291219.

Kari Kostiainen is a senior scientist at ETH Zurich. His research interests include trusted computing, mobile security, secure user interaction, and blockchain security. Kostiainen received a Ph.D. from Aalto University, Espoo, Finland. Contact him at kari.kostiainen@inf.ethz.ch.

Aritra Dhar is a Ph.D. student at ETH Zurich. His research interests include secure user interaction, trusted computing, anonymous networks, and program analysis. Dhar received an M.S. from the Indraprastha Institute of Information Technology, Delhi, India. Contact him at aritra.dhar@inf.ethz.ch.

Srdjan Capkun is a full professor at ETH Zurich. His research interests include wireless security, secure positioning, trusted computing, and blockchain technologies. Capkun received a Ph.D. from École Polytechnique Fédérale de Lausanne, Switzerland. He is a Fellow of ACM. Contact him at srdjan.capkun@inf.ethz.ch.



IBM - Ex. 1002-Part 10 of 10, Page 3404
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on April 14,2021 at 14:25:24 UTC from IEEE Xplore. Restrictions apply.

HAL
archives-ouvertes.fr

Computing the Real Isolated Points of an Algebraic Hypersurface

Huu Phuoc Le, Mohab Safey El Din, Timo de Wolff

To cite this version:

Huu Phuoc Le, Mohab Safey El Din, Timo de Wolff. Computing the Real Isolated Points of an Algebraic Hypersurface. ISSAC'20 - International Symposium on Symbolic and Algebraic Computation, Jul 2020, Kalamata / Virtual, Greece. pp.297-304, 10.1145/3373207.3404049 . hal-02920059

HAL Id: hal-02920059
https://hal.archives-ouvertes.fr/hal-02920059

Submitted on 24 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

COMPUTING THE REAL ISOLATED POINTS OF AN ALGEBRAIC HYPERSURFACE

Huu Phuoc Le
Sorbonne Université, CNRS,
Laboratoire d'Informatique de Paris 6, LIP6,
Équipe POLSYS
F-75252, Paris Cedex 05, France
huu-phuoc.le@lip6.fr

Mohab Safey El Din
Sorbonne Université, CNRS,
Laboratoire d'Informatique de Paris 6, LIP6,
Équipe POLSYS
F-75252, Paris Cedex 05, France
mohab.safey@lip6.fr

Timo de Wolff
Technische Universität Braunschweig,
Institut für Analysis und Algebra, AG Algebra
Universitätsplats 2
38106 Braunschweig, Germany
t.de-wolff@tu-braunschweig.de

August 24, 2020

ABSTRACT

Let $\boldsymbol{R}$ be the field of real numbers. We consider the problem of computing the real isolated points of a real algebraic set in $\boldsymbol{R}^n$ given as the vanishing set of a polynomial system. This problem plays an important role for studying rigidity properties of mechanism in material designs. In this paper, we design an algorithm which solves this problem. It is based on the computations of critical points as well as roadmaps for answering connectivity queries in real algebraic sets. This leads to a probabilistic algorithm of complexity $(nd)^{O(n\log(n))}$ for computing the real isolated points of real algebraic hypersurfaces of degree d. It allows us to solve in practice instances which are out of reach of the state-of-the-art.

Keywords Semi-algebraic sets · Critical point method · Real algebraic geometry · Auxetics · Rigidity

Mohab Safey El Din and Huu Phuoc Le are supported by the ANR grants ANR-18-CE33-0011 SESAME, and ANR-19-CE40-0018 DE RERUM NATURA, the joint ANR-FWF ANR-19-CE48-0015 ECARP project, the PGMO grant CAMISADO and the European Union's Horizon 2020 research and innovative training network programme under the Marie Skłodowska-Curie grant agreement N° 813211 (POEMA). Timo de Wolff is supported by the DFG grant WO 2206/1-1.

1 Introduction

Let $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{C}$ be respectively the fields of rational, real and complex numbers. For $\boldsymbol{x} \in \boldsymbol{R}^n$ and $r \in \boldsymbol{R}$, we denote by $B(\boldsymbol{x}, r) \subset \boldsymbol{R}^n$ the open ball centered at $\boldsymbol{x}$ of radius r.

Let $f \in \boldsymbol{Q}[x_1, \ldots, x_n]$ and $\mathcal{H} \subset \boldsymbol{C}^n$ be the hypersurface defined by $f = 0$. We aim at computing the *isolated points* of $\mathcal{H} \cap \boldsymbol{R}^n$, i.e. the set of points $\boldsymbol{x} \in \mathcal{H} \cap \boldsymbol{R}^n$ s.t. for some positive r, $B(\boldsymbol{x}, r) \cap \mathcal{H} = \{\boldsymbol{x}\}$. We shall denote this set of isolated real points by $\mathscr{Z}(\mathcal{H})$.

Motivation We consider here a particular instance of the more general problem of computing the isolated points of a *semi-algebraic* set. Such problems arise naturally and frequently in the design of rigid mechanism in material design. Those are modeled canonically with semi-algebraic constraints, and isolated points to the semi-algebraic set under consideration are related to mobility/rigidity properties of the mechanism. A particular example is the study of *auxetic* materials, i.e., materials that shrink in all directions under compression. These materials appear in nature (first discovered in [20]) e.g., in foams, bones or propylene; see

e.g. [32], and have various potential applications. They are an active field of research, not only on the practical side, e.g., [16, 11], but also with respect to mathematical foundations; see e.g. [5, 6]. On the constructive side, these materials are closely related to *tensegrity frameworks*, e.g., [21, 8], which can possess various sorts of rigidity properties.

Hence, we aim to provide a practical algorithm for computing these real isolated points in the particular case of real traces of complex hypersurfaces first. This simplification allows us to significantly improve the state-of-the-art complexity for this problem and to establish a new algorithmic framework for such computations.

State-of-the-art As far as we know, there is no established algorithm dedicated to the problem under consideration here. However, effective real algebraic geometry provides subroutines from which such a computation could be done. Let $\mathcal{H}$ be a hypersurface defined by $f = 0$ with $f \in \boldsymbol{Q}[x_1, \ldots, x_n]$ of degree d.

A first approach would be to compute a cylindrical algebraic decomposition adapted to $\mathcal{H} \cap \boldsymbol{R}^n$ [7]. It partitions $\mathcal{H} \cap \boldsymbol{R}^d$ into connected *cells*, i.e. subsets which are homeomorphic to $]0, 1[^i$ for some $1 \leq i \leq n$. Next, one needs to identify cells which correspond to isolated points using adjacency information (see e.g. [1]). Such a procedure is at least doubly exponential in n and polynomial in d.

A better alternative is to encode real isolated points with quantified formula over the reals. Using e.g. [2, Algorithm 14.21], one can compute isolated points of $\mathcal{H} \cap \boldsymbol{R}^n$ in time $d^{O(n^2)}$. Note also that [31] allows to compute isolated points in time $d^{O(n^3)}$.

A third alternative (suggested by the reviewers) is to use [2, Algorithm 12.16] to compute sample points in each connected component of $\mathcal{H} \cap \boldsymbol{R}^n$ and then decide whether spheres, centered at these points, of infinitesimal radius, meet $\mathcal{H} \cap \boldsymbol{R}^n$. Note that these points are encoded with parametrizations of degree $d^{O(n)}$ (their coordinates are evaluations of polynomials at the roots of a univariate polynomial with infinitesimal coefficients). Applying [2, Alg. 12.16] on this last real root decision problem would lead to a complexity $d^{O(n^2)}$ since the input polynomials would have degree $d^{O(n)}$. Another approach would be to run [2, Alg. 12.16] modulo the algebraic extension used to define the sample points. That would lead to a complexity $d^{O(n)}$ but this research direction requires modifications of [2, Alg. 12.16] since it assumes the input coefficients to lie in an *integral domain*, which is not satisfied in our case. Besides, we report on practical experiments showing that using [2, Alg. 12.16] to compute *only* sample points in $\mathcal{H} \cap \boldsymbol{R}^n$ does not allow us to solve instances of moderate size.

The topological nature of our problem is related to connectedness. Computing isolated points of $\mathcal{H} \cap \boldsymbol{R}^n$ is equivalent to computing those connected components of $\mathcal{H} \cap \boldsymbol{R}^n$ which are reduced to a single point (see Lemma 1). Hence, one considers computing *roadmaps*: these are algebraic curves contained in $\mathcal{H}$ which have a non-empty and connected intersection with all connected components of the real set under study. Once such a roadmap is computed, it suffices to compute the isolated points of a semi-algebraic curve in $\boldsymbol{R}^n$. This latter step is not trivial; as many of the algorithms computing roadmaps output either curve segments (see e.g., [4]) or algebraic curves (see e.g., [28]). Such curves are encoded through *rational parametrizations*, i.e., as the Zariski closure of the projection of the $(x_1, \ldots, x_n)$-space of the solution set to

$$w(t, s) = 0, x_i = v_i(t, s)/\tfrac{\partial w}{\partial t}(t, s), \quad 1 \leq i \leq n$$

where $w \in \boldsymbol{Q}[t, s]$ is square-free and monic in t and s and the v_i's lie in $\boldsymbol{Q}[t, s]$ (see e.g., [28]). As far as we know, there is no published algorithm for computing isolated points from such an encoding.

Computing roadmaps started with Canny's (probabilistic) algorithm running in time $d^{O(n^2)}$ on real algebraic sets. Later on, [27] introduced new types of connectivity results enabling more freedom in the design of roadmap algorithms. This led to [27, 4] for computing roadmaps in time $(nd)^{O(n^{1.5})}$. More recently, [3], still using these new types of connectivity results, provide a roadmap algorithm running in time $d^{O(n \log^2 n)} n^{O(n \log^3 n)}$ for general real algebraic sets (at the cost of introducing a number of infinitesimals). This is improved in [28], for smooth bounded real algebraic sets, with a probabilistic algorithm running in time $O((nd)^{12n \log_2 n})$. These results makes plausible to obtain a full algorithm running in time $(nd)^{O(n \log n)}$ to compute the isolated points of $\mathcal{H} \cap \boldsymbol{R}^n$.

Main result We provide a probabilistic algorithm which takes as input f and computes the set of real isolated points $\mathscr{Z}(\mathcal{H})$ of $\mathcal{H} \cap \boldsymbol{R}^n$. A few remarks on the output data-structure are in order. Any finite algebraic set $Z \subset \boldsymbol{C}^n$ defined over $\boldsymbol{Q}$ can be represented as the projection on the $(x_1, \ldots, x_n)$-space of the solution set to

$$w(t) = 0, x_i = v_i(t), \quad 1 \leq i \leq n$$

where $w \in \boldsymbol{Q}[t]$ is square-free and the v_i's lie in $\boldsymbol{Q}[t]$. The sequence of polynomials $(w, v_1, \ldots, v_n)$ is called a *zero-dimensional parametrization*; such a representation goes back to Kronecker [19]. Such representations (and their variants with denominators) are widely used in computer algebra (see e.g. [12, 13, 14]). For a zero-dimensional parametrization $\mathfrak{Q}$, $Z(\mathfrak{Q}) \subset \boldsymbol{C}^n$ denotes the finite set represented by $\mathfrak{Q}$. Observe that considering additionally isolating boxes, one can encode $Z(\mathfrak{Q}) \cap \boldsymbol{R}^n$. Our main result is as follows.

Theorem 1. *Let $f \in \boldsymbol{Q}[x_1, \ldots, x_n]$ of degree d and $\mathcal{H} \subset \boldsymbol{C}^n$ be the algebraic set defined by $f = 0$. There exists a probabilistic algorithm which, on an input f of degree d, computes a zero-dimensional parametrization $\mathfrak{P}$ and isolating boxes which encode $\mathscr{Z}(\mathcal{H})$ using $(nd)^{O(n \log(n))}$ arithmetic operations in $\boldsymbol{Q}$.*

In Section 5, we report on practical experiments showing that it already allows us to solve non-trivial problems which are actually out of reach of [2, Alg. 12.16] to compute sample points in $\mathcal{H} \cap \boldsymbol{R}^n$ only. We sketch now the geometric ingredients which allow

IBM - Ex. 1002-Part 10 of 10, Page 3407
IPR2025-00379 (IBM v. Croga)

us to obtain such an algorithm. Assume that f is non-negative over $\boldsymbol{R}^n$ (if this is not the case, just replace it by its square) and let $\boldsymbol{x} \in \mathscr{Z}(\mathcal{H})$. Since $\boldsymbol{x}$ is isolated and f is non-negative over $\boldsymbol{R}^n$, the intuition is that for $e > 0$ and small enough, the real solution set to $f = e$ looks like a ball around $\boldsymbol{x}$, hence a bounded and closed connected component $C_{\boldsymbol{x}}$. Then the restriction of every projection on the x_i-axis to the algebraic set $\mathcal{H}_e \subset \boldsymbol{C}^n$ defined by $f = e$ intersects $C_{\boldsymbol{x}}$. When e tends to 0, these critical points in $C_{\boldsymbol{x}}$ "tend to $\boldsymbol{x}$". This first process allows us to compute a superset of candidate points in $\mathcal{H} \cap \boldsymbol{R}^n$ containing $\mathscr{Z}(\mathcal{H})$. Of course, one would like that this superset is finite and this will be the case up to some generic linear change of coordinates, using e.g. [25].

All in all, at this stage we have "candidate points" that may lie in $\mathscr{Z}(\mathcal{H})$. Writing a quantified formula to decide if there exists a ball around these points which does not meet $\mathcal{H} \cap \boldsymbol{R}^n$ raises complexity issues (those points are encoded by zero-dimensional parametrizations of degree $d^{O(n)}$, given as input to a decision procedure).

Hence we need new ingredients. Note that our "candidate points" lie on "curves of critical points" which are obtained by letting e vary in the polynomial systems defining the aforementioned critical points. Assume now that $\mathcal{H} \cap \boldsymbol{R}^n$ is bounded, hence contained in a ball B. Then, for e' small enough, the real algebraic set defined by $f = 0$ is "approximated" by the union of the connected components of the real set defined by $f = e'$ which are contained in B. Besides, these "curves of critical points", that we just mentioned, hit these connected components when one fixes e'. We actually prove that two distinct points of our set of "candidate points" are connected through these "curves of critical points" and those connected components defined by $f = e'$ in B if and only if they do not lie in $\mathscr{Z}(\mathcal{H})$. Hence, we use computations of roadmaps of the real set defined by $f = e'$ to answer those connectivity queries. Then, advanced algorithms for roadmaps and polynomial system solving allows us to achieve the announced complexity bound.

Many details are hidden in this description. In particular, we use infinitesimal deformations and techniques of semi-algebraic geometry. While infinitesimals are needed for proofs, they may be difficult to use in practice. On the algorithmic side, we go further exploiting the geometry of the problem to avoid using infinitesimals.

Structure of the paper In Section 2, we study the geometry of our problem and prove a series of auxiliary results (in particular Proposition 7, which coins the theoretical ingredient we need). Section 3 is devoted to describe the algorithm. Section 4 is devoted to the complexity analysis and Section 5 reports on the practical performances of our algorithm.

Acknowledgments. We thank the reviewers for their helpful comments.

2 The geometry of the problem

2.1 Candidates for isolated points

As above, let $f \in \boldsymbol{Q}[x_1, \ldots, x_n]$ and $\mathcal{H} \subset \boldsymbol{C}^n$ be the hypersurface defined by $f = 0$. Let $\mathbf{f}$ be a subset of $\boldsymbol{C}[x_1, \ldots, x_n]$, we denote by $V(\mathbf{f})$ the simultaneous vanishing locus in $\boldsymbol{C}^n$ of $\mathbf{f}$.

Lemma 1. *The set $\mathscr{Z}(\mathcal{H})$ is the (finite) union of the semi-algebraically connected components of $\mathcal{H} \cap \boldsymbol{R}^n$ which are a singleton.*

Proof. Recall that real algebraic sets have a finite number of semi-algebraically connected components [2, Theorem 5.21]. Let $\mathcal{C}$ be a semi-algebraically connected component of $\mathcal{H} \cap \boldsymbol{R}^n$.

Assume that C is not a singleton and take $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathcal{C}$ with $\boldsymbol{x} \neq \boldsymbol{y}$. Then, there exists a semi-algebraic continuous map $\gamma : [0, 1] \to \mathcal{C}$ s.t. $\gamma(0) = \boldsymbol{x}$ and $\gamma(1) = \boldsymbol{y}$; besides, since $\boldsymbol{x} \neq \boldsymbol{y}$, there exist $t \in (0, 1)$ such that $\gamma(t) \neq \boldsymbol{x}$. By continuity of γ and the norm function, any ball B centered at $\boldsymbol{x}$ contains a point $\gamma(t) \neq \boldsymbol{x}$.

Now assume that $\mathcal{C} = \{\boldsymbol{x}\}$. Observe that $\mathcal{H} \cap \boldsymbol{R}^n - \{\boldsymbol{x}\}$ is closed (since semi-algebraically connected components of real algebraic sets are closed). Since $\mathcal{H} \cap \boldsymbol{R}^n$ is bounded, we deduce that $\mathcal{H} \cap \boldsymbol{R}^n - \{\boldsymbol{x}\}$ is closed and bounded. Then, the map $\boldsymbol{y} \to \|\boldsymbol{y} - \boldsymbol{x}\|^2$ reaches a minimum over $\mathcal{H} \cap \boldsymbol{R}^n - \{\boldsymbol{x}\}$. Let e be this minimum value. We deduce that any ball centered at $\boldsymbol{x}$ of radius less than e does not meet $\mathcal{H} \cap \boldsymbol{R}^n - \{\boldsymbol{x}\}$. □

To compute those connected components of $\mathcal{H} \cap \boldsymbol{R}^n$ which are singletons, we use classical objects of optimization and Morse theory which are mainly *polar varieties*. Let $\boldsymbol{K}$ be an algebraically closed field, let $\phi \in \boldsymbol{K}[x_1, \ldots, x_n]$ which defines the polynomial mapping $(x_1, \ldots, x_n) \mapsto \phi(x_1, \ldots, x_n)$ and $V \subset \boldsymbol{K}^n$ be a smooth equidimensional algebraic set. We denote by $W(\phi, V)$ the set of critical points of the restriction of ϕ to V. If c is the co-dimension of V and $(g_1, \ldots, g_s)$ generates the vanishing ideal associated to V, then $W(\phi, V)$ is the subset of V at which the Jacobian matrix associated to $(g_1, \ldots, g_s, \phi)$ has rank less than or equal to c (see e.g., [28, Subsection 3.1]).

In particular, the case where ϕ is replaced by the canonical projection on the i-th coordinate

$$\pi_i : (x_1, \ldots, x_n) \mapsto x_i,$$

is excessively used throughout our paper.

In our context, we do not assume that $\mathcal{H}$ is smooth. Hence, to exploit strong topological properties of polar varieties, we retrieve a smooth situation using deformation techniques. We consider an infinitesimal ε, i.e., a transcendental element over $\boldsymbol{R}$ such that $0 < \varepsilon < r$ for any positive element $r \in \boldsymbol{R}$, and the field of Puiseux series over $\boldsymbol{R}$, denoted by

$$\boldsymbol{R}\langle\varepsilon\rangle = \left\{ \textstyle\sum_{i \geq i_0} a_i \varepsilon^{i/q} \mid i \in \mathbb{N}, i_0 \in \mathbb{Z}, q \in \mathbb{N} - \{0\}, a_i \in \boldsymbol{R} \right\}.$$

IBM - Ex. 1002-Part 10 of 10, Page 3408
IPR2025-00379 (IBM v. Croga)

Recall that $\boldsymbol{R}\langle\varepsilon\rangle$ is a real closed field [2, Theorem 2.91]. One defines $\boldsymbol{C}\langle\varepsilon\rangle$ as for $\boldsymbol{R}\langle\varepsilon\rangle$ but taking the coefficients of the series in $\boldsymbol{C}$. Recall that $\boldsymbol{C}\langle\varepsilon\rangle$ is an algebraic closure of $\boldsymbol{R}\langle\varepsilon\rangle$ [2, Theorem 2.17]. Consider $\sigma = \sum_{i\geq i_0} a_i\varepsilon^{i/q} \in \boldsymbol{R}\langle\varepsilon\rangle$ with $a_{i_0} \neq 0$. Then, a_{i_0} is called the *valuation* of σ. When $i_0 \geq 0$, σ is said to be *bounded over* $\boldsymbol{R}$ and the set of bounded elements of $\boldsymbol{R}\langle\varepsilon\rangle$ is denoted by $\boldsymbol{R}\langle\varepsilon\rangle_b$. One defines the function $\lim_\varepsilon : \boldsymbol{R}\langle\varepsilon\rangle_b \to \boldsymbol{R}$ that maps σ to a_0 (which is 0 when $i_0 > 0$) and writes $\lim_\varepsilon \sigma = a_0$; note that $\lim_\varepsilon$ is a ring homomorphism from $\boldsymbol{R}\langle\varepsilon\rangle_b$ to $\boldsymbol{R}$. All these definitions extend to $\boldsymbol{R}\langle\varepsilon\rangle^n$ componentwise. For a semi-algebraic set $\mathcal{S} \subset \boldsymbol{R}\langle\varepsilon\rangle^n$, we naturally define the limit of $\mathcal{S}$ as $\lim_\varepsilon \mathcal{S} = \{\lim_\varepsilon \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{S} \text{ and } \boldsymbol{x} \text{ is bounded over } \boldsymbol{R}\}$.

Let $\mathcal{S} \subset \boldsymbol{R}^n$ be a semi-algebraic set defined by a semi-algebraic formula Φ. We denote by $\mathrm{ext}(\mathcal{S}, \boldsymbol{R}\langle\varepsilon\rangle)$ the semi-algebraic set of points which are solutions of Φ in $\boldsymbol{R}\langle\varepsilon\rangle^n$. We refer to [2, Chap. 2] for more details on infinitesimals and real Puiseux series.

By e.g., [22, Lemma 3.5], $\mathcal{H}_\varepsilon$ and $\mathcal{H}_{-\varepsilon}$ respectively defined by $f = \varepsilon$ and $f = -\varepsilon$ are two disjoint smooth algebraic sets in $\boldsymbol{C}\langle\varepsilon\rangle^n$.

Lemma 2. *For any $\boldsymbol{x}$ lying in a bounded connected component of $\mathcal{H} \cap \boldsymbol{R}^n$, there exists a point $\boldsymbol{x}_\varepsilon \in (\mathcal{H}_\varepsilon \cup \mathcal{H}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle_b^n$ such that $\lim_\varepsilon \boldsymbol{x}_\varepsilon = \boldsymbol{x}$. For such a point $\boldsymbol{x}_\varepsilon$, let $\mathcal{C}_\varepsilon$ be the connected component of $(\mathcal{H}_\varepsilon \cup \mathcal{H}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^n$ containing $\boldsymbol{x}_\varepsilon$. Then, $\mathcal{C}_\varepsilon$ is bounded over $\boldsymbol{R}$.*

Proof. See [22, Lemma 3.6] for the first claim. The second part can be deduced following the proof of [2, Proposition 12.51]. □

Proposition 3. *Assume that $\mathscr{Z}(\mathcal{H})$ is not empty and let $\boldsymbol{x} \in \mathscr{Z}(\mathcal{H})$. There exists a semi-algebraically connected component $\mathcal{C}_\varepsilon$ that is bounded over $\boldsymbol{R}$ of $(\mathcal{H}_\varepsilon \cup \mathcal{H}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^n$ such that $\lim_\varepsilon \mathcal{C}_\varepsilon = \{\boldsymbol{x}\}$.*

Consequently, for $1 \leq i \leq n$, there exists an $\boldsymbol{x}_\varepsilon \in (W(\pi_i, \mathcal{H}_\varepsilon) \cup W(\pi_i, \mathcal{H}_{-\varepsilon})) \cap \mathcal{C}_\varepsilon$ such that $\lim_\varepsilon \boldsymbol{x}_\varepsilon = \boldsymbol{x}$. Hence we have that

$$\mathscr{Z}(\mathcal{H}) \subset \cap_{i=1}^n \lim_\varepsilon((W(\pi_i, \mathcal{H}_\varepsilon) \cup W(\pi_i, \mathcal{H}_{-\varepsilon})) \cap \boldsymbol{R}\langle\varepsilon\rangle_b^n).$$

Proof. By Lemma 2, there exists $\boldsymbol{x}_\varepsilon \in (\mathcal{H}_\varepsilon \cup \mathcal{H}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^n$ such that $\lim_\varepsilon \boldsymbol{x}_\varepsilon = \boldsymbol{x}$. Assume that $\boldsymbol{x}_\varepsilon \in \mathcal{H}_\varepsilon$ and let $\mathcal{C}_\varepsilon$ be the connected component of $\mathcal{H}_\varepsilon \cap \boldsymbol{R}\langle\varepsilon\rangle^n$ containing $\boldsymbol{x}_\varepsilon$. Again, by Lemma 2, $\mathcal{C}_\varepsilon$ is bounded over $\boldsymbol{R}$. We prove that $\lim_\varepsilon \mathcal{C}_\varepsilon = \{\boldsymbol{x}\}$ by contradiction. The case $\boldsymbol{x}_\varepsilon \in \mathcal{H}_{-\varepsilon}$ is done similarly.

Assume that there exists a point $\boldsymbol{y}_\varepsilon \in \mathcal{C}_\varepsilon$ such that $\lim_\varepsilon \boldsymbol{y}_\varepsilon = \boldsymbol{y}$ and $\boldsymbol{y} \neq \boldsymbol{x}$. Since $\mathcal{C}_\varepsilon$ is semi-algebraically connected, there exists a semi-algebraically continuous function $\gamma : \mathrm{ext}([0,1], \boldsymbol{R}\langle\varepsilon\rangle) \to \mathcal{C}_\varepsilon$ such that $\gamma(0) = \boldsymbol{x}_\varepsilon$ and $\gamma(1) = \boldsymbol{y}_\varepsilon$. By [2, Proposition 12.49], $\lim_\varepsilon \mathrm{Im}(\gamma)$ is connected and contains $\boldsymbol{x}$ and $\boldsymbol{y}$. As $\lim_\varepsilon$ is a ring homomorphism, $f(\lim_\varepsilon \gamma(t)) = \lim_\varepsilon f(\gamma(t)) = 0$, so $\lim_\varepsilon \mathrm{Im}(\gamma)$ is contained in $\mathcal{H} \cap \boldsymbol{R}^n$. This contradicts the isolatedness of $\boldsymbol{x}$, then we conclude that $\lim_\varepsilon \mathcal{C}_\varepsilon = \{\boldsymbol{x}\}$.

Since $\mathcal{C}_\varepsilon$ is a semi-algebraically connected component of the real algebraic set $\mathcal{H}_\varepsilon \cap \boldsymbol{R}\langle\varepsilon\rangle^n$, it is closed. Also, $\mathcal{C}_\varepsilon$ is bounded over $\boldsymbol{R}$. Hence, for any $1 \leq i \leq n$, the projection π_i reaches its extrema over $\mathcal{C}_\varepsilon$ [2, Proposition 7.6], which implies that $\mathcal{C}_\varepsilon \cap W(\pi_i, \mathcal{H}_\varepsilon)$ is non-empty. Take $\boldsymbol{x}_\varepsilon \in W(\pi_i, \mathcal{H}_\varepsilon) \cap \mathcal{C}_\varepsilon$, then $\boldsymbol{x}_\varepsilon$ is bounded over $\boldsymbol{R}$ and its limit is $\boldsymbol{x}$. Thus, $\mathscr{Z}(\mathcal{H}) \subset \lim_\varepsilon(W(\pi_i, \mathcal{H}_\varepsilon) \cap \boldsymbol{R}\langle\varepsilon\rangle_b^n)$ for any $1 \leq i \leq n$, which implies $\mathscr{Z}(\mathcal{H}) \subset \cap_{i=1}^n \lim_\varepsilon(W(\pi_i, \mathcal{H}_\varepsilon) \cap \boldsymbol{R}\langle\varepsilon\rangle_b^n)$. □

2.2 Simplification

We introduce in this subsection a method to reduce our problem to the case where $\mathcal{H} \cap \boldsymbol{R}^n$ is bounded for all $\boldsymbol{x} \in \boldsymbol{R}^n$. Such assumptions are required to prove the results in Subsection 2.3. Our technique is inspired by [2, Section 12.6]. The idea is to associate to the possibly unbounded algebraic set $\mathcal{H} \cap \boldsymbol{R}^n$ a bounded real algebraic set whose isolated points are strongly related to $\mathscr{Z}(\mathcal{H})$. The construction of such an algebraic set is as follows.

Let x_{n+1} be a new variable and $0 < \rho \in \boldsymbol{R}$ such that ρ is greater than the Euclidean norm $\|\cdot\|$ of every isolated point of $\mathcal{H} \cap \boldsymbol{R}^n$. Note that such a ρ can be obtained from a finite set of points containing the the isolated points of $\mathcal{H} \cap \boldsymbol{R}^n$. We explain in Subsection 3.2 how to compute such a finite set.

We consider the algebraic set $\mathcal{V}$ defined by the system

$$f = 0,\ x_1^2 + \ldots + x_n^2 + x_{n+1}^2 - \rho^2 = 0.$$

Let $\pi_{\boldsymbol{x}}$ be the projection $(x_1, \ldots, x_n, x_{n+1}) \mapsto (x_1, \ldots, x_n)$.

The real counterpart of $\mathcal{V}$ is the intersection of $\mathcal{H}$ lifted to $\boldsymbol{R}^{n+1}$ with the sphere of center $\mathbf{0}$ and radius ρ. Therefore, $\mathcal{V}$ is a bounded real algebraic set in $\boldsymbol{R}^{n+1}$. Moreover, the restriction of $\pi_{\boldsymbol{x}}$ to $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ is exactly $\mathcal{H} \cap B(\mathbf{0}, \rho)$. By the definition of ρ, this image contains all the real isolated points of $\mathcal{H}$. Lemma 4 below relates $\mathscr{Z}(\mathcal{H})$ to the isolated points of $\mathcal{V} \cap \boldsymbol{R}^{n+1}$.

Lemma 4. *Let $\mathcal{V}$ and $\pi_{\boldsymbol{x}}$ as above. We denote by $\mathscr{Z}(\mathcal{V}) \subset \boldsymbol{R}^{n+1}$ the set of real isolated points of $\mathcal{V}$ with non-zero x_{n+1} coordinate. Then, $\pi_{\boldsymbol{x}}(\mathscr{Z}(\mathcal{V})) = \mathscr{Z}(\mathcal{H})$.*

Proof. Note that $\pi_{\boldsymbol{x}}(\mathcal{V} \cap \boldsymbol{R}^{n+1}) = (\mathcal{H} \cap \boldsymbol{R}^n) \cap B(\mathbf{0}, \rho)$. We consider a real isolated point $\boldsymbol{x}' = (\alpha_1, \ldots, \alpha_n, \alpha_{n+1})$ of $\mathcal{V}$ with $\alpha_{n+1} \neq 0$ and $\boldsymbol{x} = \pi_{\boldsymbol{x}}(\boldsymbol{x}') = (\alpha_1, \ldots, \alpha_n)$. Assume by contradiction that $\boldsymbol{x} \notin \mathscr{Z}(\mathcal{H})$, we will prove that $\boldsymbol{x}' \notin \mathscr{Z}(\mathcal{V})$, i.e., for any $r > 0$, there exists $\boldsymbol{y}' = (\beta_1, \ldots, \beta_n, \beta_{n+1}) \in \mathcal{V} \cap \boldsymbol{R}^{n+1}$ such that $\|\boldsymbol{y}' - \boldsymbol{x}'\| < r$. Since $\boldsymbol{x}$ is not isolated, there exists a point $\boldsymbol{y} \neq \boldsymbol{x}$ such that $\|\boldsymbol{y} - \boldsymbol{x}\| < \frac{r}{1+2\rho/|\alpha_{n+1}|}$. Let $\boldsymbol{y}' \in \pi_{\boldsymbol{x}}^{-1}(\boldsymbol{y})$ such that $\alpha_{n+1}\beta_{n+1} \geq 0$. We have that $\|\boldsymbol{x}\|^2 + \alpha_{n+1}^2 = \|\boldsymbol{y}\|^2 + \beta_{n+1}^2 = \rho^2$. Now we estimate

$$|\|\boldsymbol{y}\|^2 - \|\boldsymbol{x}\|^2| = (\|\boldsymbol{x}\| + \|\boldsymbol{y}\|) \cdot |\|\boldsymbol{y}\| - \|\boldsymbol{x}\|| \leq 2\rho \cdot \|\boldsymbol{y} - \boldsymbol{x}\|,$$

IBM - Ex. 1002-Part 10 of 10, Page 3409
IPR2025-00379 (IBM v. Croga)

$$|\alpha_{n+1} - \beta_{n+1}| \leq \frac{|\alpha_{n+1}^2 - \beta_{n+1}^2|}{|\alpha_{n+1}|} = \frac{|\|\boldsymbol{y}\|^2 - \|\boldsymbol{x}\|^2|}{|\alpha_{n+1}|} \leq \frac{2\rho \cdot \|\boldsymbol{y} - \boldsymbol{x}\|}{|\alpha_{n+1}|}.$$

Finally,

$$\|\boldsymbol{y}' - \boldsymbol{x}'\| \leq \|\boldsymbol{y} - \boldsymbol{x}\| + |\alpha_{n+1} - \alpha_{n+1}| \leq \left(1 + \frac{2\rho}{|\alpha_{n+1}|}\right) \|\boldsymbol{y} - \boldsymbol{x}\| < r.$$

So, $\boldsymbol{x}'$ is not isolated in $\mathcal{V} \cap \boldsymbol{R}^{n+1}$. This contradiction implies that $\pi_{\boldsymbol{x}}(\mathscr{Z}(\mathcal{V})) \subset \mathscr{Z}(\mathcal{H})$.

It remains to prove that $\mathscr{Z}(\mathcal{H}) \subset \pi_{\boldsymbol{x}}(\mathscr{Z}(\mathcal{V}))$. For any real isolated point $\boldsymbol{x} \in \mathscr{Z}(\mathcal{H})$, we consider a ball $B(\boldsymbol{x}, r') \subset B(\boldsymbol{0}, \rho) \subset \boldsymbol{R}^n$ such that $B(\boldsymbol{x}, r') \cap \mathcal{H} = \{\boldsymbol{x}\}$. We have that $\pi_{\boldsymbol{x}}^{-1}(B(\boldsymbol{x}, r')) \cap \mathcal{V} \cap \boldsymbol{R}^{n+1}$ is equal to $\pi_{\boldsymbol{x}}^{-1}(\boldsymbol{x}) \cap \mathcal{V} \cap \boldsymbol{R}^{n+1}$, which is finite. So, all the points in $\pi_{\boldsymbol{x}}^{-1}(B(\boldsymbol{x}, r')) \cap \mathcal{V} \cap \boldsymbol{R}^{n+1}$ are isolated. Since $\mathscr{Z}(\mathcal{H}) \subset B(\boldsymbol{0}, \rho)$, we deduce that $\mathscr{Z}(\mathcal{H})$ is contained in $\pi_{\boldsymbol{x}}(\mathscr{Z}(\mathcal{V}))$.

Thus, we conclude that $\pi_{\boldsymbol{x}}(\mathscr{Z}(\mathcal{V})) = \mathscr{Z}(\mathcal{H})$. □

Note that the condition $x_{n+1} \neq 0$ is crucial. For a connected component $\mathcal{C}$ of $\mathcal{H} \cap \boldsymbol{R}^n$ that is not a singleton, its intersection with the closed ball $\overline{B(\boldsymbol{0}, \rho)}$ can have an isolated point on the boundary of the ball, which corresponds to an isolated point of $\mathcal{V} \cap \boldsymbol{R}^{n+1}$. This situation depends on the choice of ρ and can be easily detected by checking the vanishing of the coordinate x_{n+1}.

2.3 Identification of isolated points

By Proposition 3, the real points of $\cap_{i=1}^n \lim_\varepsilon W(\pi_i, \mathcal{H}_\varepsilon)$ are potential isolated points of $\mathcal{H} \cap \boldsymbol{R}^n$. We study now how to identify, among those candidates, which points are truely isolated.

We use the same $g = x_1^2 + \ldots + x_{n+1}^2 - \rho^2$ and $\mathcal{V} = V(f, g) \subset \boldsymbol{C}^{n+1}$ as in Subsection 2.2. Let $\mathcal{V}_\varepsilon = V(f - \varepsilon, g)$ and $\mathcal{V}_{-\varepsilon} = V(f + \varepsilon, g)$, note that they are both algebraic subsets of $\boldsymbol{C}\langle\varepsilon\rangle^{n+1}$.

Lemma 5. *Let $\boldsymbol{x} \in \mathcal{V} \cap \boldsymbol{R}^{n+1}$ such that its x_{n+1} coordinate is non-zero. Then, $\boldsymbol{x}$ is not an isolated point of $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ if and only if there exists a semi-algebraically connected component $\mathcal{C}_\varepsilon$ of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$, bounded over $\boldsymbol{R}$, such that $\{\boldsymbol{x}\} \subsetneq \lim_\varepsilon \mathcal{C}_\varepsilon$.*

Proof. Let $\boldsymbol{x} = (\alpha_1, \ldots, \alpha_{n+1}) \in \mathcal{V} \cap \boldsymbol{R}^{n+1}$ such that $\alpha_{n+1} \neq 0$. As $f(\alpha_1, \ldots, \alpha_n) = 0$, by Lemma 2, there exists a point $\boldsymbol{x}_\varepsilon = (\beta_1, \ldots, \beta_{n+1}) \in \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ such that $(\beta_1, \ldots, \beta_n) \in (\mathcal{H}_\varepsilon \cup \mathcal{H}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^n$ and $\lim_\varepsilon(\beta_1, \ldots, \beta_n) = (\alpha_1, \ldots, \alpha_n)$. Since $\alpha_{n+1} \neq 0$, we can choose β_{n+1} such that $g(\boldsymbol{x}_\varepsilon) = 0$. Therefore, for any $\boldsymbol{x}$ as above, there exists $\boldsymbol{x}_\varepsilon \in (\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ such that $\lim_\varepsilon \boldsymbol{x}_\varepsilon = \boldsymbol{x}$.

Since $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ lies on the sphere (in $\boldsymbol{R}\langle\varepsilon\rangle^{n+1}$) defined by $g = 0$, every connected component of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ is bounded over $\boldsymbol{R}$. Hence, the points of $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ whose x_{n+1} coordinates are not zero are contained in $\lim_\varepsilon (\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$.

Let $\boldsymbol{x}$ be a non-isolated point of $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ whose x_{n+1}-coordinate is not zero. We assume by contradiction that for any semi-algebraically connected component $\mathcal{C}_\varepsilon$ of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ (which is bounded over $\boldsymbol{R}$ by above), then it happens that either $\lim_\varepsilon \mathcal{C}_\varepsilon = \{\boldsymbol{x}\}$ or $\boldsymbol{x} \notin \lim_\varepsilon \mathcal{C}_\varepsilon$.

Since $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ has finitely many connected components, the number of connected components of the second type is also finite. Since $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ is not a singleton (by the existence of $\boldsymbol{x}$), the connected components of the second type exist. So, we enumerate them as $\mathcal{C}_1, \ldots, \mathcal{C}_k$ and $\boldsymbol{x} \notin \lim_\varepsilon \mathcal{C}_j$ for $1 \leq j \leq k$.

As $\boldsymbol{x}$ is not isolated in $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ with non-zero x_{n+1} coordinate by assumption, there exists a sequence of points $(\boldsymbol{x}_i)_{i \geq 0}$ in $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ of non-zero x_{n+1} coordinates that converges to $\boldsymbol{x}$. Since there are finitely many $\mathcal{C}_i$, there exists an index j such that $\lim_\varepsilon \mathcal{C}_j$ contains a sub-sequence of $(\boldsymbol{x}_i)_{i \geq 0}$. By Proposition 12.49 [BPR], the limit of the semi-algebraically connected component $\mathcal{C}_j$ (which is bounded over $\boldsymbol{R}$) is a closed and connected semi-algebraic set. It follows that $\boldsymbol{x} \in \lim_\varepsilon \mathcal{C}_j$, which is a contradiction. Therefore, there exists a semi-algebraically connected component of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$, bounded over $\boldsymbol{R}$, such that $\{\boldsymbol{x}\} \subsetneq \lim_\varepsilon \mathcal{C}_\varepsilon$.

It remains to prove the reverse implication. Assume that $\{\boldsymbol{x}\} \subsetneq \lim_\varepsilon \mathcal{C}_\varepsilon$ for some semi-algebraically connected component $\mathcal{C}_\varepsilon$ of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ that is bounded over $\boldsymbol{R}$. As $\lim_\varepsilon \mathcal{C}_\varepsilon$ is connected, we finish the proof. □

Lemma 6. *Let $\boldsymbol{x} \in \mathcal{V} \cap \boldsymbol{R}^{n+1}$ whose x_{n+1} coordinate is non-zero. Assume that $\boldsymbol{x}$ is not an isolated point of $\mathcal{V} \cap \boldsymbol{R}^{n+1}$. For any semi-algebraically connected component $\mathcal{C}_\varepsilon$ of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}^{n+1}$, bounded over $\boldsymbol{R}$, such that $\{\boldsymbol{x}\} \subsetneq \lim_\varepsilon \mathcal{C}_\varepsilon$, there exists $1 \leq i \leq n$ such that $\mathcal{C}_\varepsilon \cap (W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon}))$ contains a point $\boldsymbol{x}'_\varepsilon$ which satisfies $\lim_\varepsilon \boldsymbol{x}'_\varepsilon \neq \boldsymbol{x}$.*

Proof. Let $\mathcal{C}_\varepsilon$ be semi-algebraically connected component of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$, bounded over $\boldsymbol{R}$, such that $\{\boldsymbol{x}\} \subsetneq \lim_\varepsilon \mathcal{C}_\varepsilon$. Lemma 5 ensures the existence of such a connected component $\mathcal{C}_\varepsilon$.

Now let $\boldsymbol{x}_\varepsilon$ and $\boldsymbol{y}_\varepsilon$ be two points contained in $\mathcal{C}_\varepsilon$ such that $\lim_\varepsilon \boldsymbol{x}_\varepsilon = \boldsymbol{x}$, $\lim_\varepsilon \boldsymbol{y}_\varepsilon = \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$. Let $\boldsymbol{x} = (\alpha_1, \ldots, \alpha_{n+1})$ and $\boldsymbol{y} = (\beta_1, \ldots, \beta_{n+1})$. Since $\boldsymbol{x} \neq \boldsymbol{y}$, there exists $1 \leq i \leq n+1$ such that $\alpha_i \neq \beta_i$. Note that if $(\alpha_1, \ldots, \alpha_n) = (\beta_1, \ldots, \beta_n)$ for any $\boldsymbol{y} \in \lim_\varepsilon \mathcal{C}_\varepsilon$, then $\lim_\varepsilon \mathcal{C}_\varepsilon$ contains at most two points (by the constraint $g = 0$). However, since $\lim_\varepsilon \mathcal{C}_\varepsilon$ is connected and contains at least two points, it must be an infinite set. So, we can choose $\boldsymbol{y}$ such that have that $1 \leq i \leq n$.

As $\mathcal{C}_\varepsilon$ is closed in $\boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ (as a connected component of an algebraic set) and bounded over $\boldsymbol{R}$ by definition, its projection on the x_i coordinate is a closed interval $[a, b] \subset \boldsymbol{R}\langle\varepsilon\rangle$ (see [2, Theorem 3.23]), which is bounded over $\boldsymbol{R}$ (because $\mathcal{C}_\varepsilon$ is). Also, since $[a, b]$ is closed, there exist $\boldsymbol{x}'_a$ and $\boldsymbol{x}'_b$ in $\boldsymbol{R}\langle\varepsilon\rangle^{n+1}$ such that $\boldsymbol{x}'_a \in \pi_i^{-1}(a) \cap \mathcal{C}_\varepsilon \cap (W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon}))$ and

IBM - Ex. 1002-Part 10 of 10, Page 3410
IPR2025-00379 (IBM v. Croga)

$\boldsymbol{x}'_b \in \pi_i^{-1}(b) \cap \mathcal{C}_\varepsilon \cap (W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon}))$. Since $\alpha_i \neq \beta_i$ both lying in $\boldsymbol{R}$, $\{\alpha_i, \beta_i\} \subset [\lim_\varepsilon a, \lim_\varepsilon b]$ implies that $\lim_\varepsilon a \neq \lim_\varepsilon b$. It follows that $\lim_\varepsilon \boldsymbol{x}'_a \neq \lim_\varepsilon \boldsymbol{x}'_b$. Thus, at least one point among $\lim_\varepsilon \boldsymbol{x}'_a$ and $\lim_\varepsilon \boldsymbol{x}'_b$ does not coincide with $\boldsymbol{x}$. Hence, there exists a point $\boldsymbol{x}'_\varepsilon$ in $\mathcal{C}_\varepsilon \cap (W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon}))$ such that $\lim_\varepsilon \boldsymbol{x}'_\varepsilon \neq \boldsymbol{x}$. □

We can easily deduce from Lemma 5 and Lemma 6 the following proposition, which is the main ingredient of our algorithm.

Proposition 7. *Let $\boldsymbol{x} \in \cap_{i=1}^n \lim_\varepsilon W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon})$ whose x_{n+1} coordinate is non-zero. Then, $\boldsymbol{x}$ is not an isolated point of $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ if and only if there exist $1 \leq i \leq n$ and a connected component $\mathcal{C}_\varepsilon$ of $\mathcal{V}_\varepsilon \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$, which is bounded over $\boldsymbol{R}$, such that $\mathcal{C}_\varepsilon \cap W(\pi_i, \mathcal{H}_\varepsilon)$ contains $\boldsymbol{x}_\varepsilon$, $\boldsymbol{x}'_\varepsilon$ satisfying $\boldsymbol{x} = \lim_\varepsilon \boldsymbol{x}_\varepsilon \neq \lim_\varepsilon \boldsymbol{x}'_\varepsilon$.*

3 Algorithm

3.1 General description

The algorithm takes as input a polynomial $f \in \boldsymbol{R}[x_1, \ldots, x_n]$.

The first step consists in computing a parametrization $\mathfrak{P}$ encoding a finite set of points which contains $\mathscr{Z}(\mathcal{H})$. Let $\mathcal{H}_\varepsilon$ and $\mathcal{H}_{-\varepsilon}$ be the algebraic subsets of $\boldsymbol{C}\langle\varepsilon\rangle^n$ respectively defined by $f = \varepsilon$ and $f = -\varepsilon$. By Proposition 3, the set $\cap_{i=1}^n \lim_\varepsilon W(\pi_i, \mathcal{H}_\varepsilon) \cup W(\pi_i, \mathcal{H}_{-\varepsilon})$ contains the real isolated points of $\mathcal{H}$. To ensure that this set is finite, we use *generically chosen* linear change of coordinates.

Given a matrix $\boldsymbol{A} \in GL_n(\boldsymbol{Q})$, a polynomial $p \in \boldsymbol{Q}[x_1, \ldots, x_n]$ and an algebraic set $\mathcal{S} \subset \boldsymbol{C}^n$, we denote by $p^{\boldsymbol{A}}$ the polynomial $p(\boldsymbol{A} \cdot \boldsymbol{x})$ obtained by applying the change of variables $\boldsymbol{A}$ to p and $\mathcal{S}^{\boldsymbol{A}} = \{\boldsymbol{A}^{-1} \cdot \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{S}\}$. Then, we have that $V(p)^{\boldsymbol{A}} = V(p^{\boldsymbol{A}})$.

In [25], it is proved that, with $\boldsymbol{A}$ outside a prescribed proper Zariski closed subset of $GL_n(\boldsymbol{Q})$, $W(\pi_i, \mathcal{H}_\varepsilon^{\boldsymbol{A}}) \cup W(\pi_i, \mathcal{H}_{-\varepsilon}^{\boldsymbol{A}})$ is finite for $1 \leq i \leq n$. Additionally, since $\boldsymbol{A}$ is assumed to be generically chosen, [24] shows that the ideal $\left\langle \ell \cdot \frac{\partial f^{\boldsymbol{A}}}{\partial x_i} - 1, \frac{\partial f^{\boldsymbol{A}}}{\partial x_j} \text{ for all } j \neq i \right\rangle$ defines either an empty set or a one-equidimensional algebraic set, where ℓ is a new variable. Those extra assumptions are required in our subroutine Candidates (see the next subsection). Note that, for any matrix $\boldsymbol{A}$, the real isolated points of $\mathcal{H}^{\boldsymbol{A}}$ is the image of $\mathscr{Z}(\mathcal{H})$ by the linear mapping associated to $\boldsymbol{A}^{-1}$. Thus, in practice, we will choose randomly a $\boldsymbol{A} \in GL_{n+1}(\boldsymbol{Q})$, compute the real isolated points of $\mathcal{H}^{\boldsymbol{A}}$, and then go back to $\mathscr{Z}(\mathcal{H})$ by applying the change of coordinates induced by $\boldsymbol{A}^{-1}$. This random choice of $\boldsymbol{A}$ makes our algorithm probabilistic.

The next step consists of identifying those of the candidates which are isolated in $\mathcal{H}^{\boldsymbol{A}} \cap \boldsymbol{R}^n$; this step relies on Proposition 7. To reduce our problem to the context where Proposition 7 can be applied, we use Lemma 4. One needs to compute $\rho \in \boldsymbol{R}$, such that ρ is larger than the maximum norm of the real isolated points we want to compute. This value of ρ can be easily obtained by isolating the real roots of the zero-dimensional parametrization encoding the candidates. Further, we call GetNormBound a subroutine which takes as input $\mathfrak{P}$ and returns ρ as we just sketched. We let $g = x_1^2 + \ldots + x_n^2 + x_{n+1}^2 - \rho^2$. By Lemma 4, $\mathscr{Z}(\mathcal{H})$ is the projection of the set of real isolated points of the algebraic set $\mathcal{V}$ defined by $f = g = 0$ at which $x_{n+1} \neq 0$. Let $\mathcal{X}$ be the set of points of $\mathcal{V}$ projecting to the candidates encoded by $\mathfrak{P}$.

Proposition 7 would lead us to compute $W(\pi_i, \mathcal{V}_\varepsilon^{\boldsymbol{A}}) \cup W(\pi_i, \mathcal{V}_{-\varepsilon}^{\boldsymbol{A}})$ as well as a roadmap of $\mathcal{V}_\varepsilon^{\boldsymbol{A}} \cup \mathcal{V}_{-\varepsilon}^{\boldsymbol{A}}$. As explained in the introduction, this induces computations over the ground field $\boldsymbol{R}\langle\varepsilon\rangle$ which we want to avoid. We bypass this computational difficulty as follows. We compute a roadmap $\mathcal{R}_e$ for $\mathcal{V}_e^{\boldsymbol{A}} \cup \mathcal{V}_{-e}^{\boldsymbol{A}} \cap \boldsymbol{R}^{n+1}$ (defined by $\{f^{\boldsymbol{A}} = e, g = 0\}$ and $\{f^{\boldsymbol{A}} = -e, g = 0\}$ respectively) for e small enough (see Subsection 3.3) and define a semi-algebraic curve $\mathcal{K}$ containing $\mathcal{X}$ such that $\boldsymbol{x} \in \mathcal{X}$ is isolated in $\mathcal{V}^{\boldsymbol{A}} \cap \boldsymbol{R}^{n+1}$ if and only if it is not connected to any other $\boldsymbol{x}' \in \mathcal{X}$ by $\mathcal{K}$. We call IsIsolated the subroutine that takes as input $\mathfrak{P}$, $f^{\boldsymbol{A}}$ and g and returns $\mathfrak{P}$ with isolating boxes B of the real points of defined by $\mathfrak{P}$ which are isolated in $\mathcal{V}^{\boldsymbol{A}} \cap \boldsymbol{R}^{n+1}$.

Once the real isolated points of $\mathcal{V}^{\boldsymbol{A}}$ is computed, we remove the boxes corresponding to points at which $x_{n+1} = 0$ and project the remaining points on the $(x_1, \ldots, x_n)$-space to obtain the isolated points of $\mathcal{H}^{\boldsymbol{A}}$. This whole step uses a subroutine which we call Remove (see [28, Appendix J]). Finally, we reverse the change of variable by applying $\boldsymbol{A}^{-1}$ to get $\mathscr{Z}(\mathcal{H})$.

We summarize our discussion in Algorithm 1 below.

3.2 Computation of candidates

Further, we let $\mathcal{H}_\varepsilon^{\boldsymbol{A}}$ (resp. $\mathcal{H}_{-\varepsilon}^{\boldsymbol{A}}$) be the algebraic set associated to $f^{\boldsymbol{A}} = \varepsilon$ (resp. $f^{\boldsymbol{A}} = -\varepsilon$). To avoid to overload notation, we omit the change of variables $\boldsymbol{A}$ as upper script. Let ℓ be a new variable. For $1 \leq i \leq n$, I_i denotes the ideal of $\boldsymbol{Q}[\ell, x_1, \ldots, x_n]$ generated by the set of polynomials $\left\{\ell \cdot \frac{\partial f}{\partial x_i} - 1, \frac{\partial f}{\partial x_j} \quad \text{for all } j \neq i\right\}$.

Following the discussion in Subsection 3.1, the algebraic set associated to I_i is either empty or one-equidimensional and $W(\pi_i, \mathcal{H}_\varepsilon) \cup W(\pi_i, \mathcal{H}_{-\varepsilon})$ is finite. Hence, [24, Theorem 1] shows that the algebraic set associated to the ideal $\langle f \rangle + (I_i \cap \boldsymbol{Q}[x_1, \ldots, x_n])$ is zero-dimensional and contains $\lim_\varepsilon W(\pi_i, \mathcal{H}_\varepsilon) \cup W(\pi_i, \mathcal{H}_{-\varepsilon})$.

In our problem, the intersection of $\lim_\varepsilon W(\pi_i, \mathcal{H}_\varepsilon) \cup W(\pi_i, \mathcal{H}_{-\varepsilon})$ is needed rather than each limit itself. Hence, we use the inclusion

$$\cap_{i=1}^n \lim_\varepsilon W(\pi_i, \mathcal{H}_\varepsilon) \cup W(\pi_i, \mathcal{H}_{-\varepsilon}) \subset V\left(\langle f \rangle + \textstyle\sum_{i=1}^n I_i \cap \boldsymbol{Q}[x_1, \ldots, x_n]\right).$$

IBM - Ex. 1002-Part 10 of 10, Page 3411
IPR2025-00379 (IBM v. Croga)

Algorithm 1: IsolatedPoints

Input: A polynomial $f \in \boldsymbol{Q}[x_1, \ldots, x_n]$
Output: A zero-dimensional parametrization $\mathfrak{P}$ such that $\mathscr{Z}(\mathcal{H}) \subset Z(\mathfrak{P})$ and a set of boxes isolating $\mathscr{Z}(\mathcal{H})$

$\boldsymbol{A}$ chosen randomly in $GL_{n+1}(\boldsymbol{Q})$
$\mathfrak{P} \leftarrow$ Candidates$(f^{\boldsymbol{A}})$
$\rho \leftarrow$ GetNormBound$(\mathfrak{P})$
$g \leftarrow x_1^2 + \ldots + x_n^2 + x_{n+1}^2 - \rho^2$
$\mathfrak{P}, B \leftarrow$ IsIsolated$(\mathfrak{P}, f^{\boldsymbol{A}}, g)$
$\mathfrak{P}, B \leftarrow$ Removes$(\mathfrak{P}, B, x_{n+1})$
$\mathfrak{P}, B \leftarrow \mathfrak{P}^{\boldsymbol{A}^{-1}}, B^{\boldsymbol{A}^{-1}}$
return $(\mathfrak{P}, B)$

We can compute the algebraic set on the right-hand side as follows:

1. For each $1 \leq i \leq n$, compute a set G_i of generators of the ideal $I_i \cap \boldsymbol{Q}[x_1, \ldots, x_n]$.
2. Compute a zero-dimensional parametrization $\mathfrak{P}$ of the set of polynomials $\{f\} \cup G_1 \cup \ldots \cup G_n$.

Such computations mimic those in [24]. The complexity of this algorithm of course depends on the algebraic elimination procedure we use. For the complexity analysis in Section 4, we employ the geometric resolution [14]. It basically consists in computing a one-dimensional parametrization of the curve defined by I_i and next computes a zero-dimensional parametrization of the finite set obtained by intersecting this curve with the hypersurface defined by $f = 0$. We call ParametricCurve a subroutine that, taking the polynomial f and $1 \leq i \leq n$, computes a one-dimensional parametrization $\mathfrak{G}_i$ of the curve defined above. Also, let IntersectCurve be a subroutine that, given a one-dimensional rational parametrization $\mathfrak{G}_i$ and f, outputs a zero-dimensional parametrization $\mathfrak{P}_i$ of their intersection. Finally, we use a subroutine Intersection that, from the parametrizations $\mathfrak{P}_i$'s, computes a zero-dimensional parametrization of $\cap_{i=1}^n Z(\mathfrak{P}_i)$.

Algorithm 2: Algorithm Candidates

Input: The polynomial $f \in \boldsymbol{Q}[x_1, \ldots, x_n]$
Output: A zero-dimensional parametrization $\mathfrak{P}$

for $1 \leq i \leq n$ **do**
    $\mathfrak{G}_i \leftarrow$ ParametricCurve(g, i)
    $\mathfrak{P}_i \leftarrow$ IntersectCurve$(\mathfrak{G}_i, g)$
end
$\mathfrak{P} \leftarrow$ Intersection$(\mathfrak{P}_1, \ldots, \mathfrak{P}_n)$
return $\mathfrak{P}$

3.3 Description of IsIsolated

This subsection is devoted to the subroutine IsIsolated that identifies isolated points of $\mathcal{H}^{\boldsymbol{A}} \cap \boldsymbol{R}^n$ among the candidates $Z(\mathfrak{P}) \cap \boldsymbol{R}^n$ computed in the previous subsection. We keep using f to address $f^{\boldsymbol{A}}$. Let $\mathcal{P}$ be the set $\{\boldsymbol{x} = (x_1, \ldots, x_{n+1}) \in \boldsymbol{R}^{n+1} \mid (x_1, \ldots, x_n) \in Z(\mathfrak{P}), g(\boldsymbol{x}) = 0, x_{n+1} \neq 0\}$.

For $e \in \boldsymbol{R}$, $\mathcal{V}_e \in \boldsymbol{C}^{n+1}$ denotes the algebraic set defined by $f = e$ and $g = 0$. We follow the idea mentioned in the end of Subsection 3.1, that is to replace the infinitesimal ε by a sufficiently small $e \in \boldsymbol{R}$ then adapt the results of Subsection 2.3 to $\mathcal{V}_e$.

By definition, $\mathcal{V}_e \cap \boldsymbol{R}^{n+1}$ is bounded for any $e \in \boldsymbol{R}$. Let t be a new variable, $\pi_{\boldsymbol{x}} : (\boldsymbol{x}, t) \mapsto \boldsymbol{x}$ and $\pi_t : (\boldsymbol{x}, t) \mapsto t$. For a semi-algebraic set $\mathcal{S} \subset \boldsymbol{R}^{n+1} \times \boldsymbol{R}$ in the coordinate $(\boldsymbol{x}, t)$ and a subset $\mathcal{I}$ of $\boldsymbol{R}$, the notation $\mathcal{S}_{\mathcal{I}}$ stands for the fiber $\pi_t^{-1}(\mathcal{I}) \cap \mathcal{S}$. Let $\mathcal{V}_t = \{(\boldsymbol{x}, t) \in \boldsymbol{R}^{n+1} \times \boldsymbol{R} \mid f(\boldsymbol{x}) = t, g(\boldsymbol{x}) = 0\}$. Note that $\mathcal{V}_t$ is smooth. Recall that the set of critical values of the restriction of π_t to $\mathcal{V}_t$ is finite by the algebraic Sard's theorem (see e.g., [28, Proposition B.2]).

Since for $e \in \boldsymbol{R}$, the set $\mathcal{V}_e \cap \boldsymbol{R}^{n+1}$ is compact, the restriction of π_t to $\mathcal{V}_t$ is proper. Then, by Thom's isotopy lemma [9], π_t realizes a locally trivial fibration over any open connected subset of $\boldsymbol{R}$ which does not intersect the set of critical values of the restriction of π_t to $\mathcal{V}_t$. Let $\eta \in \boldsymbol{R}$ such that the open set $]-\eta, 0[\cup]0, \eta[$ does not contain any critical value of the restriction of π_t to the algebraic set $\mathcal{V}_t$. Hence, $\mathcal{V}_e$ is nonsingular for $e \in]-\eta, 0[\cup]0, \eta[$, $(\mathcal{V}_e \cap \boldsymbol{R}^{n+1}) \times (]-\eta, 0[\cup]0, \eta[)$ is diffeomorphic to $\mathcal{V}_{t,]-\eta,0[\cup]0,\eta[}$.

We need to mention that $W(\pi_i, \mathcal{H}_e)$ corresponds to the critical points of π_i restricted to $\mathcal{V}_e$ with non-zero x_{n+1} coordinate. Further, we use $W(\pi_i, \mathcal{V}_e)$ to address those latter critical points.

IBM - Ex. 1002-Part 10 of 10, Page 3412
IPR2025-00379 (IBM v. Croga)

Now, for $1 \leq i \leq n$, we define $\mathcal{W}_i$ as the closure of

$$\left\{(\boldsymbol{x}, t) \in \boldsymbol{R}^{n+2} \mid \tfrac{\partial f}{\partial x_i}(\boldsymbol{x}) \neq 0, \tfrac{\partial f}{\partial x_j}(\boldsymbol{x}) = 0 \text{ for } j \neq i, x_{n+1} \neq 0\right\} \cap \mathcal{V}_t.$$

Since $\boldsymbol{A}$ is assumed to be generically chosen, $\mathcal{W}_i$ is either empty or one-equidimensional (because $\langle \ell \cdot \frac{\partial f}{\partial x_i} - 1, \frac{\partial f}{\partial x_j} \forall j \neq i \rangle$ either defines an empty set or a one-equidimensional algebraic set by [24]). This implies that the set of singular points of $\mathcal{W}_i$ is finite.

By [18], the set of non-properness of the restriction of π_t to $\mathcal{W}_i$ is finite (this is the set of points y such that for any closed interval U containing y, $\pi_t^{-1}(U) \cap \mathcal{W}_i$ is not bounded). Using again [18], the restriction of π_t to $\mathcal{W}_i$ realizes a locally trivial fibration over any connected open subset which does not meet the union of the images by π_t of the singular points of $\mathcal{W}_i$, the set of non-properness, and the set of critical values of the restriction of π_t to $\mathcal{W}_i$. We let η_i' be the minimum of the absolute values of the points in this union.

We choose now $0 < e_0 < \min(\eta, \eta_1', \ldots, \eta_n')$. We call SpecializationValue a subroutine that takes as input f and g and returns such a rational number e_0. Note that SpecializationValue is easily obtained from elimination algorithms solving polynomial systems (from which we can compute critical values) and from [26] to compute the set of non-properness of some map.

With e_0 as above, we denote $\mathcal{I} =]-e_0, 0[\cup]0, e_0[$. Let $\mathcal{W}_{i,\mathcal{I}}$ is semi-algebraically diffeomorphic to $\mathcal{W}_{i,e} \times \mathcal{I}$ for every $e \in \mathcal{I}$. As $\mathcal{V}_e$ is nonsingular, the critical locus $W(\pi_i, \mathcal{V}_e)$ is guaranteed to be finite by the genericity of the change of variables $\boldsymbol{A}$ (hence $\mathcal{W}_{i,e}$ is) and that $W(\pi_i, \mathcal{V}_e) \cap \boldsymbol{R}^{n+1}$ coincides with $\pi_{\boldsymbol{x}}(\mathcal{W}_{i,e})$. Thus, the above diffeomorphism implies that, for any connected component $\mathcal{C}$ of $\mathcal{W}_{i,\mathcal{I}}$, $\mathcal{C}$ is diffeomorphic to an open interval in $\boldsymbol{R}$. Moreover, if $\mathcal{C}$ is bounded, then $\overline{\mathcal{C}} \setminus \mathcal{C}$ contains exactly two points which satisfy respectively $f = 0$ and $f^2 = e_0^2$. We now consider

$$\mathcal{L}_i = \left\{\boldsymbol{x} \in \boldsymbol{R}^{n+1} \mid 0 < f < e_0, g = 0, \tfrac{\partial f}{\partial x_j} = 0 \text{ for} j \neq i, x_{n+1} \neq 0\right\}.$$

It is the intersection of the Zariski closure $\mathcal{K}_i$ of the solution set to $\left\{\frac{\partial f}{\partial x_i} \neq 0, \frac{\partial f}{\partial x_j} = 0 \text{ for } j \neq i, x_{n+1} \neq 0\right\}$ with the semi-algebraic set defined by $0 < f < e_0$. Note that $\mathcal{K}_i$ is either empty or one-equidimensional. As $\mathcal{V}_e$ is nonsingular for $e \in \mathcal{I}$, $\mathcal{L}_i$ and $\mathcal{L}_j$ are disjoint for $i \neq j$. Since the restriction of $\pi_{\boldsymbol{x}}$ to $\mathcal{V}_t$ is an isomorphism between the algebraic sets $\mathcal{V}_t$ and $\boldsymbol{R}^{n+1}$ with the inverse map $\boldsymbol{x} \mapsto (\boldsymbol{x}, f(\boldsymbol{x}))$, the properties of $\mathcal{W}_{\mathcal{I}}$ mentioned above are transferred to its image $\mathcal{L}_i$ by the projection $\pi_{\boldsymbol{x}}$.

Further, we consider a subroutine ParametricCurve which takes as input f and $i \in [1, n]$ and returns a rational parametrization $\mathfrak{K}_i$ of $\mathcal{K}_i$. Also, let Union be a subroutine that takes a family of rational parametrizations $\mathfrak{K}_1, \ldots, \mathfrak{K}_n$ to compute a rational parametrization encoding the union of the algebraic curves defined by the $\mathfrak{K}_i$'s. We denote by $\mathfrak{K}$ the output of Union ; it encodes $\mathcal{K} = \cup_{i=1}^n \mathcal{K}_i$. We refer to [28, Appendix J.2] for these two subroutines.

Lemma 8 below establishes a *well-defined* notion of limit for a point $\boldsymbol{x}_e \in W(\pi_i, \mathcal{V}_e) \cup W(\pi_i, \mathcal{V}_{-e})$ when e tends to 0.

Lemma 8. *Let e_0 and $\mathcal{L}_i$ be as above. For $e \in]0, e_0[$ and $\boldsymbol{x}_e \in (W(\pi_i, \mathcal{V}_e) \cup W(\pi_i, \mathcal{V}_{-e})) \cap \boldsymbol{R}^{n+1}$, there exists a (unique) connected component $\mathcal{C}$ of $\mathcal{L}_i$ containing $\boldsymbol{x}_e$. If $\mathcal{C}$ is bounded, let $\boldsymbol{x}$ be the only point in $\overline{\mathcal{C}}$ satisfying $f(\boldsymbol{x}) = 0$, then $\boldsymbol{x} \in \lim_\varepsilon(W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon})) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$. Thus, we set $\lim_0 \boldsymbol{x}_e = \boldsymbol{x}$.*

Moreover, the extension $\mathrm{ext}(\mathcal{C}, \boldsymbol{R}\langle\varepsilon\rangle)$ contains exactly one point $\boldsymbol{x}_\varepsilon$ such that $f(\boldsymbol{x}_\varepsilon)^2 = \varepsilon^2$ and $\lim_\varepsilon \boldsymbol{x}_\varepsilon = \boldsymbol{x}$.

Proof. Since $\boldsymbol{x}_e \in (W(\pi_i, \mathcal{V}_e) \cup W(\pi_i, \mathcal{V}_{-e})) \cap \boldsymbol{R}^{n+1}$ and $0 < e < e_0$, we have $\boldsymbol{x}_e \in \mathcal{L}_i$, the existence of $\mathcal{C}$ follows naturally. Let $\boldsymbol{x}$ be the unique point of $\overline{\mathcal{C}}$ satisfying $f = 0$. Then, the notion $\lim_0$ is well-defined. From the proof of [2, Theorem 12.43], we have that

$$\lim_\varepsilon(W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon})) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1} = \pi_{\boldsymbol{x}}\left(\overline{\mathcal{W}_{(0,+\infty)}} \cap V(t)\right).$$

As $\pi_{\boldsymbol{x}}\left(\overline{\mathcal{W}_{(0,+\infty)}} \cap V(t)\right)$ is the set of points corresponding to $f = 0$ of $\mathcal{L}_i$, we deduce that $\boldsymbol{x} \in \lim_\varepsilon(W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon})) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$.

Since the extension $\mathrm{ext}(\mathcal{C}, \boldsymbol{R}\langle\varepsilon\rangle)$ is a connected component of $\mathrm{ext}(\mathcal{L}_i, \boldsymbol{R}\langle\varepsilon\rangle)$ and homeomorphic to an open interval in $\boldsymbol{R}\langle\varepsilon\rangle$, there exists $\boldsymbol{x}_\varepsilon \in \mathrm{ext}(\mathcal{C}, \boldsymbol{R}\langle\varepsilon\rangle)$ such that $f(\boldsymbol{x}_\varepsilon)^2 = \varepsilon^2$. Moreover, since $0 = \lim_\varepsilon f(\boldsymbol{x}_\varepsilon)^2 = f(\lim_\varepsilon \boldsymbol{x}_\varepsilon)^2$ and $\boldsymbol{x}$ is the only point in $\overline{\mathcal{C}}$ satisfying $f = 0$, we conclude that $\lim_\varepsilon \boldsymbol{x}_\varepsilon = \boldsymbol{x}$. □

Now, let $\mathcal{R}_e$ be a roadmap associated to the algebraic set $\mathcal{V}_e \cup \mathcal{V}_{-e}$, i.e. $\mathcal{R}_e$ is contained in $(\mathcal{V}_e \cup \mathcal{V}_{-e}) \cap \boldsymbol{R}^{n+1}$, of at most dimension one and has non-empty intersection with every connected component of $(\mathcal{V}_e \cup \mathcal{V}_{-e}) \cap \boldsymbol{R}^{n+1}$. We also require that $\mathcal{R}_e$ contains $\cup_{i=1}^n (W(\pi_i, \mathcal{V}_e) \cup W(\pi_i, \mathcal{V}_{-e})) \cap \boldsymbol{R}^{n+1}$. The proposition below is the key to describe IsIsolated.

Proposition 9. *Given $e \in]0, e_0[$ and $\mathcal{I} =]-e_0, 0[\cup]0, e_0[$ as above. Let $\mathcal{L} = \cup_{i=1}^n \mathcal{L}_i$ and $\boldsymbol{x} \in \mathcal{P}$. Then $\boldsymbol{x}$ is not isolated in $\mathcal{V} \cap \boldsymbol{R}^{n+1}$ if and only if there exists $\boldsymbol{x}' \in \mathcal{P}$ such that $\boldsymbol{x}$ and $\boldsymbol{x}'$ are connected in $\mathcal{P} \cup \mathcal{L} \cup \mathcal{R}_e$.*

Proof. Assume first that $\boldsymbol{x}$ is not isolated. By Proposition 7, there exists $1 \leq i \leq n$ and a connected component $\mathcal{C}_\varepsilon$ of $(\mathcal{V}_\varepsilon \cup \mathcal{V}_{-\varepsilon}) \cap \boldsymbol{R}\langle\varepsilon\rangle^{n+1}$, which is bounded over $\boldsymbol{R}$, such that $\mathcal{C}_\varepsilon \cap (W(\pi_i, \mathcal{V}_\varepsilon) \cup W(\pi_i, \mathcal{V}_{-\varepsilon}))$ contains $\boldsymbol{x}_\varepsilon$ and $\boldsymbol{x}'_\varepsilon$ satisfying $\boldsymbol{x} = \lim_\varepsilon \boldsymbol{x}_\varepsilon \neq \lim_\varepsilon \boldsymbol{x}'_\varepsilon$. By the choice of e_0, there exist a diffeomorphism $\theta : \mathcal{V}_{t,\mathcal{I}} \to \mathcal{V}_e \times \mathcal{I}$ such that $\theta(\mathcal{W}_{i,\mathcal{I}}) = \theta(\mathcal{W}_{i,e}) \times \mathcal{I}$. Using [2, Exercise 3.2], $\mathrm{ext}(\theta, \boldsymbol{R}\langle\varepsilon\rangle)$ is a diffeomorphism between:

$$\mathrm{ext}(\mathcal{V}_{t,\mathcal{I}}, \boldsymbol{R}\langle\varepsilon\rangle) \cong \mathrm{ext}(\mathcal{V}_e, \boldsymbol{R}\langle\varepsilon\rangle) \times \mathrm{ext}(\mathcal{I}, \boldsymbol{R}\langle\varepsilon\rangle),$$

IBM - Ex. 1002-Part 10 of 10, Page 3413
IPR2025-00379 (IBM v. Croga)

$$\mathrm{ext}(\mathcal{W}_{i,\mathcal{I}}, \boldsymbol{R}\langle\varepsilon\rangle) \cong \mathrm{ext}(\mathcal{W}_{i,e}, \boldsymbol{R}\langle\varepsilon\rangle) \times \mathrm{ext}(\mathcal{I}, \boldsymbol{R}\langle\varepsilon\rangle).$$

As $\pi_{\boldsymbol{x}}$ is an isomorphism from $\mathcal{V}_t$ to $\boldsymbol{R}^{n+1}$, there exists a (unique) bounded connected component $\mathcal{C}_e$ of $\mathcal{V}_e \cap \boldsymbol{R}^{n+1}$ s.t. $\mathcal{C}_\varepsilon$ is diffeomorphic to $\mathrm{ext}(\mathcal{C}_e, \boldsymbol{R}\langle\varepsilon\rangle)$. Moreover, let L and L' be the connected components of $\mathrm{ext}(\mathcal{L}_i, \boldsymbol{R}\langle\varepsilon\rangle)$ containing $\boldsymbol{x}_\varepsilon$ and $\boldsymbol{x}'_\varepsilon$ respectively and $\boldsymbol{x}_e$ and $\boldsymbol{x}'_e$ ($\in \mathrm{ext}(\mathcal{C}_e, \boldsymbol{R}\langle\varepsilon\rangle)$) be the intersections of $\mathrm{ext}(\mathcal{C}_e, \boldsymbol{R}\langle\varepsilon\rangle)$ with L and L' respectively. Then, $\lim_\varepsilon \boldsymbol{x}_\varepsilon$ ($\lim_\varepsilon L'$) connects $\lim_\varepsilon \boldsymbol{x}_e$ ($\lim_\varepsilon \boldsymbol{x}'_e$) to $\boldsymbol{x}$ ($\boldsymbol{x}'$). As $\lim_\varepsilon \boldsymbol{x}_e$ and $\lim_\varepsilon \boldsymbol{x}'_e$ are connected in $\mathcal{C}_e$, we conclude that $\boldsymbol{x}$ and $\boldsymbol{x}'$ are also connected in $\mathcal{P} \cup \mathcal{L} \cup \mathcal{R}_e$. The reverse implication is immediate using the above techniques □

From Lemma 8 and Proposition 9, any e lying in the interval $]0, e_0[$ defined above can be used to replace the infinitesimal ε. So, we simply take $e = e_0/2$. For $1 \leq i \leq n$, we use a subroutine ZeroDimSolve which takes as input $\left\{f - e_0/2, g, \frac{\partial f}{\partial x_j} \text{ for all } j \neq i\right\}$ to compute a zero-dimensional parametrization $\mathfrak{Q}_i$ such that $W(\pi_i, \mathcal{V}_e) = \{\boldsymbol{x} \in Z(\mathfrak{Q}_i) | x_{n+1} \neq 0\}$.

To use Proposition 9, we need to compute $\mathcal{R}_{e_0/2}$, which we refer to the algorithm Roadmap in [28]. This algorithm allows us to compute roadmaps for smooth and bounded real algebraic sets, which is indeed the case of $(\mathcal{V}_{e_0/2} \cup \mathcal{V}_{-e_0/2}) \cap \boldsymbol{R}^{n+1}$. First, we call (another) Union that, on the zero-dimensional parametrizations $\mathfrak{Q}_i$, it computes a zero-dimensional parametrization $\mathfrak{Q}$ encoding $\cup_{i=1}^n Z(\mathfrak{Q}_i)$. Given the polynomials f, g, the value $e_0/2$ and the parametrization $\mathfrak{Q}$, a combination of Union and Roadmap returns a one-dimensional parametrization $\mathfrak{R}$ representing $\mathcal{R}_{e_0/2}$.

Deciding connectivity over $\mathcal{P} \cup \mathcal{L} \cup \mathcal{R}_e$ is done as follows. We use Union to compute a rational parametrization $\mathfrak{S}$ encoding $\mathcal{K} \cup \mathcal{R}_e$. Then, with input $\mathfrak{S}$, $\mathfrak{P}$, $x_{n+1} \neq 0$ and the inequalities $0 < f < e_0$, we use Newton Puiseux expansions and cylindrical algebraic decomposition (see [10, 30]) following [27], taking advantage of the fact that polynomials involved in rational parametrizations of algebraic curves are bivariate. We denote by ConnectivityQuery the subroutine that takes those inputs and returns $\mathfrak{P}$ and isolating boxes of the points defined by $\mathfrak{P}$ which are not connected to other points of $\mathfrak{P}$.

Algorithm 3: IsIsolated

Input: The polynomials $f^{\boldsymbol{A}} \in \boldsymbol{Q}[x_1, \ldots, x_n]$ and $g \in \boldsymbol{Q}[x_1, \ldots, x_{n+1}]$ and the zero-dimensional parametrization $\mathfrak{P}$.

Output: $\mathfrak{P}$ with isolating boxes of the isolated points of $\mathcal{V}^{\boldsymbol{A}} \cap \boldsymbol{R}^{n+1}$

$e_0 \leftarrow$ SpecializationValue$(f^{\boldsymbol{A}}, g)$
for $1 \leq i \leq n$ **do**
  $\mathfrak{Q}_i \leftarrow$ ZeroDimSolve $\left(\left\{f^{\boldsymbol{A}} - e_0/2, g, \frac{\partial f^{\boldsymbol{A}}}{\partial x_j} \text{ for all } j \neq i\right\}\right)$
  $\mathfrak{K}_i \leftarrow$ ParametricCurve$(f^{\boldsymbol{A}}, i)$
end
$\mathfrak{K} \leftarrow$ Union$(\mathfrak{K}_1, \ldots, \mathfrak{K}_n)$
$\mathfrak{Q} \leftarrow$ Union$(\mathfrak{Q}_1, \ldots, \mathfrak{Q}_n)$
$\mathfrak{R} \leftarrow$ Union(RoadMap$(f^{\boldsymbol{A}} - e_0/2, g, \mathfrak{Q})$, RoadMap$(f^{\boldsymbol{A}} + e_0/2, g, \mathfrak{Q})$)
$\mathfrak{S} \leftarrow$ Union$(\mathfrak{K}, \mathfrak{R})$
$B \leftarrow$ ConnectivityQuery$(\mathfrak{S}, \mathfrak{P}, x_{n+1} \neq 0, 0 < f^{\boldsymbol{A}} < e_0)$
return $(\mathfrak{P}, B)$

4 Complexity analysis

All complexity results are given in the number of arithmetic operations in $\boldsymbol{Q}$. Hereafter, we assume that a generic enough matrix $\boldsymbol{A}$ is found from a random choice. In order to end the proof of Theorem 1, we now estimate the arithmetic runtime of the calls to Candidates and IsIsolated.

Complexity of Algorithm 2 Since $W(\pi_i, \mathcal{H}_\varepsilon^{\boldsymbol{A}})$ is the finite algebraic set associated to $\left\langle f^{\boldsymbol{A}} - e, \frac{\partial f^{\boldsymbol{A}}}{\partial x_j} \text{ for all } j \neq i\right\rangle$, its degree is bounded by $d(d-1)^n$ [17]. Consequently, the degree of the output zero-dimensional parametrization lies in $d^{O(n)}$. Using [24, Theorem 6] (which is based on the geometric resolution algorithm in [14]), it is computed within $d^{O(n)}$ arithmetic operations in $\boldsymbol{Q}$. The last step which takes intersections is done using the algorithm in [28, Appendix J.1] ; it does not change the asymptotic complexity.

We have seen that GetNormBound reduces to isolate the real roots of a zero-dimensional parametrization of degree $d^{O(n)}$. This can be done within $d^{O(n)}$ operations by Uspensky's algorithm [23].

Complexity of Algorithm 3 Each call to SpecializationValue reduces to computing critical values of π_i of a smooth algebraic set defined by polynomials of degree $\leq d$. This is done using $(nd)^{O(n)}$ arithmetic operations in $\boldsymbol{Q}$ (see [15]). Using [14] for

IBM - Ex. 1002-Part 10 of 10, Page 3414
IPR2025-00379 (IBM v. Croga)

ZeroDimSolve and [29] for ParametricCurve does not increase the overall complexity. The loop is performed n times ; hence the complexity lies in $(nd)^{O(n)}$. All output zero-dimensional parametrizations have degree bounded by $d^{O(n)}$. Running Union on these parametrizations does not increase the asymptotic complexity. One gets then parametrizations of degree bounded by $nd^{O(n)}$. Finally, using [28] for Roadmap uses $(nd)^{O(n\log(n))}$ arithmetic operations in $\boldsymbol{Q}$ and outputs a rational parametrization of degree lying in $(nd)^{O(n\log(n))}$. The call to ConnectivityQuery, done as explained in [27] is polynomial in the degree of the roadmap.

The final steps which consist in calling Removes and undoing the change of variables does not change the asymptotic complexity.

Summing up altogether the above complexity estimates, one obtains an algorithm using $(nd)^{O(n\log(n))}$ arithmetics operations in $\boldsymbol{Q}$ at most. This ends the proof of Theorem 1.

5 Experimental results

We report on practical performances of our algorithm. Computations were done on an Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz with 32GB of RAM. We take sums of squares of n random dense quadrics in n variables (with a non-empty intersection over $\boldsymbol{R}$) ; we obtain *dense quartics* defining a finite set of points. Timings are given in seconds (s.), minutes (m.), hours (h.) and days (d.).

We used Faugère's FGB library for computing Gröbner bases in order to perform algebraic elimination in Algorithms 1, 2 and 3. We also used our C implementation for bivariate polynomial system solving (based on resultant computations) which we need to analyze connectivity queries in roadmaps. Timings for Algorithm 2 are given in the column CAND below. Timings for the computation of the roadmaps are given in the column RMP and timings for the analysis of connectivity queries are given in the column QRI below.

Roadmaps are obtained as the union of critical loci of some maps in slices of the input variety [28]. We report on the highest degree of these critical loci in the column SRMP. The column SQRI reports on the maximum degree of the bivariate zero-dimensional system we need to study to analyze connectivity queries on the roadmap.

None of the examples we considered could be tackled using the implementations of Cylindrical Algebraic Decomposition algorithms in Maple and Mathematica.

We also implemented [2, Alg. 12.16] using the FLINT C library with evaluation/interpolation techniques instead to tackle coefficients involving infinitesimals. This algorithm only computes sample points per connected components. *That implementation was not able to compute sample points of the input quartics for any of our examples*. We then report in the column [BPR] on the degree of the zero-dimensional system which is expected to be solved by [2, BPR]. This is to be compared with columns SRMP and SQRI.

n	CAND	RMP	QRI	total	SRMP	SQRI	[BPR]
4	2 s.	15 s.	33 s.	50 s.	36	359	7290
5	$<$ 10 min.	1h.	7h.	8 h.	108	4644	65 610
6	$< 12h$	2 d.	18 d.	20 d.	308	47952	590 490

References

[1] ARNON, D. S. A cluster-based cylindrical algebraic decomposition algorithm. *J. Symb. Comput. 5*, 1/2 (1988), 189–212.

[2] BASU, S., POLLACK, R., AND ROY, M.-F. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, 2006.

[3] BASU, S., AND ROY, M. Divide and conquer roadmap for algebraic sets. *Discrete & Computational Geometry 52*, 2 (2014), 278–343.

[4] BASU, S., ROY, M., SAFEY EL DIN, M., AND SCHOST, É. A baby step-giant step roadmap algorithm for general algebraic sets. *Foundations of Computational Mathematics 14*, 6 (2014), 1117–1172.

[5] BORCEA, C., AND STREINU, I. Geometric auxetics. *Proc. R. Soc. Lond., A, Math. Phys. Eng. Sci. 471*, 2184 (2015), 24.

[6] BORCEA, C. S., AND STREINU, I. Periodic auxetics: structure and design. *Q. J. Mech. Appl. Math. 71*, 2 (2018), 125–138.

[7] COLLINS, G. E. Quantifier elimination for real closed fields by cylindrical algebraic decomposition: a synopsis. *ACM SIGSAM Bulletin 10*, 1 (1976), 10–12.

[8] CONNELLY, R., AND WHITELEY, W. The stability of tensegrity frameworks. *International Journal of Space Structures 7*, 2 (1992), 153–163.

[9] COSTE, M., AND SHIOTA, M. Thom's first isotopy lemma: a semialgebraic version, with uniform bounds. *RIMS Kokyuroku 815* (1992), 176–189.

[10] DUVAL, D. Rational puiseux expansions. *Compositio Mathematica 70*, 2 (1989), 119–154.

[11] GASPAR, N., REN, X., SMITH, C., GRIMA, J., AND EVANS, K. Novel honeycombs with auxetic behaviour. *Acta Materialia 53*, 8 (2005), 2439 – 2445.

[12] GIANNI, P. M., AND MORA, T. Algebraic solution of systems of polynomial equations using Gröebner bases. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 5th International Conference, AAECC-5, Menorca, Spain, Proceedings* (1987), pp. 247–257.

IBM - Ex. 1002-Part 10 of 10, Page 3415
IPR2025-00379 (IBM v. Croga)

[13] GIUSTI, M., HEINTZ, J., MORAIS, J. E., AND PARDO, L. M. When polynomial equation systems can be "solved" fast? In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 11th International Symposium, AAECC-11, Paris, France, Proceedings* (1995), pp. 205–231.

[14] GIUSTI, M., LECERF, G., AND SALVY, B. A Gröbner free alternative for polynomial system solving. *Journal of Complexity 17*, 1 (2001), 154 – 211.

[15] GREUET, A., AND SAFEY EL DIN, M. Probabilistic algorithm for polynomial optimization over a real algebraic set. *SIAM Journal on Optimization 24*, 3 (2014).

[16] GRIMA, J. N., AND EVANS, K. E. Auxetic behavior from rotating triangles. *Journal of Materials Science 41*, 10 (May 2006), 3193–3196.

[17] HEINTZ, J. Definability and fast quantifier elimination in algebraically closed fields. *Theor. Comput. Sci. 24* (1983), 239–277.

[18] JELONEK, Z., AND KURDYKA, K. Quantitative generalized Bertini-Sard theorem for smooth affine varieties. *Discrete & Computational Geometry 34*, 4 (2005), 659–678.

[19] KRONECKER, L. Grundzüge einer arithmetischen theorie der algebraischen grössen. *Journal für die reine und angewandte Mathematik 92* (1882), 1–122.

[20] LAKES, R. Foam structures with a negative poisson's ratio. *Science 235*, 4792 (1987), 1038–1040.

[21] ROTH, B., AND WHITELEY, W. Tensegrity frameworks. *Trans. Am. Math. Soc. 265* (1981), 419–446.

[22] ROUILLIER, F., ROY, M., AND SAFEY EL DIN, M. Finding at least one point in each connected component of a real algebraic set defined by a single equation. *J. Complexity 16*, 4 (2000), 716–750.

[23] ROUILLIER, F., AND ZIMMERMANN, P. Efficient isolation of polynomial's real roots. *Journal of Computational and Applied Mathematics 162*, 1 (2004), 33–50.

[24] SAFEY EL DIN, M. Computing Sampling Points on a Singular Real Hypersurface using Lagrange's System. Research Report RR-5464, INRIA, 2005.

[25] SAFEY EL DIN, M., AND SCHOST, E. Polar varieties and computation of one point in each connected component of a smooth real algebraic set. In *Proc. of the 2003 Int. Symp. on Symb. and Alg. Comp.* (2003), ISSAC '03, ACM, p. 224–231.

[26] SAFEY EL DIN, M., AND SCHOST, É. Properness defects of projections and computation of at least one point in each connected component of a real algebraic set. *Discrete & Computational Geometry 32*, 3 (2004), 417–430.

[27] SAFEY EL DIN, M., AND SCHOST, É. A baby steps/giant steps probabilistic algorithm for computing roadmaps in smooth bounded real hypersurface. *Discrete & Computational Geometry 45*, 1 (2011), 181–220.

[28] SAFEY EL DIN, M., AND SCHOST, É. A nearly optimal algorithm for deciding connectivity queries in smooth and bounded real algebraic sets. *J. ACM 63*, 6 (Jan. 2017), 48:1–48:37.

[29] SCHOST, É. Computing parametric geometric resolutions. *Applicable Algebra in Engineering, Communication and Computing 13*, 5 (2003), 349–393.

[30] SCHWARTZ, J. T., AND SHARIR, M. On the "piano movers" problem. II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics 4*, 3 (1983), 298 – 351.

[31] VOROBJOV, N. Complexity of computing the local dimension of a semialgebraic set. *Journal of Symbolic Computation 27*, 6 (1999), 565–579.

[32] YANG, W., LI, Z.-M., SHI, W., AND XIE, B.-H. Review on auxetic materials. *Journal of Materials Science 39* (2004), 3269–3279.

IBM - Ex. 1002-Part 10 of 10, Page 3416
IPR2025-00379 (IBM v. Croga)

Thwarting Memory Disclosure with Efficient Hypervisor-enforced Intra-domain Isolation

Yutao Liu, Tianyu Zhou, Kexin Chen, Haibo Chen, Yubin Xia
Shanghai Key Laboratory of Scalable Computing and Systems &
Institute of Parallel and Distributed Systems
Shanghai Jiao Tong University, Shanghai, China
{ytliu.cc, zhoutianyu007, kalzium, haibochen, xiayubin}@sjtu.edu.cn

ABSTRACT

Exploiting memory disclosure vulnerabilities like the HeartBleed bug may cause arbitrary reading of a victim's memory, leading to leakage of critical secrets such as crypto keys, personal identity and financial information. While isolating code that manipulates critical secrets into an isolated execution environment is a promising countermeasure, existing approaches are either too coarse-grained to prevent intra-domain attacks, or require excessive intervention from low-level software (e.g., hypervisor or OS), or both. Further, few of them are applicable to large-scale software with millions of lines of code.

This paper describes a new approach, namely SeCage, which retrofits commodity hardware virtualization extensions to support efficient isolation of sensitive code manipulating critical secrets from the remaining code. SeCage is designed to work under a strong adversary model where a victim application or even the OS may be controlled by the adversary, while supporting large-scale software with small deployment cost. SeCage combines static and dynamic analysis to decompose monolithic software into several compartments, each of which may contain different secrets and their corresponding code. Following the idea of separating control and data plane, SeCage retrofits the VMFUNC mechanism and nested paging in Intel processors to transparently provide different memory views for different compartments, while allowing low-cost and transparent invocation across domains without hypervisor intervention.

We have implemented SeCage in KVM on a commodity Intel machine. To demonstrate the effectiveness of SeCage, we deploy it to the Nginx and OpenSSH server with the OpenSSL library as well as CryptoLoop with small efforts. Security evaluation shows that SeCage can prevent the disclosure of private keys from HeartBleed attacks and memory scanning from rootkits. The evaluation shows that SeCage only incurs small performance and space overhead.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
CCS'15, October 12–16, 2015, Denver, Colorado, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813690 .

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access control, information flow controls*

General Terms

Security

Keywords

Privacy protection, memory disclosure, virtualization

1. INTRODUCTION

Problem. Cloud servers are increasingly being deployed with services that touch critical *secrets* such as cryptographic keys, personal identity and financial information. Yet, such *secrets* are being continually disclosed due to memory disclosure attacks. For example, HeartBleed (CVE-2014-0160) [3, 23], as one of the most notorious vulnerabilities, can be leveraged by attackers to persistently read up to 64KB memory data. This leads to the leakage of the most confidential *secrets* such as private keys or session keys of all connections.

Actually, memory disclosure vulnerabilities are routinely discovered and pose severe threats to the privacy of *secrets*. Table 1 shows that there are 388 such vulnerabilities from the CVE database according to the triggering mechanisms for both applications and kernels, which can be exploited by local (L column) or remote (R column) attacks. Worse even, leakage of *secrets* can be directly done if an attacker gained control over a privileged application or even the OS kernel through other vulnerabilities such as privilege escalation.

Facing such a strong adversary model, it is notoriously hard to prevent attackers from controlling victim applications or even the OS kernel from reading such memory-resident *secrets*. This is further being exaggerated due to the fact that large-scale software usually has a large code base and thus large attack surfaces. While there have been a number of approaches aiming at protecting an application from various attacks, they still fall short in several aspects. For example, hypervisor-based protection schemes [16, 17, 18, 53, 19] only provide protection at an application level and thus vulnerabilities still exist inside a victim application; approaches targeting at Pieces of Application Logic (PAL) [34] require the PAL being self-contained and thus allowing no interaction with other parts of an application. This makes it hard to be adopted for some real and large

IBM - Ex. 1002-Part 10 of 10, Page 3417
IPR2025-00379 (IBM v. Croga)

software like OpenSSL[1]. Further, due to the close-coupling of security and functionality, prior approaches usually require frequent intervention from a privileged system (e.g., hypervisor), forcing users to make a tradeoff between security and performance.

Table 1: Sensitive memory disclosure vulnerabilities from CVE (2000-2015)

Error type	Application		Kernel		Summary
	L	R	L	R	
Uninitialized memory	1	34	104	8	**147**
Out-of-bound read	4	37	20	4	**65**
Others	3	28	16	7	**54**
Use-after-free vuln	14	7	6	4	**31**
Permission uncheck	4	14	10	0	**28**
Bad error handling	1	12	8	0	**21**
Plaintext in memory	10	2	0	0	**12**
Bad null terminator	0	3	6	0	**9**
Invalid pointer	1	0	6	0	**7**
Off-by-one vuln	1	5	0	0	**6**
String format vuln	0	4	0	0	**4**
Leak to readable file	3	0	1	0	**4**
Total	**42**	**146**	**177**	**23**	**388**

Our solution. In this paper, we leverage the idea of privilege separation by hybrid analysis to (mostly) automatically decompose a monolithic software system into a set of compartments, with each *secret* compartment containing a set of *secrets* and its corresponding code and a main compartment handling the rest of the application logic. This ensures that only the functions inside a *secret* compartment can access the *secrets*. As static analysis is usually imprecise and may introduce large code base in the *secret* compartment, SeCage further combines dynamic analysis to extract the most commonly used functions. To handle possible coverage issues, SeCage resorts to runtime exception handling to detect if an access is legal or not based on the static analysis result and runtime information.

SeCage leverages hardware virtualization techniques to enforce strong isolation among *secret* compartments and the main compartment, even under a strong adversary model such that an application or even the OS is controlled by an attacker. Specifically, SeCage assigns each compartment a completely isolated address space and leverages hardware-assisted nested paging to enforce strong isolation. To provide secure and efficient communication among components, SeCage follows the idea of separating control plane from data plane by using the VMFUNC feature from Intel's hardware-assisted virtualization support. In particular, SeCage first designates the security policy on which *secret* compartment is invokable by another compartment to the CPU, and allows such an invocation from being done without the hypervisor intervention. This significantly reduces the overhead caused by frequent trapping into the hypervisor.

Evaluation on real-world applications. We have implemented a prototype of SeCage in KVM using Intel's hardware virtualization support; we use CIL [1], an analysis framework of C programming language, to (mostly) automatically decompose a software system into a set of compartments. We further use a dynamic analysis engine to refine the results. To demonstrate the effectiveness of SeCage, we apply SeCage to Nginx and OpenSSH server with the OpenSSL library, as well as the Linux kernel's disk encryption module CryptoLoop to demonstrate its effectiveness. We use HeartBleed, kernel memory disclosure and rootkit defenses as examples to illustrate how SeCage protects user-defined *secrets*. Our evaluation shows that SeCage only introduces negligible performance overhead, since the portion of *secrets* related code is small and there are very few hypervisor interventions at runtime.

Contributions. In summary, this paper makes the following contributions:

- A new scheme to protect *secrets* from extensive attack surfaces by compartmentalizing critical code and data from normal ones.
- A separation of control plane (policy) and data plane (invocation) for cross-compartment communication by leveraging commodity hardware features (VMFUNC).
- A working prototype implemented on KVM and its application to large-scale software such as Nginx and OpenSSH with the OpenSSL library and the CryptoLoop module in Linux kernel, as well as security and performance evaluations that confirm the effectiveness and efficiency of SeCage.

The rest of the paper is organized as follows: the next section first illustrates the architecture overview of SeCage and the threat model. Section 3 and section 4 illustrate the design and detailed implementation of the runtime isolation and application decomposition parts of SeCage, followed by the usage scenarios in section 5. Then the security evaluation of SeCage and its incurred performance overhead are evaluated in section 6 and section 7 accordingly. Finally, we review and compare SeCage with state-of-the-art in section 8, discuss the limitation of SeCage in section 9 and conclude the paper in section 10.

2. OVERVIEW

The primary goal of SeCage is to offer the strong assurance of confidentiality for user-specific *secrets* (e.g., private keys), even facing a vulnerable application or a malicious OS. The secondary goal of SeCage is to make the approach of SeCage practical and thus can be deployed for large software systems with small overhead.

2.1 Approach Overview

Hybrid analysis to extract *secret* closure. As *secrets* may be copied and propagated through its lifecycle, it is far from enough to simply secure the storage of *secrets*. Instead, SeCage must provide a thorough mechanism to prevent *secrets* and their provenance from being disclosed during the whole application execution, while not affecting the normal usage of the *secrets*. Hence, SeCage needs to find a closure of all functions that may manipulate the *secrets*.

One intuitive approach is using static analysis to discover the closure of code. However, the static analysis still has precision issues for large-scale software written in C/C++ due to issues such as pointer aliasing. This may easily lead to a significantly larger closure than necessary, which may enlarge the code base of the *secret* compartment and add

[1]For example, the authors of TrustVisor [34] explicitly acknowledged that "We faced the greatest porting challenge with Apache + OpenSSL ...This proved to be difficult due to OpenSSL's extensive use of function pointers and adaptability to different cryptographic providers and instead resort to PolarSSL" in section 6.4.

IBM - Ex. 1002-Part 10 of 10, Page 3418
IPR2025-00379 (IBM v. Croga)

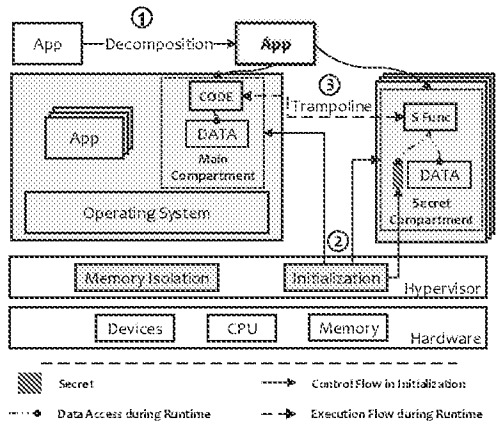


Figure 1: Architecture overview of SeCage

a large number of unnecessary context switches. Another approach would be rewriting the code related to *secrets* and decoupling the operations of *secrets* into a standalone service [13, 34], or even to a trusted third-party node. However, this may involve high manual effort and can be prohibitively difficult for large-scale software such as OpenSSL [34].

SeCage instead combines static and dynamic analysis to extract the closures of functions related to *secrets*. It first uses static analysis to discover potential functions related to *secrets*. To reduce the size of *secret* closure, it then relies on dynamic analysis using a set of training input to derive a compact and precise set of functions related to *secrets*. To handle coverage issue that a function may be legal to touch the *secrets* but is excluded in the *secret* compartment, SeCage adaptively includes this function to the *secret* compartment according to the static analysis result and execution context during runtime exception handling.

Hypervisor-enforced protection. Facing the strong adversary model of a malicious OS, SeCage leverages a trusted hypervisor to protect the privacy of *secrets*. Specifically, SeCage runs the closure of a particular *secret* into a separate compartment and leverages hardware virtualization support to provide strong isolation among different compartments.

Separating control and data plane. As each compartment still needs to communicate with each other, it seems inevitable that the hypervisor will need to frequently intervenes such communications. This, however, will cause frequent VMExits and thus high overhead. To mitigate such overhead, SeCage leverages the idea of separating control and data plane to minimize hypervisor intervention. Specifically, SeCage only requires the hypervisor to define policies on whether a call between two compartments are legal (control plane), while letting the communications between two compartments go as long as they conform to the predefined policies (data plane). In the entry gate of each compartment, it may do further check of the caller to see if the communication should be allowed or not. SeCage achieves such a scheme by leveraging the commodity hardware features called VM functions (section 3.1).

Architecture overview. An overview of SeCage's architecture is shown in Figure 1. The protected application is divided into one main compartment and a set of *secret* compartments. Each *secret* compartment comprises a set of *secrets* and the corresponding sensitive functions manipulating them. We do not assume that the compartment is self-contained, the functions inside are able to interact with the main compartment of the application. However, SeCage guarantees that the *secret* in one compartment cannot be accessed by other compartments of the same application and the underlying software.

Once the compartments are generated (step ①), during the application initialization phase (step ②), the hypervisor is responsible to setup one isolated memory for each compartment, and to guarantee that the *secrets* can only be accessed by the functions in the corresponding compartment. During runtime, *secret* compartments are confined to interact with the main compartment through a trampoline mechanism (step ③) without trapping to the hypervisor. Only when a function outside of a *secret* compartment, e.g., the main compartment, tries to access the *secrets*, the hypervisor will be notified to handle such a violation.

2.2 Threat Model and Assumptions

SeCage aims at protecting critical *secrets* from both vulnerable applications and malicious operating systems (which may also collude).

For vulnerable applications, we consider an adversary with the ability to block, inject, or modify network traffic, so that she can conduct all of the well-known attacks in order to illegally access any data located in memory space of the vulnerable application. Specifically, the adversary can exploit buffer over-read attack as in HeartBleed bug [3], or try to use sophisticated control flow hijacking attacks [40, 11, 12], to invalidate access-control policy or bypass permission check, and read sensitive *secrets* located in the same address space.

The underlying system software (e.g., OS) is untrusted that they can behave in arbitrarily malicious ways to subvert application into disclosing its *secrets*. We share this kind of attacker model with other related systems [18, 17, 28]. Additionally, SeCage also considers the Iago attack [15], where the malicious OS can cause application to harm itself by manipulating return value of system services (e.g., system call), as well as rollback attack [51, 41, 52], where the privileged software can rollback the applications' critical states by forcing memory snapshot rollback.

SeCage assumes that the protected *secrets* should only be used within an application, the functions inside the *secret* compartment won't voluntarily send them out. This is usually true for commodity software like OpenSSL as the software itself is designed to keep such *secrets*. Even if not, this can be detected during the static and dynamic phase of SeCage when generating *secret* compartments. Further, SeCage makes no attempt to prevent against DoS attack which is not aimed at disclosing data. It does not try to protect against side-channel attacks [38, 55], as well as the implicit flow[39] attack which consists in leakage of information through the program control flow, since they are typically hard to deploy and have very limited bandwidth to leak *secrets* in our case. Finally, SeCage does not consider the availability of application in the face of a hostile OS.

3. RUNTIME ISOLATION ENFORCEMENT

In this section, we will introduce how to enforce SeCage protection during application runtime, including memory protection, mechanisms of runtime execution flow and other aspects.

IBM - Ex. 1002-Part 10 of 10, Page 3419
IPR2025-00379 (IBM v. Croga)

3.1 Memory Protection

In SeCage, compartment isolation is guaranteed by two-dimensional paging[2] mechanism. In general, a guest VM can only see the mapping of guest virtual address (GVA) to guest physical address (GPA), while the hypervisor maintains one lower-level extended page table (EPT) for each guest, the EPT maps GPA to the host physical address (HPA).

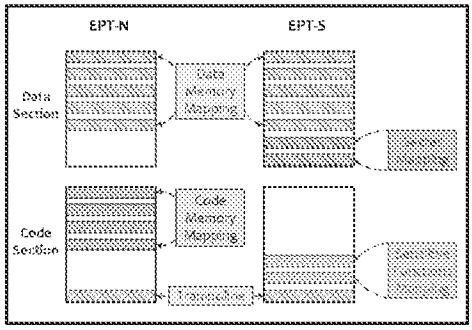


Figure 2: EPT layout of SeCage-enabled application and one of its *secret* compartments

In the initialization phase of SeCage, besides the original EPT called *EPT-N* for the entire guest VM, the hypervisor initializes another EPT, called *EPT-S*, for each protected *secret* compartment. As shown in Figure 2, SeCage classifies the memory into two parts: data and code. For the data section, *EPT-S* maps all data including the *secrets*, while *EPT-N* has data other than the *secrets*. For the code section, the trampoline code is mapped into these two EPTs as read-only. Besides, *EPT-S* only contains the sensitive functions code in the *secret* compartment, while *EPT-N* maps code other than the sensitive functions.

Through the above EPT configuration, SeCage ensures that *secrets* will never exist in *EPT-N*, only code in sensitive functions can access the corresponding *secrets*. These code pages are verified in the setup phase, and the EPT entries are set as executable and read-only. Meanwhile, the data pages in *EPT-S* are set to non-executable so that they cannot be used to inject code by attackers. Therefore, both a vulnerable application and the malicious OS have no means to access the *secrets*. It should be noted that if we only put *secrets* in the *secret* compartment, there may be excessive context switches since the sensitive functions may access other data memory besides *secrets*. For simplicity, since the code pieces of sensitive functions are very small and considered to be trusted in our threat model, SeCage maps the whole data sections into the *secret* compartment.

EPTP switching. SeCage leverages the Intel hardware virtualization extension called VMFUNC, which provides *VM Functions* for non-root guest VMs to directly invoke without VMExit. EPTP switching is one of these VM functions, which allows software (in both kernel and user mode) in guest VM to directly load a new EPT pointer (EPTP), thereby establishing a different EPT paging-structure hierarchy. The EPTP can only be selected from a list of potential EPTP values configured in advance by the hypervisor, which acts as the control plane defining the rules that a guest VM should comply with. During runtime, the hypervisor will not disturb the execution flow within a guest VM.

[2]called *EPT* in Intel, and *NPT* in AMD; we use EPT in this paper.

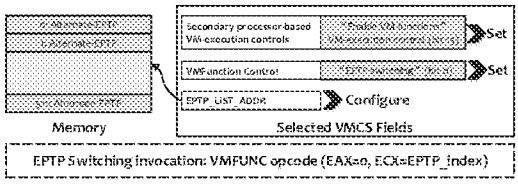


Figure 3: Description of EPTP switching VMFUNC

Figure 3 shows an example configuration that a hypervisor needs to set in order to use EPTP switching VM function: besides some function-enable bits, the hypervisor needs to set bit 0 (EPTP switching bit) in the *VM Function Control* VMCS field, and store the configured EPT pointers to the memory pointed by the *EPTP_LIST_ADDR* VMCS field. During runtime, the non-root software invokes the VMFUNC instruction with *EAX* setting to 0 to trigger the EPTP switching VM function and *ECX* to select an entry from the EPTP list. Currently, EPTP switching supports at most 512 EPTP entries, which means SeCage can support up to 512 compartments for each guest VM.

3.2 Securing Execution Flow

SeCage divides the logic into sensitive functions, trampoline and other code (including application code and system software code). Only sensitive functions in the *secret* compartment can access *secrets*, and the trampoline code is used to switch between the *secret* and the main compartment. During runtime, functions in the main compartment may invoke sensitive functions, while sensitive functions may also call functions outside of the *secret* compartment.

Figure 4 shows possible execution flow during runtime. For clarity, we classify trampoline code into *trampoline* and *springboard*, according to the calling direction. *trampoline* invocation is illustrated in the top half of Figure 4: when code in the main compartment invokes a function in a *secret* compartment, instead of directly calling the sensitive function, it calls into the corresponding trampoline code, which at first executes the VMFUNC instruction to load memory of the *secret* compartment, then the stack pointer is modified to point to the secure stack page. If the number of parameters is larger than six, which is the maximum number of parameters passing supported using register, the rest of parameters should be copied to the secure stack. When everything is ready, it calls the real sensitive function. Once this function returns, the *trampoline* code wipes out the content of secure stack, restores the ESP to the previous stack frame location, reversely executes the VMFUNC instruction and returns the result back to the caller.

During the execution of sensitive functions, it may call functions in the main compartment, e.g., library calls, system calls, and other non-sensitive functions in the application. SeCage classifies these calls into two categories: the calls without *secrets* involved, and the calls which may access the *secrets*. SeCage instruments the first category of calls with *springboard* code, as shown in the bottom half of Figure 4, which just reverses the operations as *trampoline* code. For the second kind of calls, if the callee function does not exist in sensitive functions like some library calls (e.g., memcpy, strlen, etc.), SeCage creates its own version

IBM - Ex. 1002-Part 10 of 10, Page 3420
IPR2025-00379 (IBM v. Croga)

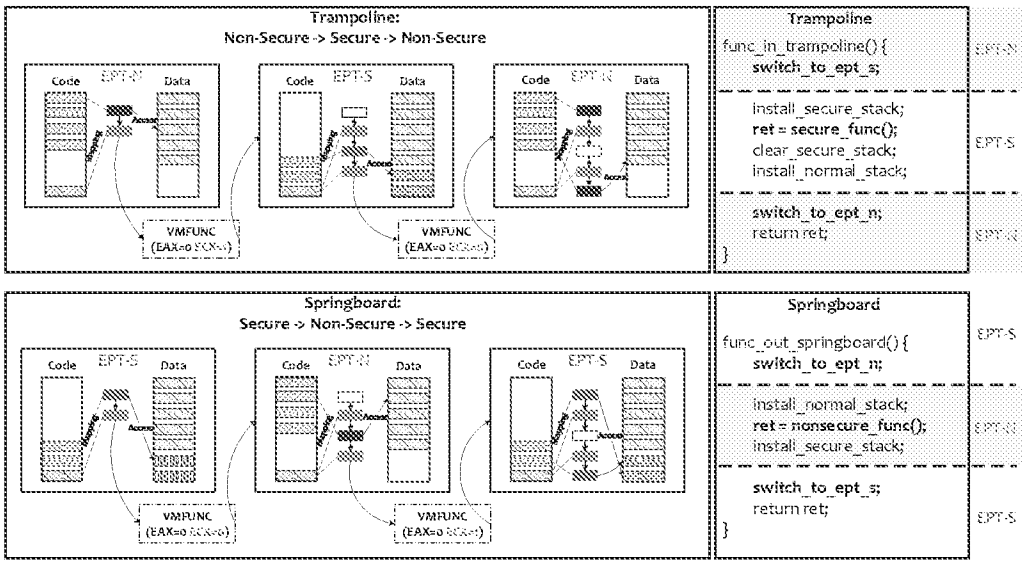


Figure 4: Execution flow through trampoline and springboard mechanisms

of these functions during creating a compartment. There is another situation that the reference to the *secrets* may be passed to the main compartment. However, since the callee function is not able to dereference the reference (otherwise, it should be added to the *secret* compartment), the *secrets* will not be leaked out.

3.3 Miscellaneous

Storage. In the initialization phase, *secrets* may be read from a standalone configuration file, executable binary, or DB schema. Such storages can always be accessed by system software, that there is no effective way to protect them. SeCage solves this problem through another approach by ensuring no *secret* in these storages. The *secrets* in the storage are replaced with some dummy data, during application launching, the dummy data will be restored to the real *secrets*. In the runtime, SeCage ensures no I/O write may happen in the sensitive functions so that *secrets* will not be leaked to the storages.

Interrupt handling. During execution in a *secret* compartment, there is no operating system support within the *EPT-S* context, thus no interrupt is allowed to be injected to the guest VM. When a non-root guest VM traps to the hypervisor due to an interrupt, the corresponding handler in SeCage checks whether it is in the context of *EPT-S*, and what kind of interrupt it is. If the interrupt happens during sensitive functions execution, it simply drops some kinds of interrupts (e.g., timer interrupt), and delays others (e.g, NMI, IPI) until *EPT-N* context.

Multi-threading. SeCage supports multi-threading programs. If there is only one VCPU running all the threads, since we drop timer interrupts to the *EPT-S* VCPU, the *EPT-S* context will not be preempted by other threads until it returning back to the *EPT-N* environment. If there are more than one VCPUs, since every VCPU has its own EPT, if one VCPU is in *EPT-S* context, other VCPUs can still run in *EPT-N* context and they are not allowed to read *secrets* in *EPT-S*.

3.4 Lifecycle Protection of *secret* Compartment

Figure 5 shows the lifecycle protection of a *secret* compartment. SeCage adds three hypercalls [3] as shown in Table 2.

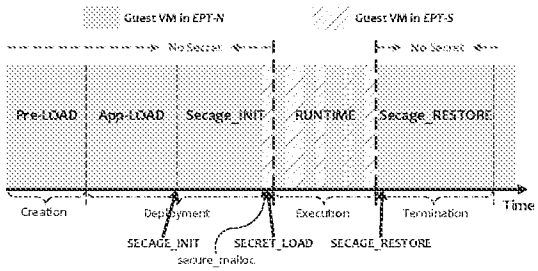


Figure 5: Life-cycle protection of *secrets*

Table 2: Descriptions of SeCage related hypercalls

Hypercall	Description
SECAGE_INIT	Called in *EPT-N*, triggers initializing.
SECRET_LOAD	Called in *EPT-S*, triggers secret loading.
SECAGE_RESTORE	Called in *EPT-N*, triggers restoration

Creation. Before an application is loaded into guest VM, SeCage utilizes the application decomposition framework to analyze the application and decompose it into a main compartment and several *secret* compartments. According to how the *secrets* are loaded, the *secrets* are replaced with dummy data in persistent storages like configuration file, executable binary or database. For example, if the *secrets* are loaded from the files (e.g., OpenSSL) or database during runtime, the *secrets* in the storage are replaced. Otherwise, the application is compiled after replacing the *secrets* in the source code with dummy data. Meanwhile, the developer is required to provide the mapping of *secrets* and the dummy

[3]Hypercall (or vmcall) is similar to syscall but is used to request the hypervisor's services

IBM - Ex. 1002-Part 10 of 10, Page 3421
IPR2025-00379 (IBM v. Croga)

data (e.g., <*key, length*> → *secret* binding), to the hypervisor through predefined secure offline channels. By this means, the hypervisor can load the real *secrets* into secure memory in the deployment phase.

Deployment. The process of application deployment includes following steps:

1. When launching an application, the instrumented code issues the *SECAGE_INIT* hypercall, which passes the start virtual addresses and the number of pages of sensitive functions and trampoline code as parameters. The hypervisor first checks the integrity of sensitive functions and trampoline code, and setups the *EPT-N* and *EPT-S* as described in section 3.1. It should be noted that *EPT-S* maps several reserved pages that are invisible from *EPT-N*, which will be used as secure heap and stack later.

2. The hypervisor invokes *VMENTER* to restore untrusted application execution. When the untrusted code invokes memory allocation function for *secrets* dummy counterpart, it is redirected to the *secure_malloc* in sensitive functions to allocate pages from the secure heap.

3. After the dummy *secrets* being copied to the secure heap in sensitive functions, the *SECRET_LOAD* hypercall is issued. The hypervisor then scans the secure heap memory, and replaces the dummy *secrets* with the real ones, according to the dummy to *secrets* mapping provided by the user.

Through the above protocol of application deployment, SeCage ensures that before *SECRET_LOAD*, there is no *secret* existing in the memory, thus even the execution environment is untrusted, no *secret* will be exposed. After *SECRET_LOAD*, *secrets* can only be accessed in the *secret* compartment, and thus code in the main compartment can never disclose it. Although the untrusted code may violate the protocol, by either skipping the invoking of hypercall, or not obeying rules of *secure_malloc*, the hypervisor can detect such a violation and the *secrets* will not be loaded into memory in such cases.

Runtime. At runtime, code in the main and the *secret* compartments execute concurrently. The SeCage mechanisms ensure that: (1) the *secrets* and their copies only exist in *EPT-S* mapping, (2) the *secrets* and their copies can only be used during sensitive functions execution. If the code in the main compartment attempts to access the *secrets* memory, a VMExit happens due to EPT violation. Hypervisor is then notified to checks the access pattern. If the access request is originated from the main compartment code to the *secrets*, and the corresponding function is not included in the extracted functions from static analysis, an attack attempt may happen. In this case, the hypervisor should stop the application execution and inform the user of the abnormal access request. If this access request is complied with the predefined policies according to the result of static analysis and the execution context, the hypervisor will then include the corresponding function to the sensitive functions closure in the *secret* compartment.

Termination. When the application is terminated, *secrets* should also be wiped out. If the application exits normally, it issues the *SECAGE_RESTORE* hypercall, so that hypervisor helps to remove the *EPT-S* of *secret* compartment. Even if the application exits abnormally or the application or OS refuses to inform the hypervisor, the *secrets* still only exist in *EPT-S* and thus will not be disclosed.

4. APPLICATION DECOMPOSITION

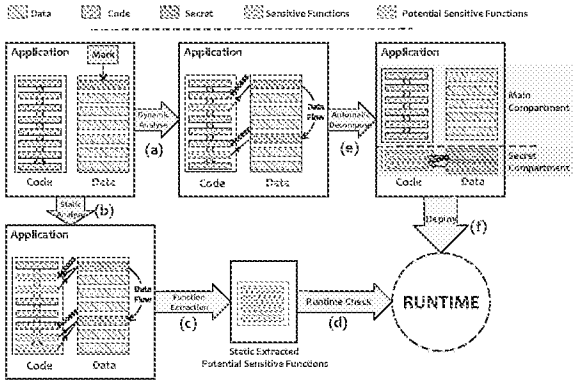


Figure 6: The general process of application analysis and decomposition

Figure 6 shows the process of the application analysis and decomposition. Given an application and the user-defined *secrets*, we need to analyze the data flow of the *secrets*, as well as the sensitive functions that are possible to access the *secret* data. While the static analysis can give a comprehensive analysis on all possible execution paths of the program, it has precision issues and may lead to larger TCB and overhead. We observe that in most cases, the execution flow of *secrets* manipulation is relatively fixed.

Based on this observation, We use a hybrid approach to extracting the *secret* closure. Specifically, we adopt the dynamic approach to achieving a flexible information flow control (IFC) analysis to get the most common but possibly incomplete *secret* closure (step (a)), and we also rely on the comprehensive results of static analysis to avoid these corner cases during runtime (step (b)(c)(d)). On the other hand, it provides a series of mechanisms to automatically decompose application during compilation time. Then, SeCage decouples these *secrets* and sensitive functions into an isolated *secret* compartment, which can be protected separately by the hypervisor (step (e)(f)).

4.1 Hybrid Secret Closure Extraction

Static taint analysis. We leverage CIL to carry out static taint analysis on the intermediate representation of the application. We denote the set of *secret* data as $\{s\}$, and the set of references to the *secrets* is donated as $\{s_{ref}\}$. Our target is to find all possible instructions, which is denoted as *sink*, that dereference variable $x \in \{s_{ref}\}$. We define the taint rules as follows:

$$\frac{n \rightarrow m, y := x, x \in \{s_{ref}\}}{\{s_{ref}\} := y :: \{s_{ref}\}} \quad (1)$$

$$\frac{n \rightarrow m, f(y_1, \ ... \ y_i, \ ... \ y_n), y_i \in \{s_{ref}\}}{\{s_{ref}\} := arg_i :: \{s_{ref}\}} \quad (2)$$

$$\frac{n \rightarrow m, y := f(y_1, ...y_i, ...y_n), ret_f \in \{s_{ref}\}}{\{s_{ref}\} := y :: \{s_{ref}\}} \quad (3)$$

IBM - Ex. 1002-Part 10 of 10, Page 3422
IPR2025-00379 (IBM v. Croga)

$$\frac{n \to m, y := \textbf{sink}(x), x \in \{s_{ref}\}}{\{s\} := y :: \{s\}, \{s_{ref}\} := y_{ref} :: \{s_{ref}\}} \quad (4)$$

When a program transmits from n to m, the data flow is tracked. Rule (1) says that the propagation of references to *secrets* should be tracked. Rule (2) and rule (3) define the rules of function calls, which mean that the propagation of references to *secrets* through function parameters and return values should also be tracked. Finally, rule (4) indicates that, upon any dereference of references to *secrets*, the *secret* and its reference need to be tracked, and the sink instruction should be also recorded. According to polyvariant analysis [7], the functions are analyzed multiple times. In our approach, the $\{s\}$ and $\{s_{ref}\}$ keep changing during the iterations and we stop the analysis when the program comes to a fixed point, where the *secret* and its reference set does not change anymore. Through this taint analysis, we can finally get a large number of potential sensitive functions.

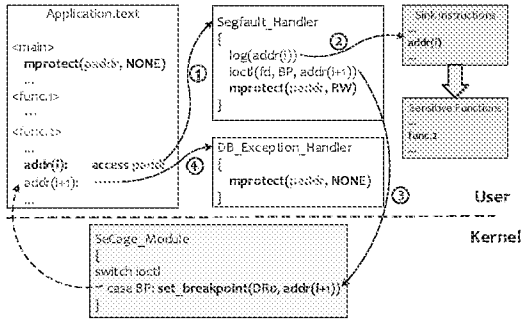


Figure 7: Combining *mprotect* and *debug exception* to dynamically extract sink instructions

Dynamic closure extraction. To get the compact *secret* closure, we adopt a simple but precise dynamic analysis by an innovative combination of *mprotect* and *debug exception* techniques. Figure 7 shows the process of dynamic sensitive functions extraction. At the very beginning, the application is instrumented to use *secure_malloc* to allocate memory for *secret* data. Then we use *mprotect* system call to protect the secure memory (*paddr*), and register a user mode handler to handle the corresponding segmentation fault upon violation. Anytime a sink instruction accesses the protected memory, it traps into the segfault handler (①). This handler records the fault address of the sink instruction (②), and issues one *IOCTL* system call to the kernel module (③), which set up the breakpoint to the next instruction in debug register 0 (DR0). After that, the handler revokes the *mprotect* to the *paddr* to make forward progress. Then, the sink instruction can successfully access the memory after the segfault handler returns; but the program traps to the predefined debug exception handler immediately in the next instruction (④), and in that case, the exception handler can setup the *mprotect* to the *paddr* again. We run the application several times with different workloads. For example, for Nginx server case, we can send different kinds of requests, until the set of sink instructions is fixed. Then we can get most of the sink instructions, as well as their corresponding sensitive functions.

For the OpenSSL case, we totally get 242 sensitive functions from static analysis and 20 from dynamic analysis. In our experiment, we select the dynamically extracted ones as the sensitive functions set $\{f_s\}$, and we do not find any EPT violation during the runtime. Nevertheless, we still decompose the potential sensitive functions which are not included in the dynamically extracted set to several separate memory sections, so that if any corner case happens, hypervisor can dynamically add them to the *secret* compartment in the runtime according to the predefined policies. When the hypervisor is required to add a potential sensitive function to the *secret* compartment, SeCage will collect as much information as possible to distinguish between legal and illegal accesses. For example, it uses the trapped instruction to check whether this is the sink instruction found in the static analysis; it is also required to check whether this potential sensitive function is modified, and whether the call-trace is forged, etc. If there is any predefined policy violated, SeCage aborts the execution and sends the abnormal information to the user to make decision.

4.2 Automatic Application Decomposition

Since the sensitive functions in $\{f_s\}$ are not self-contained, SeCage uses the trampoline mechanism to enable communications between $\{f_s\}$ and the functions in the main compartment, which is defined as $\{f_n\}$. For each function call f, we notate f_{caller} and f_{callee} as its caller and callee functions. If and only if one of the f_{caller} and f_{callee} belongs to $\{f_s\}$, it is required to define a corresponding trampoline function *t(f)*, which is used to replace f in the next phase. The formal definition is as follows:

$$P_s(func) = true \iff func \in \{f_s\}$$

$$\forall f, \exists t(f) = \begin{cases} f_{in} & P_s(f_{callee}) \wedge \neg P_s(f_{caller}) \\ f_{out} & P_s(f_{caller}) \wedge \neg P_s(f_{callee}) \\ f & \text{else} \end{cases}$$

If there is any function call from $\{f_n\}$ to $\{f_s\}$, we define a trampoline function f_{in}. Similarly, a springboard function f_{out} is defined for each function call from $\{f_s\}$ to $\{f_n\}$. We notate $\{f_t\}$ as the union of f_{in} and f_{out} sets.

Then we decompose an application to *secret* and main compartments for SeCage protection. There are totally three steps involved. First, 3 hypercalls are added to the application accordingly (Section 3.4). Second, an automated script is used to generate a file with definition and declaration of trampoline functions $\{f_t\}$, and modify the definition of sensitive functions *sfunc* in $\{f_s\}$ and trampoline functions *tfunc* in $\{f_t\}$ with GCC *section* attribute:

```
__attribute__ ((section (.se))) sfunc;
__attribute__ ((section (.tr))) tfunc;
```

Normally, the GCC compiler organizes the code into the *.text* section. The additional *section* attributes specify that *sfunc* and *tfunc* live in two particular *.se* and *.tr* sections' memory region which are isolated from memory of $\{f_n\}$. Thus, SeCage can protect them in the page granularity. Finally, during the compilation phase, the CIL parses the whole application, and replaces the $\{f_s\}$ involved function calls with their respective trampoline function calls in $\{f_t\}$.

SeCage also needs to modify the linker, to link the newly created *.se* and *.tr* sections to the predefined memory location, so that the *SECAGE_INIT* hypercall can pass the appropriate memory addresses as the *secret* compartment memory for the hypervisor to protect.

IBM - Ex. 1002-Part 10 of 10, Page 3423
IPR2025-00379 (IBM v. Croga)

5. USAGE SCENARIOS

We have implemented the compartment isolation part of SeCage based on KVM. The hypervisor is mainly responsible for compartment memory initialization, EPT violation and interrupt handling. In total, SeCage adds 1,019 lines of C code to the KVM, and the application decomposition framework consists of 167 lines of C code, 391 Bash code and 1,293 OCaml code to the CIL framework [1].

In this section, we present our case studies by applying SeCage to three representative examples: protecting keys of Nginx server with OpenSSL support from infamous HeartBleed attack, protecting keys of OpenSSH server from a malicious OS, and protecting CryptoLoop from kernel-level memory disclosure.

5.1 Protecting Nginx from HeartBleed

The HeartBleed attack allows attackers to persistently over-read 64KB memory data, which can lead to leakage of private keys [4]. We use one of the PoCs [6] to reproduce RSA private keys disclosure attack targeted on the Nginx server with a vulnerable OpenSSL version *1.0.1f*.

In our case, OpenSSL uses RSA as its cryptographic scheme. In short, two large prime numbers (p and q) are chosen as *secrets* in RSA, while their production N ($N = p \times q$) is referred to as the public key. When a client connects to the server, the server uses p and q as private keys to encrypt data (e.g., certificate), so that the client can use the corresponding public key N to decrypt them. To steal the private keys, an attacker can search the memory data returned in HeartBleed messages for prime numbers. Since the N and the length of the prime numbers (in bits) are known, it is pretty simple to find p or q by iteratively searching the exact (e.g., 1024) number of bits and checking if they can be divided by N as a prime. Once the attacker gets either of p or q, the other is just the result of dividing N by the known prime. In addition, there is a private key exponent called d which is another prime number we need to protect.

To defend against HeartBleed attack using SeCage, we first mark the *secrets* to protect. In the OpenSSL case, the exact prime numbers are stored in the memory pointed by d field in BIGNUM structure p, q and d. During the application decomposition phase, we get all of sensitive and trampoline functions as illustrated in section 4. The *secure_malloc* function is added to replace *OPENSSL_malloc* in some cases. OpenSSL uses BIGNUM memory pools to manage BIGNUM allocation, SeCage adds another secure pool which is used to allocate BIGNUM when it is protected one. The CILLY engine then delivers the modified intermediate code to GCC, which compiles it to the final executable binary. Besides, the private keys stored in the configuration file should be replaced by the same length of dummy data. After that, the hypervisor pushes the generated binary and configuration file to the guest VM, and starts the Nginx.

5.2 Protecting OpenSSH from Rootkit

In this scenario, we assume that the kernel is untrusted, e.g., there is a malicious rootkit[4] acting as part of operating system to arbitrarily access system memory, so that the *secret* is exposed to the light of day. We simulate this scenario by manually installing a rootkit that simply scans the system's memory and tries to find out *secret* in applications.

[4]Rootkits are mostly written as loadable kernel module that can do whatever kernel can do.

We run the OpenSSH server in this untrusted environment. During the authentication of OpenSSH server, the client uses the host key and server key as server's public keys to encrypt the session key, and the private keys are used by OpenSSH server for session key decryption. The server is also required to send the acknowledge encrypted using the session key to the client for authentication. Thus the private keys are of vital importance such that the disclosure of them can lead to the leakage of all network traffic, and the attacker can pretend as server to deceive clients. Similar to the Nginx server protection, we leverage the application decomposition framework to analyze and decompose OpenSSH, since OpenSSH uses OpenSSL with RSA cryptographic schema as well, the process is quite similar with the previous example.

5.3 Protecting CryptoLoop from Kernel Memory Disclosure

As shown in Table 1, about half of the vulnerabilities are kernel-level memory disclosure. Different from application-level memory disclosure, one successful exploit of these vulnerabilities can put the whole kernel memory data at risk of being disclosed. In this scenario, we choose to enhance the Linux kernel's disk encryption module, CryptoLoop, which is used to create and manipulate encrypted file systems by making use of loop devices. CryptoLoop relies on the Crypto API in kernel, which provides multiple transformation algorithms. In our case, it uses the CBC-AES cipher algorithm to do the cryptographic operations.

In this case, we define the AES cryptographic keys as *secrets*. The sensitive functions extraction framework is a little different, but the overall principle is the same: we allocate a new page for the AES keys, and set the corresponding page table entry as non-present. After that, we combine the page fault with the debug exception to track and record the *secrets* related functions. Different from the cases of Nginx and OpenSSH, the AES keys are originated from user-provided password with hash transformation, thus the user has to provide a dummy password, and calculate the real hash value offline, so that SeCage can replace the dummy AES keys with the real ones.

6. SECURITY EVALUATION

6.1 Empirical Evaluation

***Secrets* exposure elimination.** We evaluate to what extent can SeCage achieve the goal of exposing *secrets* to adversaries. To achieve this, we write a tool to scan the whole VM memory and find targeted *secrets* in it when running Nginx and OpenSSH server. Figure 8 shows the heap memory [5] of related processes, for the software without SeCage protection ((a) and (b)), there are several targeted *secrets* found in the heap. In contrast, for the SeCage protected ones ((c) and (d)), we cannot find any fragment of the *secrets*. For the CryptoLoop case, we run the *fio* benchmark to constantly do I/O operations in the encrypted file system, and traverse the kernel page tables to dump the valid kernel memory pages in a kernel module. With SeCage protection, we find no AES keys within the dumped kernel memory. While without SeCage protection, we can find 6 AES keys.

[5]we also dumped the stack memory, and found no *secret* at all

IBM - Ex. 1002-Part 10 of 10, Page 3424
IPR2025-00379 (IBM v. Croga)

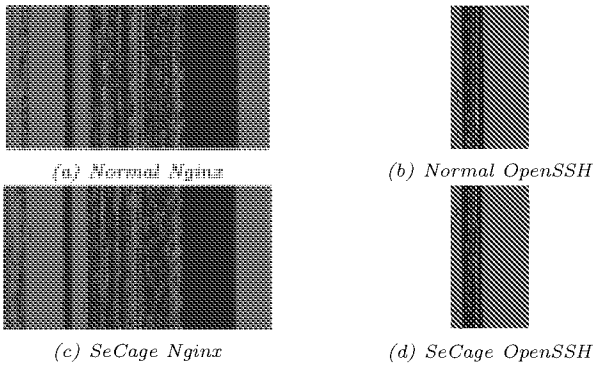


Figure 8: Heap memory layout for Nginx and OpenSSH server processes. (Green blocks are memory in use, black areas are unused and the red bars are targeted *secrets* found in the memory.)

Security protection for Nginx+OpenSSL. Before applying SeCage to Nginx server, we conduct HeartBleed attacks on it, it is relatively simple to retrieve the RSA private keys, we get private keys after sending 69 HeartBleed requests. After decomposing the OpenSSL and deploying it to SeCage framework, we mount the attack again. At this time, no fragment of private keys is leaked no matter how many HeartBleed requests are sent by the attacker.

Security protection for OpenSSH+OpenSSL. From figure 8 (b), we can see that 3 fragments of private keys exist in the OpenSSH process's dumped memory, which are exactly what the kernel rootkit can read and expose. While leveraging the SeCage, no one exists in the malicious rootkit's view, which means no *secret* will be leaked.

Security protection for CryptoLoop. AES cryptographic keys are stored in kernel memory space, any *Out-of-bound read* kernel memory disclosure vulnerabilities (e.g., CVE-2013-5666) and rootkit can read arbitrary memory from kernel space, leading to kernel *secrets* disclosure. With the Crypto API being enhanced by SeCage, no cryptographic key is stored in the normal kernel memory, thus no *secrets* can be leaked during CryptoLoop execution.

6.2 Security Analysis

Reduced attack surfaces. The attack surfaces of SeCage are quite related to the code size of the sensitive functions. The larger code base the *secret* compartment contains, the more vulnerabilities may be exploited by the attackers. After applying the proposed dynamic analysis, we get only 1350 LoCs for OpenSSL case, and 430 LoCs for CryptoLoop case. What is more, SeCage adds some restrictions for the sensitive functions, e.g., there should be no printf-like functions thus no format attack can be conducted, etc. Therefore, it is reasonable to assume no vulnerabilities in the code of *secret* compartment.

Iago attack and ROP attack. Iago attack [15] presents a complete attack example that the malicious kernel can cause s_server in OpenSSL to disclose its secret key by manipulating the return values of *brk* and *mmap2* system calls, and conduct a return-oriented programming (ROP) attack to write the contents of the secret key to *stderr*. With the protection by SeCage, if there is any system call invocation in the *secret* compartment, the return value from the *springboard* will be checked to avoid the invalid one.

Meanwhile, trampoline code is marked as read-only thus not injectable thanks to the protection from EPT. On the other hand, there is only very small code base in sensitive functions for an attacker to exploit ROP attack. The SeCage's mechanism ensures that the trampoline can only enter *EPT-S* through designated function entries. This means gadgets can only be at the function granularity and thus attackers may not have sufficient gadgets, which further reduces the probability of ROP attacks. What is more, as the execution in the *secret* compartment will use the secure stack, an adversary has no chance to fake a stack on behalf of the *secret* compartment. This further nullifies the chance of performing an ROP attack. If an ROP attack succeeds in the main compartment, the payload still cannot access the *secrets* because of the compartment isolation.

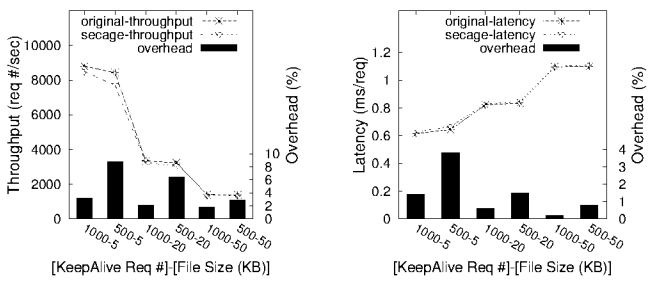
Rollback attack. An attacker may collude with a malicious OS to rollback the password states using tricks like memory snapshot rollback, and launch a brute-force attack to guess the login password. SeCage is resistant to such an attack, due to the fact that the *secret* states are maintained by the hypervisor rather than the OS, it can refuse to rollback *secrets* to old states by well-designed protocols.

Coppersmith's attack. According to [4], when processing requests in Nginx, some fragments of private keys may exist in memory. This makes it vulnerable to Coppersmith's Attack [2], a complex mathematical approach that is used to split N into p and q with the knowledge of just part of a prime number. SeCage is designed to prevent such attacks since fragments of *secrets* can only be copied to secure heap and stack, and no fragment copies can be leaked to the memory of the main compartment.

Guest page mapping attack. A malicious guest OS can conduct attacks by changing the guest page mapping of the secure heap and the stack pages after SeCage initialization. Hence, during invoking trampoline and copying *secrets*, *secrets* may be leaked to unprotected memory space. SeCage prevents such attacks by tracking the page tables modification of these protected data pages. According to section 7, only 3 pages are required to protect, and at most 12 PTEs need to be tracked.

VMFUNC faking attack. Another attack the malicious OS can conduct is to fake the *EPT-S* entry point by invoking self-prepared VMFUNC instructions. However, in *EPT-S*, the executable memory is predefined and fixed, the memory pages which contain the faked VMFUNC instructions are non-executable when switching to the *EPT-S* context, thus the execution will be trapped into the hypervisor. A more sophisticated attack is to carefully construct a guest memory mapping that maps GVA to GPA, so that the GVA of the memory page is the prior one page before the sensitive functions pages, and put the VMFUNC instruction into the last bytes of this page. Hence, when this VMFUNC is invoked, the next program counter locates in the sensitive functions, which is executable. This complicated attack can bypass the restriction that the calls to the *secret* compartment can only be done with several fixed entry gates in the trampoline. Nevertheless, it just adds a few more harmless entry gates (the first byte of the sensitive functions pages). To further eliminate such attack surfaces, we can adapt the approach used by [48] to putting a single byte *INT3* instruction at the beginning of each page of sensitive functions, to prevent this carefully constructed VMFUNC faking attack.

IBM - Ex. 1002-Part 10 of 10, Page 3425
IPR2025-00379 (IBM v. Croga)



(a) Throughput *(b) Latency*

Figure 9: Nginx performance degradation

7. PERFORMANCE EVALUATION

To evaluate the efficiency of SeCage, we measure the performance slowdown for applications under protection by SeCage and collect some statistics regarding the performance and resource consumption to conduct a detailed analysis.

All experiments are done on a machine with 4 Intel Xeon cores (8 hardware threads using hyper-threading) running at 3.4 GHz and with 32 GB memory. The host kernel for KVM is Linux 3.13.7 and the guest kernel is Linux 3.16.1. Each guest VM is configured with 2 virtual cores and 4 GB memory.

7.1 Application Benchmarks

Nginx throughput and latency. Figure 9 shows the performance degradation of Nginx server with SeCage protection. We use the *ab* benchmark to simulate 20 concurrent clients constantly sending N KeepAlive [6] requests, each request asks the server to transfer X bytes file. To decide what are the common values N and X should be, we scan through the Facebook pages, and find that within one session, there are normally hundreds to thousands of requests, each one ranges from 1 to 100 KB. Since the private keys are only used during connection establishment, the overhead can be largely amortized as the number of KeepAlive requests and the size of file grow. When N is 1000 and X is 50K, SeCage only adds about 0.002 ms latency for each request in average, and the throughput overhead is only about 1.8%. We also run the SeCage in a problematic configuration where all requests require new TLS establishment with 0.5KB small file, to test the performance slowdown in the worst case. The result shows that the overhead is about 40%. While in real-world scenarios, most HTTP-connections are persistent and keep-alive headers are used [5].

OpenSSH latency. To evaluate the slowdown of OpenSSH operation, we run a script to constantly use SSH to login the server and execute common Linux commands such as *ls* and *netstat*, and utilize *time* command to show the latency. We find that on average, SeCage introduces about 6 ms (%3 overhead) for each request.

CryptoLoop I/O bandwidth. We evaluate disk I/O throughput when using CryptoLoop encrypted file system. We leverage *fio* benchmark with sequential read/write configurations to show the overhead introduced by SeCage. We find that the average slowdown of I/O operations is about 4% when using SeCage protection.

[6]KeepAlive header asks the server to not shut down the connection after each request is done.

7.2 Performance Analysis

EPTP switching times and cost. The overhead introduced by SeCage is mainly from context switches between *EPT-N* and *EPT-S*. We instrument the application to log the times of trampoline invocation for different operations; each trampline invocation introduces 2 vmfunc calls (section 3.2). Table 3 shows the statistics after the Nginx server processed 1000 requests, OpenSSH handled one login request, and CryptoLoop operated 10M I/O writes. N→S means the number of trampoline calls from the main to the *secret* compartment, and S→N means that from the *secret* to the main compartment. We can see that there are limited numbers of context switches compared to the long execution path. We then do a study of how much time each vmfunc call takes, and compare it with normal function call, syscall and vmcall (hypercall). As shown in Table 4, the cost of a vmfunc is similar with a syscall, while a vmcall takes much longer time. Thus, SeCage provides the hypervisor-level protection at the cost of system calls.

Table 3: Statistics of trampoline invocation times

Nginx/1000 req		OpenSSH/login		CryptoLoop/10M	
N→S	S→N	N→S	S→N	N→S	S→N
64693	8550	20480	40960	98230	11658

Table 4: Overhead for call, syscall, vmcall and vmfunc

call	syscall	vmcall	vmfunc
3.37 ns	64.54 ns	544.4 ns	63.22 ns

Memory consumption. Since SeCage needs to setup a shadow *EPT-S* for each protected *secret* compartment, we calculate how much additional memory is needed. It depends on how many sensitive functions and trampoline code are extracted, and how much secure heap and stack is reserved. The results are shown in Table 5, in the OpenSSL case, we extracted about 20 sensitive functions, and declare the corresponding 41 trampoline functions. For the *secret* memory, we only reserve 1 page for the secure heap and 2 pages for stack. Thus the total amount of shadow memory is only 40KB. For the CryptoLoop case, since it is much simpler than OpenSSL, it only takes about 7 pages (28KB).

Table 5: Statistics regarding memory consumption

	$\{f_s\}$	$\{f_t\}$	*sec-heap*	*sec-stack*	total
Nginx	20 (5)	41 (2)	1	2	10 pages
OpenSSH	21 (5)	41 (2)	1	2	10 pages
CryptoLoop	12 (3)	14 (1)	1	2	7 pages

8. RELATED WORK

Application-level secret protection. There are many systems aiming at protecting secret data (e.g., cryptographic keys) from application-level memory disclosure attacks by leveraging library or OS kernel support [36, 8, 22, 37, 26]. For example, DieHarder [36] presents a security-oriented memory allocator to defend heap-based memory attacks. SecureHeap [8] is a patch for OpenSSL from Akamai Technology, which allocates specific area to store private keys to prevent private keys from disclosing. CRYPTON [22] designs

IBM - Ex. 1002-Part 10 of 10, Page 3426
IPR2025-00379 (IBM v. Croga)

a data abstraction and browser primitive to isolate sensitive data within the same origin. However, they all rely on the trustworthiness of the entire library of OS kernel, and thus still suffer from larger attack surfaces compared to SeCage.

Other approaches try to keep cryptographic keys solely inside CPU [37, 26]. For example, Safekeeping [37] uses x86 SSE XMM registers to ensure no cryptographic key appear in its entirety in the RAM, while allowing efficient cryptographic computations. Similarly, Copker [26] implements asymmetric cryptosystems entirely within the CPU, without leaking the plaintext of private keys to memory. However, code running in cache needs to be specially crafted and it needs to rely on the trustworthiness of OS kernel for security.

Hypervisor-based application protection. Systems like CHAOS [17, 16], OverShadow [18] and others [19, 28, 53] leverage virtualization-based approaches to providing isolated execution environment to protect an application from a compromised OS. Specifically, they leverage hypervisor provided memory isolation, and intercept transition between a protected application and the OS to enforce isolation. InkTag [28] makes a step further to use paraverification to ease verifying of OS to defend again Iago attacks [15]. Compared to these systems, SeCage aims to provide fine-grained intra-domain protection instead of whole application protection. Further, the separation of control and data planes significantly reduces hypervisor intervention and thus reduces performance overhead.

Some other virtualization-based systems [29, 25, 45, 42] run protected applications in a trusted component with restricted functionalities (e.g., an application-specific OS), and use hypervisor to enforce the interactions between trusted and untrusted components. Compared with SeCage, such approaches have larger TCB since they need to trust all secure components like the OS. Further, they all need to retrofit the whole OS and thus require significant porting effort to run real, large software atop.

There are various systems aiming at providing isolated environment for the whole virtual machine [54, 50, 44], though they are efficient in guest VM protection, they cannot defend against intra-domain attacks.

Protecting pieces of application logic. Flicker [35] leverages TPM and takes a bottom-up approach to provide a hardware-support isolation of security-sensitive code. With the help of TPM, securely running Pieces of Application Logic (PAL) can rely only on a few hundred lines of code as its TCB. However, since Flicker heavily uses the hardware support for a dynamic root of trust for measurement (DRTM), it introduces very high overhead. TrustVisor [34] proposes a special-purpose hypervisor to isolate the execution of the PAL. Compared with Flicker, it introduces the software-based TPM called *micro-TPM (uTPM)* for each PAL, to realize a more efficient PAL protection. Flicker and TrustVisor show a kind of effectively isolated code execution with extremely small TCB, they both assume that the protected code is self-contained with predefined inputs and outputs, and heavily depend on programmers to specify sensitive functions, as well as (un)marshal parameters between trusted and untrusted mode, which kind of design makes them very difficult to adapt to existing systems. MiniBox [31] focus on providing a two-way sandbox for both PAL and OS: besides protecting PAL from malicious OS as TrustVisor does, it also prevents untrusted applications compromising the underlying OS. Since MiniBox uses TrustVisor to do the first part, they share the same problems.

Hardware-assisted protection. There is also much work on designing new hardware features to isolate critical code and data [33, 43, 24, 14, 20, 10, 47, 49]. For example, XOMOS [33] is built upon XOM [32] to support tamper-resistant software. The concept of compartment is borrowed from XOMOS, but is used to run a closure of code operating specific secret data. Further, SeCage leverages existing hardware virtualization features to enforce strong protection. Haven [10] introduces the notion of shielded execution, which utilizes Intel's recent SGX security proposal to protect the code and data of the unmodified application from divulging and tampering from privileged code and physical attack, but only targets whole application protection instead of intra-application protection. SeCage illustrates how to decompose large-scale software and adapt it to different hardware-assisted isolation environment.

Mimosa [27] uses hardware transactional memory (HTM) to protect private keys from memory disclosure attacks. It leverages the strong atomicity guarantee provided by HTM to prevent malicious concurrent access to the memory of sensitive data. Though Mimosa shares almost the same threat model with SeCage, it has some limitations, e.g., in order to prevent attackers from accessing debug registers (which is used to store the AES master key of the private keys), it needs to patch the ptrace system call, disable loadable kernel modules (LKMs) and kmem, and remove JTAG ports. Compared with SeCage, Mimosa can only adapt much simplified PolarSSL instead of OpenSSL to the HTM protection.

CODOMs [47] enforces instruction-pointer capabilities to provide a code-centric memory domain protection. CHERI [49] implements a capability coprocessor and tagged memory to enable capability-based addressing for intra-program protection. Both designs promise very small TCB (e.g., microprocessor and up to limited software component like trusted kernel), and can provide strong isolation enforced by hardware. However, since they all need newly specific designed hardware which are currently not available, they are all implemented on emulators. Further, they usually require rewriting of the whole software stack, which lead to non-trivial engineering work and make them not ready for deployment.

Privilege separation. PrivTrans [13] is a privilege separation tool that can automatically decouple a program into monitor and slave parts provided a few programmer annotations. Compared with the mechanism of SeCage, PrivTrans approach partitions code in instruction-level, which introduces a large number of expensive calls between the two parts. Meanwhile, it trusts the whole operating system, whose threat model is quite different with SeCage. Virtual Ghost [21] is the first one to combines compilation technique and OS code runtime check to provide critical application with some secure services, and protect it from compromised OS without higher privilege level than the kernel. However, it needs to retrofit OS and runs it in a secure virtual architecture (SVA), which requires non-trivial efforts to deploy it in current infrastructures.

VMFUNC utilization. SeCage is not the first system to use the Intel's VMFUNC hardware feature. Following the philosophy of separating authentication from authorization, CrossOver [30] extends Intel's VMFUNC mechanism to provide a flexible cross-world call scheme that allows calls not only across VMs, but across different privilege levels

IBM - Ex. 1002-Part 10 of 10, Page 3427
IPR2025-00379 (IBM v. Croga)

and address spaces. We believe such a mechanism can be applied to SeCage to provide more flexible protection. An effort from the open-source community [9] takes advantages of VMFUNC, and proposes an efficient NFV on Xen. As far as we know, SeCage is the first system making uses of VMFUNC to protect pieces of application logic and prevent from both user and kernel level memory disclosure attacks.

9. DISCUSSION AND LIMITATION

Deployment effort. The deployment of SeCage is not fully automated, the minor manual effort is required. For OpenSSL and CryptoLoop, we respectively add 25 and 15 LoCs. They are mainly attributed to the dynamic analysis, *secret* memory allocation and hypercall used for initialization. Comparing it with other work, TrustVisor [34], Mimosa [27] and many others did not support OpenSSL so they resort to the much more simplified PolarSSL. PrivTrans [13] only adds 2 annotations, however it puts all cryptographic operations in the monitor, while SeCage only allows very limited code in the *secret* compartment.

SeCage TCB. For one *secret* compartment, the TCB contains hardware, hypervisor and the sensitive functions. Currently, our implementation uses KVM as the trusted hypervisor, which arguably has a large TCB. However, as SeCage only involves minor functionalities in hypervisor, it should be easy to port SeCage to a security-enhanced hypervisor such as TrustVisor [34] or XMHF [46] to further reduce TCB, which will be our future work.

The scope of *secrets*. SeCage is supposed to protect critical *secrets* like private keys, whose related sensitive functions are limited to a small piece of code. Since the small code base of a *secret* compartment is one of the most important assurance to *secrets*' security, SeCage is not designed for protecting data whose closure involves large fraction of the code. Fortunately, the three cases we studied, which should be representative of large-scale software, fit very well with our scope.

Static library vs. dynamic library. Currently, SeCage compiles OpenSSL as a static library, and slightly modifies the linker to link the sensitive and trampoline functions to specific memory locations. We can also adopt dynamic libraries, by modifying system's loader. We do not assume the loader as trusted; even if it behaves maliciously by refusing to load sensitive functions or loading them to wrong memory locations, the *secrets* will still not be disclosed. This is because the hypervisor can reject to load the *secrets* during the integrity checking phase (Section 3.4).

10. CONCLUSION AND FUTURE WORK

We presented SeCage, a novel approach that leverages virtualization to protect user-defined *secrets* from application vulnerabilities and malicious OS. SeCage prevents sensitive information disclosure by completely shadowing the *secret* compartment from the main compartment, and utilizes Intel hardware extension to enormously reduce hypervisor intervention during application runtime. Meanwhile, SeCage provides a practical application analysis and decomposition framework to (mostly) automatically deploy applications. SeCage was shown to be useful by protecting real and large-scale software from HeartBleed attack, kernel memory disclosure and rootkit memory scanning effectively, and incurring negligible performance overhead to applications.

We plan to extend our work in several directions. First, we plan to deploy more applications with SeCage to prevent sensitive memory disclosure based on the CVEs. Second, we plan to adopt binary rewriting techniques to avoid the requirement of source code rewriting. Third, we plan to implement SeCage in other virtualization platform, e.g., Xen.

11. ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful comments. This work is supported in part by a research grant from Huawei Technologies, Inc., National Natural Science Foundation (61303011), a foundation for the Author of National Excellent Doctoral Dissertation of PR China (No. TS0220103006), Program for New Century Excellent Talents in University of Ministry of Education of China (ZXZY037003), the Shanghai Science and Technology Development Fund for high-tech achievement translation (No. 14511100902), Zhangjiang Hi-Tech program (No. 201501-YP-B108-012), and the Singapore NRF (CREATE E2S2).

12. REFERENCES

[1] CIL. http://kerneis.github.io/cil/.
[2] Coppersmith's attack. http://en.wikipedia.org/wiki/Coppersmith's_Attack.
[3] The heartbleed bug. http://heartbleed.com/.
[4] How heartbleed leaked private keys. http://blog.cloudflare.com/searching-for-the-prime-suspect-how-heartbleed-leaked-private-keys/.
[5] Http persistent connection wiki. https://en.wikipedia.org/wiki/HTTP_persistent_connection.
[6] Poc of private key leakage using heartbleed. https://github.com/einaros/heartbleed-tools.
[7] Polyvariance. http://en.wikipedia.org/wiki/Polyvariance.
[8] Secure heap patch for heartbleed. http://www.mail-archive.com/openssl-users@openssl.org/msg73503.html.
[9] Xen as high-performance nfv platform. http://events.linuxfoundation.org/sites/events/files/slides/XenAsHighPerformanceNFVPlatform.pdf.
[10] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *OSDI*, 2014.
[11] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazieres, and D. Boneh. Hacking blind. In *S&P*, 2014.
[12] E. Bosman and H. Bos. Framing signals: A return to portable shellcode. In *S&P*, 2014.
[13] D. Brumley and D. Song. Privtrans: Automatically partitioning programs for privilege separation. In *Usenix Security*, 2004.
[14] D. Champagne and R. B. Lee. Scalable architectural support for trusted software. In *HPCA*, 2010.
[15] S. Checkoway and H. Shacham. Iago attacks: why the system call api is a bad untrusted rpc interface. In *ASPLOS*, 2013.
[16] H. Chen, J. Chen, W. Mao, and F. Yan. Daonity–grid security from two levels of virtualization. *Information Security Technical Report*, 12(3):123–138, 2007.
[17] H. Chen, F. Zhang, C. Chen, Z. Yang, R. Chen, B. Zang, and W. Mao. Tamper-resistant execution in

IBM - Ex. 1002-Part 10 of 10, Page 3428
IPR2025-00379 (IBM v. Croga)

an untrusted operating system using a virtual machine monitor. *Technical Report, FDUPPITR-2007-0801*.

[18] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *ASPLOS*, 2008.

[19] Y. Cheng, X. Ding, and R. Deng. Appshield: Protecting applications against untrusted operating system. *Technical Report, SMU-SIS-13*, 2013.

[20] S. Chhabra, B. Rogers, Y. Solihin, and M. Prvulovic. Secureme: a hardware-software approach to full system security. In *ICS*, 2011.

[21] J. Criswell, N. Dautenhahn, and V. Adve. Virtual ghost: protecting applications from hostile operating systems. In *ASPLOS*, 2014.

[22] X. Dong, Z. Chen, H. Siadati, S. Tople, P. Saxena, and Z. Liang. Protecting sensitive web content from client-side vulnerabilities with cryptons. In *CCS*, 2013.

[23] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, et al. The matter of heartbleed. In *IMC*, 2014.

[24] J. S. Dwoskin and R. B. Lee. Hardware-rooted trust for secure key management and transient trust. In *CCS*, 2007.

[25] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *SOSP*, 2003.

[26] L. Guan, J. Lin, B. Luo, and J. Jing. Copker: Computing with private keys without ram. In *NDSS*, 2014.

[27] L. Guan, J. Lin, B. Luo, J. Jing, and J. Wang. Protecting private keys against memory disclosure attacks using hardware transactional memory. In *S&P*, 2015.

[28] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel. Inktag: secure applications on an untrusted operating system. 2013.

[29] P. C. Kwan and G. Durfee. Practical uses of virtual machines for protection of sensitive user data. In *ISPEC*. 2007.

[30] W. Li, Y. Xia, H. Chen, B. Zang, and H. Guan. Reducing world switches in virtualized environment with flexible cross-world calls. In *ISCA*, 2015.

[31] Y. Li, J. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry. Minibox: A two-way sandbox for x86 native code. In *ATC*, 2014.

[32] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. 2000.

[33] D. Lie, C. A. Thekkath, and M. Horowitz. Implementing an untrusted operating system on trusted hardware. In *SOSP*, 2003.

[34] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. Trustvisor: Efficient tcb reduction and attestation. In *S&P*, 2010.

[35] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for tcb minimization. In *EuroSys*, 2008.

[36] G. Novark and E. D. Berger. Dieharder: securing the heap. In *CCS*, 2010.

[37] T. P. Parker and S. Xu. A method for safekeeping cryptographic keys from memory disclosure attacks. In *Trusted Systems*. 2010.

[38] J. Shi, X. Song, H. Chen, and B. Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *DDSN-W*, 2011.

[39] G. Smith, C. E. Irvine, D. Volpano, et al. A sound type system for secure flow analysis. Journal of Computer Security, 1996.

[40] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In *S&P*, 2013.

[41] R. Strackx, B. Jacobs, and F. Piessens. Ice: a passive, high-speed, state-continuity scheme. In *ACSAC*, 2014.

[42] R. Strackx and F. Piessens. Fides: Selectively hardening software application components against kernel-level or process-level malware. In *CCS*, 2012.

[43] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas. Aegis: architecture for tamper-evident and tamper-resistant processing. In *ICS*, 2003.

[44] J. Szefer and R. B. Lee. Architectural support for hypervisor-secure virtualization. In *ASPLOS*, 2012.

[45] R. Ta-Min, L. Litty, and D. Lie. Splitting interfaces: Making trust between applications and operating systems configurable. In *OSDI*, 2006.

[46] A. Vasudevan, S. Chaki, L. Jia, J. McCune, J. Newsome, and A. Datta. Design, implementation and verification of an extensible and modular hypervisor framework. In *S&P*, 2013.

[47] L. Vilanova, M. Ben-Yehuda, N. Navarro, Y. Etsion, and M. Valero. Codoms: protecting software with code-centric memory domains. In *ISCA*, 2014.

[48] Z. Wang, C. Wu, M. Grace, and X. Jiang. Isolating commodity hosted hypervisors with hyperlock. In *CCS*, 2012.

[49] J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, et al. The cheri capability model: Revisiting risc in an age of risk. In *ISCA*, 2014.

[50] Y. Xia, Y. Liu, and H. Chen. Architecture support for guest-transparent vm protection from untrusted hypervisor and physical attacks. In *HPCA*, 2013.

[51] Y. Xia, Y. Liu, H. Chen, and B. Zang. Defending against vm rollback attack. In *DCDV*, 2012.

[52] Y. Xia, Y. Liu, H. Guan, Y. Chen, T. Chen, B. Zang, and H. Chen. Secure outsourcing of virtual appliance. *IEEE Transactions on Cloud Computing*, 2015.

[53] J. Yang and K. G. Shin. Using hypervisor to provide data secrecy for user applications on a per-page basis. In *VEE*, 2008.

[54] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *SOSP*, 2011.

[55] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-vm side channels and their use to extract private keys. In *CCS*, 2012.

IBM - Ex. 1002-Part 10 of 10, Page 3429
IPR2025-00379 (IBM v. Croga)

Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections

Abhinav Srivastava and Jonathon Giffin

School of Computer Science, Georgia Institute of Technology
{abhinav,giffin}@cc.gatech.edu

Abstract. Application-level firewalls block traffic based on the process that is sending or receiving the network flow. They help detect bots, worms, and backdoors that send or receive malicious packets without the knowledge of users. Recent attacks show that these firewalls can be disabled by knowledgeable attackers. To counter this threat, we develop VMwall, a fine-grained tamper-resistant process-oriented firewall. VMwall's design blends the process knowledge of application-level firewalls with the isolation of traditional stand-alone firewalls. VMwall uses the Xen hypervisor to provide protection from malware, and it correlates TCP or UDP traffic with process information using virtual machine introspection. Experiments show that VMwall successfully blocks numerous real attacks—bots, worms, and backdoors—against a Linux system while allowing all legitimate network flows. VMwall is performant, imposing only a 0–1 millisecond delay on TCP connection establishment, less than a millisecond delay on UDP connections, and a 1–7% slowdown on network-bound applications. Our attack analysis argues that with the use of appropriate external protection of guest kernels, VMwall's introspection remains robust and helps identify malicious traffic.

Keywords: Firewall, virtual machine introspection, attack prevention.

1 Introduction

Application-level firewalls are an important component of a computer system's layered defenses. They filter inbound and outbound network packets based on an access policy that includes lists of processes allowed to make network connections. This fine-grained filtering is possible because application-level firewalls have a complete view of the system on which they execute. In contrast, network- or host-level firewalls provide coarse-grained filtering using ports and IP addresses. Application-level firewalls help detect and block malicious processes, such as bots, worms, backdoors, adware, and spyware, that try to send or receive network flows in violation of the fine-grained policies. To be successful, these firewalls must be fast, mediate all network traffic, and accurately identify executing processes.

The conventional design of application-level firewalls has a deficiency that may prevent filtering of malicious traffic. The architectures pass packet information

R. Lippmann, E. Kirda, and A. Trachtenberg (Eds.): RAID 2008, LNCS 5230, pp. 39–58, 2008.
© Springer-Verlag Berlin Heidelberg 2008

from a kernel-level network tap up to a user-level firewall process that executes alongside malicious software. The firewall is both performant and able to identify the processes attached to a network flow, but it is exposed to direct attack by any malicious software aware of the firewall. Baliga et al. [1] demonstrated the ease of such attacks by manipulating the netfilter framework inside the Linux kernel to remove the hooks to packet filtering functions. Similarly, attackers can disable the Windows Firewall by halting particular services normally running on the system. Once the firewall fails, then all network traffic will be unmediated and the malware can send and receive data at will.

An alternative design isolates firewalls from vulnerable systems to gain protection from direct attack. Virtual machines allow construction of firewall appliances that execute outside of operating systems under attack. Such firewalls dispense with application-level knowledge and filter inbound and outbound packets using coarse-grained rules over IP addresses and port numbers. Attacks can easily evade these firewalls by using allowed ports directly or via tunneling.

This paper leverages the benefits of both application-level firewalls and virtual machine isolation to develop tamper-resistant application-oriented firewalls. Such a firewall needs good visibility of the system so that it can correlate network flows with processes, but it also needs strong isolation from any user-level or kernel-level malware that may be present. We architect an application-level firewall resistant to direct attack from malicious software on the system. Our design isolates the application-level firewall in a trusted virtual machine (VM) and relies on the hypervisor to limit the attack surface between any untrusted VM running malware and the trusted VM. Our firewall, executing in the trusted VM, becomes an application-level firewall by using virtual machine introspection (VMI) [10] to identify the process in another VM that is connected to a suspicious network flow.

Our prototype implementation, VMwall, uses the Xen [2] hypervisor to remain isolated from malicious software. VMwall executes entirely within Xen's trusted virtual machine dom0; it operates with both paravirtualized and fully virtualized domains. A dom0 kernel component intercepts network connections to and from untrusted virtual machines. A user-space process performs introspection to correlate each flow to a sending or receiving process, and it then uses a predefined security policy to decide whether the connection should be allowed or blocked. Policies are straightforward whitelists of known software in the untrusted VM allowed to communicate over the network. To correlate network flows with processes, VMwall's user-space component maps the untrusted operating system's kernel memory into its own address space and uses programmed knowledge of kernel data structures to extract the identity of the process attached to the flow.

VMwall is effective at identifying and blocking malicious network connections without imposing significant performance degradation upon network traffic. Using a Linux system and a collection of known attacks that either send or receive network traffic, we show that VMwall identifies all malicious connections immediately when the first packet is sent or received. In particular, VMwall blocked 100% of the malicious connections when tested against bots, worms,

IBM - Ex. 1002-Part 10 of 10, Page 3431
IPR2025-00379 (IBM v. Croga)

and backdoors, and it correctly allowed all legitimate network traffic. In our design, VMwall only performs introspection for the first packet of a new connection, so network performance remains high. Our tool adds only about 0–1 milliseconds of overhead to the first packet of a session. This is a latency cost to network connection creation that will not impact the subsequent data transfer of legitimate connections.

VMwall looks into the state of the untrusted operating system's memory to find the process bound to a network connection. The system monitors network flows, and it is not an intrustion detection system designed to detect an attack against the OS. Hence, an attacker may try to evade VMwall either by hijacking a process or by subverting the inspected kernel data structures. In Sect. 6.4, we study this problem, provide an in-depth security analysis of VMwall, and suggest appropriate measures to thwart these attacks.

We believe that our tamper-resistant application-oriented firewall represents an appropriate use of virtualization technology for improved system security. We feel that our paper provides the following contributions:

- Correlation between network flows and processes from outside the virtual machine (Sect. 4).
- VMwall, an implementation of a tamper-resistant application-oriented firewall (Sect. 5).
- Evidence that application-aware firewalls outside the untrusted virtual machine can block malicious network connections successfully while maintaining network performance (Sect. 6).

2 Related Work

Prior research has contributed to the development of conventional host-based firewalls. Mogul et al. [21] developed a kernel-resident packet filter for UNIX that gave user processes flexibility in selecting legitimate packets. Venema [29] designed a utility to monitor and control incoming network traffic. These traditional firewalls performed filtering based on restrictions inherent in network topology and assumed that all parties inside the network were trusted. As part of the security architecture of the computer system, they resided in kernel-space and user-space, and hence were vulnerable to direct attack by malicious software.

Administration of firewalls can be cumbersome, and distributed firewalls have been proposed to ease the burden [3,15]. In distributed firewalls, an administrator manages security policies centrally but pushes enforcement of these policies out to the individual hosts. Although we have not implemented support for distributed management, we expect VMwall to easily fit into this scheme. VMwall policies dictate which processes can legitimately make use of network resources. In a managed environment where administrators are knowledgeable of the software running on the machines in the local network, preparing and distributing VMwall policies from a central location may be an appealing solution.

IBM - Ex. 1002-Part 10 of 10, Page 3432
IPR2025-00379 (IBM v. Croga)

The recent support for virtual machines by commodity hardware has driven development of new security services deployed with the assistance of VMs [27, 9, 30]. Garfinkel et al. [11] showed the feasibility of implementing distributed network-level firewalls using virtual machines. In another work [10], they proposed an intrusion detection system design using virtual machine introspection of an untrusted VM. VMwall applies virtual machine introspection to a different problem, using it to correlate network flows with the local processes bound to those flows.

Other research used virtual machines for malware detection. Borders et al. [4] designed a system, Siren, that detected malware running within a virtual machine. Yin et al. [33] proposed a system to detect and analyze privacy-breaching malware using taint analysis. Jiang et al. [17] presented an out-of-the-box VMM-based malware detection system. Their proposed technique constructed the internal semantic views of a VM from an external vantage point. In another work [16], they proposed a monitoring tool that observes a virtual machine based honeypot's internal state from outside the honeypot. As a pleasant side-effect of malicious network flow detection and process correlation, VMwall can often identify processes in the untrusted system that comprise portions of an attack.

Previous research has developed protection strategies for different types of hardware-level resources in the virtualized environment. Xu et al. [32] proposed a VMM-based usage control model to protect the integrity of kernel memory. Ta-Min et al. [28] proposed a hypervisor based system that allowed applications to partition their system call interface into trusted and untrusted components. VMwall, in contrast, protects network resources from attack by malware that runs inside the untrusted virtual machine by blocking the illegitimate network connections attempts.

These previous hypervisor-based security applications generally take either a network-centric or host-centric view. Our work tries to correlate activity at both levels. VMwall monitors network connections but additionally peers into the state of the running, untrusted operating system to make its judgments about each connection's validity. Moreover, VMwall easily scales to collections of virtual machines on a single physical host. A single instance of VMwall can act as an application-level firewall for an entire network of VMs.

3 Overview

We begin with preliminaries. Section 3.1 explains our threat model, which assumes that attackers have the ability to execute the real-world attacks infecting widespread computer systems today. Section 3.2 provides a brief overview of Xen-based virtual machine architectures and methods allowing inspection of a running VM's state.

3.1 Threat Model

We assume that attackers have abilities commonly displayed by real-world attacks against commodity computer systems. Attackers can gain *superuser priv-*

IBM - Ex. 1002-Part 10 of 10, Page 3433
IPR2025-00379 (IBM v. Croga)

ileye from remote. Attackers are external and have no physical access to the attacked computers, but they may install malicious software on a victim system by exploiting a software vulnerability in an application or operating system or by enticing unsuspecting users to install the malware themselves. The software exploit or the user often executes with full system privileges, so the malware may perform administrative actions such as kernel module or driver installation. Hence, malicious code may execute at both user and kernel levels. For ease of explanation, we initially describe VMwall's architecture in Sect. 4 under the assumption that kernel data structure integrity is maintained. This assumption is not valid in our threat model, and Sect. 6.4 revisits this point to describe technical solutions ensuring that the assumption holds.

The installed malware may periodically make or receive network connections. Many examples exist. *Bots* make network connections to a command and control channel to advertise their presence and receive instruction, and they send bulk network traffic such as denial-of-service packets and email spam. *Spyware* programs collect information, such as keystrokes and mouse clicks, and then transmit the confidential data across a network to the attacker. *Worms* may generate network connections to scan the network in search of additional victims suitable for infection. *Backdoors* open holes in machines by listening for incoming connections from the attacker. One common feature of these different classes of attacks is their interest in the network.

In a typical system, malware can directly affect an application-level firewall's execution. The architecture of these malware instances frequently combines a user-level application performing network activity with a kernel-level module that hides the application from the view of host-level security software. The malicious application, likely running with full system privileges, may halt the execution of the firewall. Similarly, the malicious kernel component may alter the hooks used by an in-kernel module supporting the user-level firewall so that the firewall is simply never invoked as data passes to and from the network. Conventional application-level firewalls fail under these direct attacks. Our goal is to develop a system that withstands direct attack from malware at the application layer or the kernel layer.

Our system has requirements for correct execution. As with all requirements, an attacker who is able to violate any requirement is likely able to escape detection. Our two requirements of note center on basic expectations for the in-memory data structures used by the kernel that may be infected by an attack.

First, we expect to be able to find the head of linked data structures, often by extracting a kernel symbol value at boot time. An attacker could conceivably cause our firewall to inspect the incorrect kernel information by replicating the data structure elsewhere in kernel memory and by altering all code references to the original structure to instead refer to the new structure. Our firewall would then analyze stale data. It is not immediately clear that such an attack is plausible; moreover, our tool could periodically verify that code references to the data match the symbol value extracted at boot.

IBM - Ex. 1002-Part 10 of 10, Page 3434
IPR2025-00379 (IBM v. Croga)

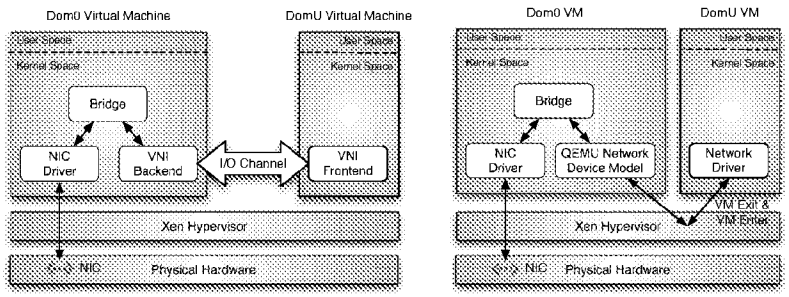


Fig. 1. Xen networking architecture

Second, we expect that attacks do not alter the ordering or length of fields in aggregate data structures. Our firewall is preprogrammed with type information about kernel structures, and an attack that alters the structure types would cause our system to read incorrect information from kernel memory. Successfully executing this attack without kernel recompilation appears to be complex, as all kernel code that accesses structure fields would need to be altered to use the attacker's structure layout. As a result, we believe that relying upon known structure definitions is not a limiting factor to our design.

3.2 Virtual Machine Introspection

Our design makes use of virtual machine technology to provide isolation between malicious code and our security software. We use Xen [2], an open source hypervisor that runs directly on the physical hardware of a computer. The virtual machines running atop Xen are of two types: unprivileged domains, called domU or guest domains, and a single fully-privileged domain, called dom0. We run normal, possibly vulnerable software in domU and deploy our application-level firewall in the isolated dom0.

Xen virtualizes the network input and output of the system. Dom0 is the device driver domain that runs the native network interface card driver software. Unprivileged virtual machines cannot directly access the physical network card, so Xen provides them with a virtualized network interface (VNI). The driver domain receives all the incoming and outgoing packets for all domU VMs executing on the physical system. Dom0 provides an Ethernet bridge connecting the physical network card to all virtual network devices provided by Xen to the domU VMs. (Xen offers other networking modes, such as network address translation, that are not used in our work and will not be considered further.) Dom0 uses its virtual bridge to multiplex and demultiplex packets between the physical network interface and each unprivileged virtual machine's VNI. Figure 1 shows the Xen networking architecture when the virtual machines' network interfaces are

IBM - Ex. 1002-Part 10 of 10, Page 3435
IPR2025-00379 (IBM v. Croga)

connected through a virtual Ethernet bridge. The guest VMs send and receive packets via either an I/O channel to dom0 or emulated virtual devices.

The strong isolation provided by a hypervisor between dom0 and the guest domains complicates the ability to correlate network flows with software executing in a guest domain. Yet, dom0 has complete access to the entire state of the guest operating systems running in untrusted virtual machines. *Virtual machine introspection* (VMI) [10] is a technique by which dom0 can determine execution properties of guest VMs by monitoring their runtime state, generally through direct memory inspection. VMI allows security software to remain protected from direct attack by malicious software executing in a guest VM while still able to observe critical system state.

Xen offers low-level APIs to allow dom0 to map arbitrary memory pages of domU as shared memory. XenAccess [31] is a dom0 userspace introspection library developed for Xen that builds onto the low-level functionality provided by Xen. VMwall uses XenAccess APIs to map raw memory pages of domU's kernel inside dom0. It then builds higher-level memory abstractions, such as aggregate structures and linked data types, from the contents of raw memory pages by using the known coding semantics of the guest operating system's kernel. Our application-level firewall inspects these meaningful, higher-level abstractions to determine how applications executing in the guest VM use network resources.

4 Tamper Resistant Architecture of VMwall

VMwall is our application-level firewall designed to resist the direct attacks possible in our threat model. The architecture of VMwall is driven by the following three goals:

- **Tamper Resistance:** VMwall should continue to function reliably and verify all network connections even if an attacker gains entry into the monitored system. In particular, the design should not rely on components installed in the monitored host as processes or kernel modules, as these have been points of direct attack in previous application-level firewalls.
- **Independence:** VMwall should work without any cooperation from the monitored system. In fact, the system may not be aware of the presence of the firewall.
- **Lightweight Verification:** Our intent is to use VMwall for online verification of network connections to real systems. The design should allow for efficient monitoring of network traffic and correlation to applications sending and receiving that traffic.

Our firewall design satisfies these goals by leveraging virtual machine isolation and virtual machine introspection. Its entire software runs within the privileged dom0 VM, and it hooks into Xen's virtual network interface to collect and filter all guest domains' network packets. Since the hypervisor provides strong isolation among the virtual machines, this design achieves the first goal of tamper-resistance.

IBM - Ex. 1002-Part 10 of 10, Page 3436
IPR2025-00379 (IBM v. Croga)

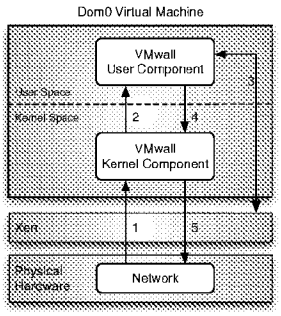


Fig. 2. VMwall's high-level architecture. (1) Packets inbound to and outbound from a guest domain are processed by dom0. (2) The VMwall kernel component intercepts the packets and passes them to a user-space component for analysis. (3) The user-space component uses virtual machine introspection to identify software in a guest domain processing the data. (4) The user-space component instructs the kernel component to either allow or block the connection. (5) Packets from allowed connections will be placed on the network.

In order to provide application-level firewalling, VMwall must identify the process that is sending or receiving packets inside domU. VMwall correlates packet and process information by directly inspecting the domU virtual machine's memory via virtual machine introspection. It looks into the kernel's memory and traverses the data structures to map process and network information. This achieves our second design goal of independence, as there are no components of VMwall inside domU. Our introspection procedure rapidly analyzes the kernel's data structures, satisfying the third goal of lightweight verification.

The high-level design of VMwall has two components: a kernel module and user agent, both in dom0 (Fig. 2). The VMwall kernel component enforces a per-packet policy given by the user agent and either allows or drops each packet. The user agent determines policy by performing introspection to extract information about processes executing in guest VMs and evaluating the legitimacy of those processes. Sections 4.1 and 4.2 present detailed information about the two components.

4.1 Kernel Component

VMwall's kernel component is a module loaded inside the dom0 Linux kernel. It intercepts all network packets to or from untrusted virtual machines and uses security policies to decide whether each packet should be allowed or dropped. Interception occurs by hooking into Xen's network bridge between the physical interface card and virtual network interface. When the kernel component intercepts a packet, it checks a rule table to see if a firewall rule already exists for the packet, as determined by the local endpoint IP address and port. If so, it takes the allow or block action specified in the rule. If there is no rule, then it

IBM - Ex. 1002-Part 10 of 10, Page 3437
IPR2025-00379 (IBM v. Croga)

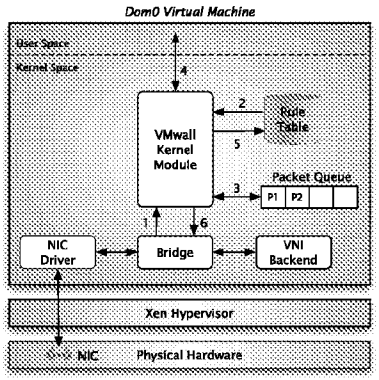


Fig. 3. VMwall's kernel module architecture. (1) Packets inbound to and outbound from a guest domain are intercepted and passed to the kernel module. (2) The module receives each packet and looks into its rule table to find the rule for the packet. (3) The kernel module queues the packet if there is no rule present. (4) VMwall sends an introspection request to the user agent and, after the agent completes, receives the dynamically generated rule for the packet. (5) The kernel module adds the rule into its rule table to process future packets from the same connection. (6) The kernel module decides based on the action of the rule either to accept the packet by reinjecting it into the network or to drop it from the queue.

invokes the VMwall user agent to analyze the packet and create a rule. The user agent performs introspection, generates a rule for the packet, and sends this rule back to the kernel module. The kernel module adds this new rule to its policy table and processes the packet. Further packets from the same connection are processed using the rule present in the kernel component without invoking the user agent and without performing introspection.

As kernel code, the kernel component cannot block and must take action on a packet before the user agent completes introspection. VMwall solves this problem for packets of unknown legitimacy by queuing the packets while waiting for the user agent's reply. When the user agent sends a reply, the module adds a rule for the connection. If the rule's action is to block the connection, then it drops all the packets that are queued. Otherwise, it re-injects all the packets into the network.

Figure 3 presents the kernel module's complete architecture. It illustrates the steps involved in processing the packet inside the kernel. It shows the queue architecture, where packets are stored inside the kernel during introspection.

4.2 User Agent

The VMwall user agent uses virtual machine introspection to correlate network packets and processes. It receives introspection requests from the kernel component containing network information such as source port, source IP address,

IBM - Ex. 1002-Part 10 of 10, Page 3438
IPR2025-00379 (IBM v. Croga)

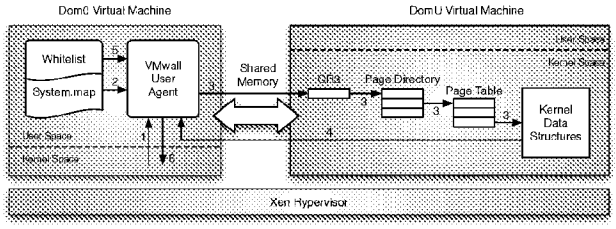


Fig. 4. VMwall's user agent architecture. (1) The VMwall user agent receives the introspection request. (2) The user agent reads the System.map file to extract the kernel virtual addresses corresponding to known kernel symbols. (3) The user agent uses Xen to map the domU kernel memory pages containing process and network data structures. (4) VMwall traverses the data structures to correlate network and process activity. (5) The agent searches for the recovered process name in the whitelist. (6) The user agent sends a filtering rule for the connection to the VMwall kernel module.

destination port, destination IP address, and protocol. It first uses the packet's source (or destination) IP address to identify the VM that is sending (or receiving) the packet. When it finds the VM, it then tries to find the process that is bound to the source (or destination) port.

VMwall's user agent maps a network port to the domU process that is bound to the port, shown in Fig. 4. As needed, it maps domU kernel data structures into dom0 memory. Process and network information is likely not available in a single data structure but instead is scattered over many data structures. VMwall works in steps by first identifying the domU kernel data structures that store IP address and port information. Then, VMwall identifies the process handling this network connection by iterating over the list of running processes and checking each process to see if it is bound to the port. When it finds the process bound to the port, it extracts the process' identifier, its name, and the full path to its executable. If the user agent does not find any process bound to the port, it considers this to be an anomaly and will block the network connection.

VMwall uses information about the process to create a firewall rule enforceable by the kernel component. The user agent maintains a whitelist of processes that are allowed to make network connections. When the user agent extracts the name of a process corresponding to the network packet, it searches the whitelist for the same name. VMwall allows the connection if it finds a match and blocks the connection otherwise. It then generates a rule for this connection that it passes to the VMwall kernel component. This rule contains the network connection information and either an allow or block action. The kernel component then uses this rule to filter subsequent packets in this attempted connection.

5 Implementation

We have implemented a prototype of VMwall using the Xen hypervisor and a Linux guest operating system. VMwall supports both paravirtualized and

IBM - Ex. 1002-Part 10 of 10, Page 3439
IPR2025-00379 (IBM v. Croga)

fully-virtualized (HVM) Linux guest operating systems. Its implementation consists of two parts corresponding to the two pieces described in the previous section: the kernel module and the user agent. The following sections describe specific details affecting implementation of the two architectural components.

5.1 Extending Ebtables

Our kernel module uses a modified ebtables packet filter to intercept all packets sent to or from a guest domain. Ebtables [7] is an open source utility that filters packets at an Ethernet bridge. VMwall supplements the existing coarse-grained firewall provided by ebtables. Whenever ebtables accepts packets based on its coarse-grained rules, we hook the operation and invoke the VMwall kernel module for our additional application-level checks. We modified ebtables to implement this hook, which passes a reference to the packet to VMwall.

Ebtables does not provide the ability to queue packets. Were it present, queuing would enable filters present inside the kernel to store packets for future processing and reinjection back into the network. To allow the VMwall kernel module to queue packets currently under inspection by the user agent, we altered ebtables to incorporate packet queuing and packet reinjection features.

5.2 Accessing DomU Kernel Memory

VMwall uses the XenAccess introspection library [31] to accesses domU kernel memory from dom0. It maps domU memory pages containing kernel data structures into the virtual memory space of the user agent executing in the trusted VM. XenAccess provides APIs that map domU kernel memory pages identified either by explicit kernel virtual addresses or by exported kernel symbols. In Linux, the exported symbols are stored in the file named `System.map`. VMwall uses certain domU data structures that are exported by the kernel and hence mapped with the help of kernel symbols. Other data structures reachable by pointers from the known structures are mapped using kernel virtual addresses. The domU virtual machine presented in Fig. 4 shows the internal mechanism involved to map the memory page that contains the desired kernel data structure.

5.3 Parsing Kernel Data Structures

To identify processes using the network, VMwall must be able to parse high-level kernel data structures from the raw memory pages provided by XenAccess. Extracting kernel data structures from the mapped memory pages is a non-trivial task. For example, Linux maintains a doubly-linked list that stores the kernel's private data for all running processes. The head pointer of this list is stored in the exported kernel symbol `init_task`. If we want to extract the list of processes running inside domU, we can map the memory page of domU that contains the `init_task` symbol. However, VMwall must traverse the complete linked list and hence requires the offset to the `next` member in the process structure. We extract this information offline directly from the kernel source code and use these values

IBM - Ex. 1002-Part 10 of 10, Page 3440
IPR2025-00379 (IBM v. Croga)

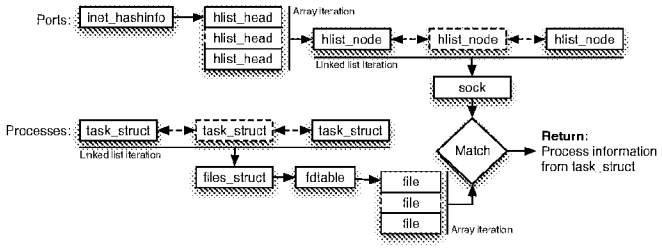


Fig. 5. DomU Linux kernel data structures traversed by the VMwall user agent during correlation of the process and TCP packet information

in the user agent. This source code inspection is not the optimal way to identify offsets because the offset values often change with the kernel versions. However, there are other automatic ways to extract this information from the kernel binary if it was compiled with a debug option [18].

This provides VMwall with sufficient information to traverse kernel data structures. VMwall uses known field offsets to extract the virtual addresses of pointer field members from the mapped memory pages. It then maps domU memory pages by specifying the extracted virtual addresses. This process is performed recursively until VMwall traverses the data structures necessary to extract the process name corresponding to the source or destination port of a network communication. Figure 5 shows the list of the kernel data structures traversed by the user agent to correlate a TCP packet and process information. First, it tries to obtain a reference to the socket bound to the port number specified in the packet. After acquiring this reference, it iterates over the list of processes to find the process owning the socket.

5.4 Policy Design and Rules

VMwall identifies legitimate connections via a whitelist-based policy listing processes allowed to send or receive data. Each process that wants to communicate over the network must be specified in the whitelist *a priori*. This whitelist resides inside dom0 and can only be updated by administrators in a manner similar to traditional application-level firewalls. The whitelist based design of VMwall introduces some usability issues because all applications that should be allowed to make network connections must be specified in the list. This limitation is not specific to VMwall and is inherent to the whitelist based products and solutions [6, 12].

VMwall's kernel module maintains its own rule table containing rules that are dynamically generated by the user agent after performing introspection. A rule contains source and destination port and IP address information, an action, and a timeout value used by the kernel module to expire and purge old rules for UDP connections. In the case of TCP connections, the kernel module purges TCP rules automatically whenever it processes a packet with the TCP `fin` or

IBM - Ex. 1002-Part 10 of 10, Page 3441
IPR2025-00379 (IBM v. Croga)

`rst` flag set. In an abnormal termination of a TCP connection, VMwall uses the timeout mechanism to purge the rules.

6 Evaluation

The basic requirement of an application-level firewall is to block connections to or from malicious software and allow connections to or from benign applications. We evaluated the ability of VMwall to filter out packets made by several different classes of attacks while allowing packets from known processes to pass unimpeded. We tested VMwall against Linux-based backdoors, worms, and bots that attempt to use the network for malicious activity. Section 6.1 tests VMwall against attacks that receive inbound connections from attackers or connect out to remote systems. Section 6.2 tests legitimate software in the presence of VMwall. We measure VMwall's performance impact in Sect. 6.3, and lastly analyze its robustness to a knowledgeable attacker in Sect. 6.4.

6.1 Illegitimate Connections

We first tested attacks that receive inbound connections from remote attackers. These attacks are rootkits that install backdoor programs. The backdoors run as user processes, listen for connections on a port known to the attacker, and receive and execute requests sent by the attacker. We used the following backdoors:

- **Blackhole** runs a TCP server on port 12345 [22].
- **Gummo** runs a TCP server at port 31337 [22].
- **Bdoor** runs a backdoor daemon on port 8080 [22].
- **Ovas0n** runs a TCP server on port 29369 [22].
- **Cheetah** runs a TCP server at the attacker's specified port number [22].

Once installed on a vulnerable system, attacks such as worms and bots may attempt to make outbound connections without prompting from a remote attacker. We tested VMwall with the following pieces of malware that generate outbound traffic:

- **Apache-ssl** is a variant of the Slapper worm that self-propagates by opening TCP connections for port scanning [23].
- **Apache-linux** is a worm that exploits vulnerable Apache servers and spawns a shell on port 30464 [23].
- **BackDoor-Rev.b** is a tool that is be used by a worm to make network connections to arbitrary Internet addresses and ports [20].
- **Q8** is an IRC-based bot that opens TCP connections to contact an IRC server to receive commands from the botmaster [14].
- **Kaiten** is a bot that opens TCP connections to contact an IRC server [24].
- **Coromputer Dunno** is an IRC-based bot providing basic functionalities such as port scanning [13].

IBM - Ex. 1002-Part 10 of 10, Page 3442
IPR2025-00379 (IBM v. Croga)

Table 1. Results of executing legitimate software in the presence of VMwall. "Allowed" indicates that the network connections to or from the processes were passed as though a firewall was not present.

Name	*Connection Type*	*Result*
rcp	Outbound	Allowed
rsh	Outbound	Allowed
yum	Outbound	Allowed
rlogin	Outbound	Allowed
ssh	Outbound	Allowed
scp	Outbound	Allowed
wget	Outbound	Allowed
tcp client	Outbound	Allowed
thttpd	Inbound	Allowed
tcp server	Inbound	Allowed
sshd	Inbound	Allowed

VMwall successfully blocked all illegitimate connections attempted by malware instances. In all cases, both sending and receiving, VMwall intercepted the first SYN packet of each connection and passed it to the userspace component. Since these malicious processes were not in the whitelist, the VMwall user space component informed the VMwall kernel component to block these malicious connections. As we used VMwall in packet queuing mode, no malicious packets were ever passed through VMwall.

6.2 Legitimate Connections

We also evaluated VMwall's ability to allow legitimate connections made by processes running inside domU. We selected a few network applications and added their name to VMwall's whitelist. We then ran these applications inside domU. Table 1 shows the list of processes that we tested, the type of connections used by the processes, and the effect of VMwall upon those connections. To be correct, all connections should be allowed.

VMwall allowed all connections made by these applications. The `yum` application, a package manager for Fedora Core Linux, had runtime behavior of interest. In our test, we updated domU with the `yum update` command. During the package update, `yum` created many child processes with the same name `yum` and these child processes made network connections. VMwall successfully validated all the connections via introspection and allowed their network connections.

6.3 Performance Evaluation

A firewall verifying all packets traversing a network may impact the performance of applications relying on timely delivery of those packets. We investigated the performance impact of VMwall as perceived by network applications running inside the untrusted virtual machine. We performed experiments both with and

IBM - Ex. 1002-Part 10 of 10, Page 3443
IPR2025-00379 (IBM v. Croga)

Table 2. Introspection time (μs) taken by VMwall to perform correlation of network flow with the process executing inside domU

Configuration	*TCP Introspection Time*	*UDP Introspection Time*
Inbound Connection to domU	251	438
Outbound Connection from domU	1080	445

without VMwall running inside dom0. All experiments were conducted on a machine with an Intel Core 2 Duo T7500 processor at 2.20 GHz with 2 GB RAM. Both dom0 and domU virtual machines ran 32 bit Fedora Core 5 Linux. DomU had 512 MB of physical memory, and dom0 had the remaining 1.5 GB. The versions of Xen and XenAccess were 3.0.4 and 0.3, respectively. We performed our experiments using both TCP and UDP connections. All reported results show the median time taken from five measurements. We measured microbenchmarks with the Linux `gettimeofday` system call and longer executions with the `time` command-line utility.

VMwall's performance depends on the introspection time taken by the user component. Since network packets are queued inside the kernel during introspection, the introspection time is critical for the performance of the complete system. We measured the introspection time both for incoming and outgoing connections to and from domU. Table 2 shows the results of experiments measuring introspection time.

It is evident that the introspection time for incoming TCP connections is very small. Strangely, the introspection time for outgoing TCP connections is notably higher. The reason for this difference lies in the way that the Linux kernel stores information for TCP connections. It maintains TCP connection information for listening and established connections in two different tables. TCP sockets in a listening state reside in a table of size 32, whereas the established sockets are stored in a table of size 65536. Since the newly established TCP sockets can be placed at any index inside the table, the introspection routine that iterates on this table from dom0 must search half of the table on average.

We also measured the introspection time for UDP data streams. Table 2 shows the result for UDP inbound and outbound packets. In this case, the introspection time for inbound and outbound data varies little. The Linux kernel keeps the information for UDP streams in a single table of size 128, which is why the introspection time is similar in both cases.

To measure VMwall's performance overhead on network applications that run inside domU, we performed experiments with two different metrics for both inbound and outbound connections. In the first experiment, we measured VMwall's impact on network I/O by transferring a 175 MB video file over the virtual network via `wget`. Our second experiment measured the time necessary to establish a TCP connection or transfer UDP data round-trip as perceived by software in domU.

We first transferred the video file from dom0 to domU and back again with VMwall running inside dom0. Table 3 shows the result of our experiments. The

IBM - Ex. 1002-Part 10 of 10, Page 3444
IPR2025-00379 (IBM v. Croga)

Table 3. Time (seconds) to transfer a 175 MB file between dom0 and domU, with and without VMwall

Direction	*Without VMwall*	*With VMwall*	*Overhead*
File Transfer from Dom0 to DomU	1.105	1.179	7%
File Transfer from DomU to Dom0	1.133	1.140	1%

Table 4. Single TCP connection setup time (μs) measured both with and without VMwall inside dom0

Direction	*Without VMwall*	*With VMwall*	*Overhead*
Connection from Dom0 to DomU	197	465	268
Connection from DomU to Dom0	143	1266	1123

median overhead imposed by VMwall is less than 7% when transferring from dom0 to domU, and less than 1% when executing the reverse transfer.

Our second metric evaluated the impact of VMwall upon connection or data stream setup time as perceived by applications executing in domU. For processes using TCP, we measured both the inbound and outbound TCP connection setup time. For software using UDP, we measured the time to transfer a small block of data to a process in the other domain and to have the block echoed back.

We created a simple TCP client-server program to measure TCP connection times. The client program measured the time required to connect to the server, shown in Table 4. Inbound connections completed quickly, exhibiting median overhead of only 268 μs. Outbound connections setup from domU to dom0 had a greater median overhead of 1123 μs, due directly to the fact that the introspection time for outbound connections is also high. Though VMwall's connection setup overhead may look high as a percentage, the actual overhead remains slight. Moreover, the introspection cost occurring at connection setup is a one-time cost that gets amortized across the duration of the connection.

We lastly measured the time required to transmit a small block of data and receive an echo reply to evaluate UDP stream setup cost. We wrote a simple UDP echo client and server and measured the round-trip time required for the echo reply. Note that only the first UDP packet required introspection; the echo reply was rapidly handled by a rule in the VMwall kernel module created when processing the first packet. We again have both inbound and outbound measurements, shown in Table 5. The cost of VMwall is small, incurring slowdowns of 381 μs and 577 μs, respectively.

VMwall currently partially optimizes its performance, and additional improvements are clearly possible. VMwall performs introspection once per connection so that further packets from the same connection are allowed or blocked based on the in-kernel rule table. VMwall's performance could be improved in future work by introducing a caching mechanism to the introspection operation. The VMwall introspection routine traverses the guest OS data structures to perform correlation. In order to traverse a data structure, the memory page that contains the data structure needs to be mapped, which is a costly operation. One possi-

IBM - Ex. 1002-Part 10 of 10, Page 3445
IPR2025-00379 (IBM v. Croga)

Table 5. Single UDP echo-reply stream setup time (μs) with and without VMwall. In an inbound-initiated echo, dom0 sent data to domU and domU echoed the data back to dom0. An outbound-initiated echo is the reverse.

Direction	*Without VMwall*	*With VMwall*	*Overhead*
Inbound Initiated	434	815	381
Outbound Initiated	271	848	577

ble improvement would be to support caching mechanisms inside VMwall's user agent to cache frequently used memory pages to avoid costly memory mapping operations each time.

6.4 Security Analysis

VMwall relies on particular data structures maintained by the domU kernel. An attacker who fully controls domU could violate the integrity of these data structures in an attempt to bypass VMwall's introspection. To counter such attacks, we rely on previous work in kernel integrity protection. Petroni et al. [26] proposed a framework for detecting attacks against dynamic kernel data structures such as `task_struct`. Their monitoring system executed outside the monitored kernel and detected any semantic integrity violation against the kernel's dynamic data. The system protected the integrity of the data structures with an external monitor that enforced high-level integrity policies. In another work, Loscocco et al. [19] introduced a system that used virtualization technology to monitor a Linux kernel's operational integrity. These types of techniques ensure that the kernel data structures read by VMwall remain valid.

Attackers can also try to cloak their malware by appearing to be whitelisted software. An attacker can guess processes that are in VMwall's whitelist by observing the incoming and outgoing traffic from the host and determining themselves what processes legally communicate over the network. They can then rename their malicious binary to the name of a process in the whitelist. VMwall counters this problem by extracting the full path to the process on the guest machine. Attackers could then replace the complete program binary with a trojaned version to evade the full path verification. VMwall itself has no defenses against this attack, but previous research has already addressed this problem with disk monitoring utilities that protect critical files [8, 25].

An attacker could hijack a process by exploiting a vulnerability, and they could then change its in-memory image. To address this problem, VMwall user-space process can perform checksumming of the in-memory image of the process through introspection and compare it with previously stored hash value. However, this process is time consuming and may affect the connection setup time for an application.

An attacker could also hijack a connection after it has been established and verified by VMwall as legitimate. They could take control of the process bound to the port via a software exploit, or they could use a malicious kernel module to

IBM - Ex. 1002-Part 10 of 10, Page 3446
IPR2025-00379 (IBM v. Croga)

alter packet data before sending it to the virtual network interface. VMwall can counter certain instances of connection hijacking by timing out entries in its kernel rule table periodically. Subtle hijacking may require deep packet inspection within VMwall.

VMwall's kernel module internally maintains a small buffer to keep a copy of a packet while performing introspection. An attacker may try to launch a denial of service (DoS) attack, such as a `SYN` flood [5], against VMwall by saturating its internal buffer. VMwall remains robust to such attempted attacks because its buffer is independent of connection status. As soon as VMwall resolves the process name bound to a connection, it removes the packet from the buffer and does not wait for a TCP handshake to complete.

7 Conclusions and Future Work

We set out to design an application-oriented firewall resistant to the direct attacks that bring down these security utilities today. Our system, VMwall, remains protected from attack by leveraging virtual machine isolation. Although it is a distinct virtual machine, it can recover process-level information of the vulnerable system by using virtual machine introspection to correlate network flows with processes bound to those flows. We have shown the efficacy of VMwall by blocking backdoor, bot, and worm traffic emanating from the monitored system. Our malicious connection detection operates with reasonable overheads upon system performance.

Our current implementation operates for guest Linux kernels. VMwall could be made to work with Microsoft Windows operating systems if it can be programmed with knowledge of the data structures used by the Windows kernel. Since VMwall depends on the guest operating system's data structures to perform network and process correlation, it currently cannot be used for Windows-based guest systems. Recently, XenAccess started providing the ability to map Windows kernel memory into dom0 in the same way as done for Linux. If we have a means to identify and map Windows kernel data structures, then network and process correlation becomes possible.

Acknowledgment of Support and Disclaimer. This material is based upon work supported by the Defense Advanced Research Projects Agency and the United States Air Force under contract number FA8750-06-C-0182. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency and the United States Air Force.

We thank the anonymous reviewers for their comments that improved the quality of the paper. We thank Steve Dawson of SRI International for his assistance with this project. Portions of this work were performed while Abhinav Srivastava was at SRI International.

IBM - Ex. 1002-Part 10 of 10, Page 3447
IPR2025-00379 (IBM v. Croga)

References

[1] Baliga, A., Kamat, P., Iftode, L.: Lurking in the shadows: Identifying systemic threats to kernel data. In: IEEE Symposium on Security and Privacy, Oakland, CA (May 2007)

[2] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: 19th ACM Symposium on Operating Systems Principles (SOSP), Bolton Landing, NY (October 2003)

[3] Bellovin, S.: Distributed firewalls. login (November 1999)

[4] Borders, K., Zhao, X., Prakash, A.: Siren: Catching evasive malware. In: IEEE Symposium on Security and Privacy, Oakland, CA (May 2005)

[5] CERT. TCP SYN Flooding and IP Spoofing Attacks. CERT Advisory CS-1996-21 (Last accessed April 4 , 2008), http://www.cert.org/advisories/CA-1996-21.html

[6] Check Point. ZoneAlarm (Last accessed April 4, 2008), http://www.zonealarm.com/store/content/home.jsp

[7] Community Developers. Ebtables (Last accessed November 1, 2007), http://ebtables.sourceforge.net/

[8] Community Developers. Tripwire (Last accessed November 1, 2007), http://sourceforge.net/projects/tripwire/

[9] Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A virtual machine-based platform for trusted computing. In: ACM Symposium on Operating Systems Principles (SOSP), October 2003, Bolton Landing, NY (2003)

[10] Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feburary (2003)

[11] Garfinkel, T., Rosenblum, M., Boneh, D.: Flexible OS support and applications for trusted computing. In: 9th Hot Topics in Operating Systems (HOTOS), Lihue, HI (May 2003)

[12] Oskoboiny, G.: Whitelist-based spam filtering (Last accessed April 4, 2008), http://impressive.net/people/gerald/2000/12/spam-filtering.html

[13] Grok. Coromputer Dunno (Last accessed April 4, 2008), http://lists.grok.org.uk/pipermail/full-disclosure/attachments/20070911/87396911/attachment-0001.txt

[14] Honeynet Project. Q8 (Last accessed April 4, 2008), http://www.honeynet.org/papers/bots/

[15] Ioannidis, S., Keromytis, A., Bellovin, S., Smith, J.: Implementing a distributed firewall. In: ACM Conference on Computer and Communications Security (CCS), Athens, Greece (November 2000)

[16] Jiang, X., Wang, X.: Out-of-the-box monitoring of VM-based high-interaction honeypots. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 198–218. Springer, Heidelberg (2007)

[17] Jiang, X., Wang, X., Xu, D.: Stealthy malware detection through VMM-based 'out-of-the-box' semantic view. In: 14th ACM Conference on Computer and Communications Security (CCS), Alexandria, VA (November 2007)

[18] LKCD Project. LKCD - Linux Kernel Crash Dump (Last accessed April 4, 2008), http://lkcd.sourceforge.net/

[19] Loscocco, P.A., Wilson, P.W., Pendergrass, J.A., McDonell, C.D.: Linux kernel integrity measurement using contextual inspection. In: 2nd ACM Workshop on Scalable Trusted Computing (STC), Alexandria, VA (November 2007)

IBM - Ex. 1002-Part 10 of 10, Page 3448
IPR2025-00379 (IBM v. Croga)

[20] McAfee. BackDoor-Rev.b. (Last accessed April 4, 2008), http://vil.nai.com/vil/Content/v_136510.htm
[21] Mogul, J., Rashid, R., Accetta, M.: The packet filter: An efficient mechanism for user-level network code. In: ACM Symposium on Operating Systems Principles (SOSP), Austin, TX (November 1987)
[22] Packet Storm (Last accessed April 4, 2008). http://packetstormsecurity.org/UNIX/penetration/rootkits/bdoor.c,blackhole.c,cheetah.c,server.c,ovasOn.c
[23] Packet Storm (Last accessed April 4, 2008), http://packetstormsecurity.org/0209-exploits/apache-ssl-bug.c,apache-linux.txt
[24] Packet Storm. Kaiten (Last accessed April 4, 2008), http://packetstormsecurity.org/irc/kaiten.c
[25] Payne, B.D., Carbone, M., Lee, W.: Secure and flexible monitoring of virtual machines. In: 23rd Annual Computer Security Applications Conference (ACSAC), Miami, FL (December 2007)
[26] Petroni Jr., N.L., Fraser, T., Walters, A., Arbaugh, W.A.: An architecture for specification-based detection of semantic integrity violations in kernel dynamic data. In: 15th USENIX Security Symposium, Vancouver, BC, Canada (August 2006)
[27] Petroni Jr., N.L., Hicks, M.: Automated detection of persistent kernel control-flow attacks. In: 14th ACM Conference on Computer and Communications Security (CCS), Alexandria, VA (November 2007)
[28] Ta-Min, R., Litty, L., Lie, D.: Splitting interfaces: Making trust between applications and operating systems configurable. In: Symposium on Operating System Design and Implementation (OSDI), Seattle, WA (October 2006)
[29] Venema, W.: TCP wrapper: Network monitoring, access control and booby traps. In: USENIX UNIX Security Symposium, Baltimore, MD (September 1992)
[30] Whitaker, A., Cox, R.S., Shaw, M., Gribble, S.D.: Constructing services with interposable virtual hardware. In: 1st Symposium on Networked Systems Design and Implementation (NSDI), San Francisco, CA (March 2004)
[31] XenAccess Project. XenAccess Library (Last accessed April 4, 2008), http://xenaccess.sourceforge.net/
[32] Xu, M., Jiang, X., Sandhu, R., Zhang, X.: Towards a VMM-based usage control framework for OS kernel integrity protection. In: 12th ACM Symposium on Access Control Models and Technologies (SACMAT), Sophia Antipolis, France (June 2007)
[33] Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis. In: ACM Conference on Computer and Communications Security (CCS), Arlington, VA (October 2007)

IBM - Ex. 1002-Part 10 of 10, Page 3449
IPR2025-00379 (IBM v. Croga)

International Journal of Science & Technology
www.ijst.co.in

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

Impact & Analysis of Virtual Workspace on Grid Computing

*Prashant Singh Yadav[1], *Pankaj Sharma[2], Dr. D. Singh[3]*
Associate Prof. Deptt. Of CSE VIMT Ghaziabd, U.P, India
Assistant Prof. Deptt. Of CSE VCE Vijnore, U.P, India
Director Vedant Institute of Management & Technology, Ghaziabad, U.P, India

Abstract

Despite significant progress in the development of Grid infrastructure, the provisioning of a customized and controllable remote execution environment remains an open issue. This paper introduces the concept of a virtual workspace, a configurable execution environment that can be created and managed as a first-class entity to reflect client requirements. Such workspaces can be dynamically deployed on a variety of resources decoupling the notion of environment and resource. We show how virtual workspaces fit into the Grid architecture, present an example implementation using virtual machines, and discuss our initial experiences using this system in practice and with applications.

1. Introduction

While significant progress has been achieved with the deployment of Grid-based applications, the preparation of a remote execution environment remains an issue. One of the reasons is that while Grids offer access to diverse software environments, an application typically requires a very specific, customized environment. As a consequence of variations in operating system, middleware versions, library environments, and file system layouts a user's application may in practice be able to use only a small fraction of the resources potentially available on the Grid. The second issue is then need to provide reliable isolation and dynamic, fine-grain control of share resources to ensure enforcement of policies and thus provide incentive for wider sharing. The development of Grid protocols provides uniform ways to manage Grid entities that could represent such environments. It now remains to find ways to describe represent and implement them.

Despite significant progress in the development of Grid infrastructure, the provisioning of a customized and controllable remote execution environment remains an open issue. This paper introduces the concept of a virtual workspace, a configurable execution environment that can be created and managed as a first-class entity to reflect client requirements. Such workspaces can be dynamically deployed on a variety of resources decoupling the notion of environment and resource. We show how virtual workspaces fit into the Grid architecture, present an example implementation using virtual machines, and discuss our initial experiences using this system in practice and with applications. In this paper, we present the concept of a virtual workspace, which allows a Grid client to define an environment in terms of its requirements (such as resource requirements or software configuration), manage it, and then deploy the environment in the Grid. Workspaces defined in this way can be implemented in a variety of ways such as for example dynamically creating Unix accounts and using system as well as software configuration tools to enforce the required properties. Here, we focus on a particularly promising implementation of virtual

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3450
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

workspaces: virtual machines (VMs). The use of virtual machines in Grid computing has been proposed before. In addition to outstanding isolation properties, VMs can provide fine-grained enforcement; and by their very nature—virtualization of the underlying hardware—they enable instantiation of a new, independently configured guest environment on a host resource. They can be rapidly suspended and their state serialized, and thus easily migrated to remote resources. Moreover, as a result of recent progress in virtual machine technology, these advantages no longer come at a performance cost to either the application or the hosting resource: systems such as Xen demonstrate that they can be used with little or no performance degradation. In the rest of the paper, we describe how virtual workspaces fit into the Grid architecture, present a prototype of this architecture based on the Globus Toolkit (GT) and experiment with two workspace implementations using Xen and the VMware Workstation. We describe our initial experiences with integrating VMs into the Grid infrastructure, and we present preliminary results of testing our prototype system with a bioinformatics application suite.

2. Virtual Workspaces

Interactions in present-day Grids focus on mapping jobs to resources, often with the assumption that an execution environment with suitable configuration and enforcement characteristics will be provided by means independent of the Grid infrastructure. Although such an assumption is true for closely knit groups of Grid users, it is not justified when applications or users with drastically different requirements and rights are trying to use the same resources. Recognizing this fact, we define the concept of a *virtual workspace* that can be automatically deployed on resources and provide a required execution environment. Thus, jobs can be mapped to workspaces, and workspaces can be mapped to actual resources in the Grid.

A virtual workspace (VW) is a definition of an execution environment in terms of its hardware requirements, software configuration, isolation properties, and other salient characteristics. The intent of defining a workspace is to capture the requirements for an execution environment in the Grid and then use automated tools in order to find, configure, and provide an environment best matching those requirements. We could, for example, use agreement-based tools to negotiate contracts defining workspaces and the use of actual resources and then negotiate a binding between them following the models described in. Depending on the requirements, such contracts could be fulfilled by simply dynamically creating and configuring user accounts as in, by using pre-configured virtual machines or other sandboxing and virtualization technologies.

A. Virtual Workspace Descriptions

To describe workspaces, we use an XML Schema, which captures generic properties of every workspace, as well as properties subject to specific workspace definition or workspace implementation-specific properties. The XML example below shows a description of a workspace. It contains the workspace category type that describes what mechanisms are used to create the workspace: we currently support implementations based on different types of virtual machines and dynamic accounts. Currently, the implementation type is used to define the isolation model as well as provide a clue to services processing the workspaces to provide implementation-specific processing. Workspace state can be one of running (a workspace deployed on a resource), shutdown (for example, a "cold"

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3451
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

VM image containing no running processes), paused (for example, a "hot" VM image), and corrupted (a workspace that cannot be deployed because of internal inconsistencies). In addition, the generic part of workspace description contains a reference that can be used to check the status of the workspace. The reference embeds information about the workspace owner's distinguished name. Other properties contained in the generic part of the schema include three time-related elements: creation Time, lifecycle, and last Modified. Creation Time records the time when the workspace was first instantiated. Lifecycle indicates how long the workspace is available for use. Last- Modified keeps the information about when the properties of the workspace were modified last. Further definition contains a description of different workspace aspects, such as required hardware, networking configuration, required software installations and workspace capability. Description of the "virtual resource" that represents the hardware requirements of the execution environment contains elements such as the RAM size, disk size, disk type, and accessing mode, as well as devices such as virtual CDROM drives. A network specification contains the description of a network connection and how to establish it (such as the method to obtain an IP address). Software Descriptions contain information about the operating system (e.g., kernel version, distribution type), library signature, and programs installed. Workspace capability describes what can be done with the workspace: for example, a workspace may be configured to run a program on startup or have some programs (in particular, hosting Programs) already running and be able to service requests on specific ports.

Wherever applicable, workspace properties are described as a set of possible values (e.g., RAM size with min and max requirements) rather than one value. The intent is to leave open the largest possible set of mappings of workspaces to real resources. The schemas are extensible to reflect the capabilities of different implementations. For example, we use a VM-based workspace category type, extended from the generic category type, to describe the category information of a workspace implemented with virtual machines. Besides the inherited category name, it contains extra properties such as the type of virtual machine monitor (VMM), which are specific for VM-based workspaces only. Based on the descriptions defined in the schema, workspaces can be selected, cloned, or refined. Cloning a workspace, for example, involves creating a new name (as encoded within the reference element), a new resource and copying the description metadata.

B. Virtual Workspaces as Virtual Machines

We have surveyed candidate technologies for workspace implementations and identified two especially promising ones: configurable dynamic accounts and virtual machines. Although dynamic accounts present an interesting implementation option (especially when used with enforcement tools such as quota and software configuration tools such as Pacman), we pursue this work elsewhere. Our focus in this paper is on a virtual machine implementation of workspaces in view of their outstanding isolation and serialization properties. A virtual machine provides a virtualization of the underlying physical host machine. Software running on the host, called virtual machine monitor (VMM) or hyper visor, is responsible for supporting the perception of multiple isolated machines by intercepting and emulating privileged instructions issued by the guest machines. AVMM typically also provides an interface allowing a client to start, pause or stop multiple guests. A VM representation contains a full image of a VM's RAM, disk,

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3452
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

and other devices, allowing its state to be fully serialized, preserved, and restored at a later date. Recent exploration of Para virtualization techniques has led to substantial performance improvements in virtualization technologies, making virtual machines an attractive option for Grid computing. The serialization properties of VMs create the potential for effortless configuration of execution environments (for example, allowing a user to configure VMs with software required by a given community and clone them). Their isolation from the host machine provides a way of running a different configuration from that of the VM host and allows guest VMs and resource owners to take

advantage of enhanced security properties of VMs. Further, those abilities combined provide potential for migration. For these reasons, VMs provide an excellent implementation option for workspaces: the configuration of a VM image can reflect a workspace's software requirements while the VMM can ensure the enforcement of hardware properties.

3. Integrating Virtual Workspaces into the Grid Architecture

Virtual workspaces refine the execution environment layer in Grid architecture: rather than mapping jobs directly onto hardware resources as in, we map jobs to preconfigured workspaces which can then be mapped to Grid resources. Since a workspace may exist beyond its deployment, and may be in fact deployed many times on different resources during its lifetime, we introduce two new services: VW Repository which provides a management interface to workspaces, and VW Manager which orchestrates their deployment. The figure below illustrates how workspaces work within the Grid infrastructure:

Client
request
inspect and manage
deploy & suspend
Create
VW Manager
Resource
start program
VW
VW ID VW Factory
VW Repository
VW
Figure 1: Grid Interactions with Virtual Workspaces

A. Grid Interactions

In order to create a workspace instance, a Grid client contacts the VW Factory with a workspace description presented in Section 2.A. A negotiation process may take place to ensure that the workspace is created in a policy controlled way. The newly created workspace is registered with a VW Repository, which provides a Grid service interface allowing for inspection and management of workspaces and keeps track of resources implementing workspaces such as virtual machine images. As a result of creation the client is returned a WSRF end-point reference (EPR) to the workspace. We leverage the abstraction of a Grid service resource to enable inspection and management of properties such as termination time. To deploy a VW on a specific resource, a client contacts a VW Manager Grid service on that resource and presents it the workspace's EPR. The VW Manager allows a client to deploy/undeploy, start/stop, and (in our VM-based implementation) also pause /unpause workspaces. Deploying a workspace simply means staging all components of the workspace (such as a VM image) to the resource so that they are available to the VW Manager on that resource. Currently, "checking out" workspace from the repository

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3453
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

puts a lock on them, since their state might change during use. Similarly, "undeploy" releases the hold on workspace resources and the corresponding lock. After the workspaces are staged, they can be started (i.e., become available for computation). Once a VW becomes ready for execution, a program can be started by using Grid infrastructure mechanisms (e.g., Globus Resource Allocation Manager, or GRAM) or by using other methods such as preconfigured program startup or a continuation of a previous execution. The VW Manager can also stop, pause ("freeze" an ongoing computation), or undeploy a stopped or paused workspace (stage it back to the repository for example). In a complete picture of interactions, a community broker would negotiate reservations or agreements for the use of specific resource allocations on a resource. The workspace agreements would be matched against those allocations and create binding agreements allowing the deployment of a specific workspace on a selected resource much as described in SNAP. Such agreements could then be renegotiated and the workspaces migrated, as need arises.

B. Implementation

Our current prototype uses both Xen (version 2.01) and VMware (Workstation, version 4.5) VMs to implement virtual workspaces. The workspaces use IP addresses from a pre-reserved pool for networking. The Grid services and infrastructure described above were implemented by using GT4 (alpha version 3.9.4). In order to be capable of deploying workspaces of a specific type, a host machine has to run a virtual machine monitor of that type as well as the VW Manager Service. At this point we assume that no VM image will leave the trusted environment of a specific site; that is, we do not yet introduce mechanisms to protect integrity and privacy of images themselves, and we assume one repository per site. New workspaces are created by cloning existing images configured with the same requirements and stored in the repository image pool, as in. Each workspace is configured with a certificate and a private key to authenticate itself to clients. At this stage, we do not support negotiation for workspace creation; a workspace request is either accepted or rejected based on existing policies. The VW Manager interfaces with VMM running on a particular resource to implement to stage/unstage, start/stop, and pause/unpause operations: in Xen via an HTTP control interface and on VMware Workstation via GUI scripting. To stage a workspace, the VW Manager transfers the workspace data (including description metadata and the implementation-specific image) from the VM Repository to the host node by using GridFTP. Once the workspace data transfer is complete, the VW Manager waits for the client to start the workspace, which includes creating a workspace resource, loading the VM image into memory, and booting the VM. At boot time the VM may initiate preconfigured operations such as obtaining its network address and starting programs (including the GT hosting environment). Once this step is completed, a VM can advertise the hosted Grid services such as GRAM for clients to invoke. Although in the current prototype we do not address the issue of ensuring privacy and integrity of workspace representations (VM images), we do support standard Grid authentication and authorization mechanisms. Running workspaces and Grid clients mutually authenticate by using the Grid Security Infrastructure GSI. Our infrastructure also accepts VOMS certificates and is capable of extracting VOMS attributes for authorization. Authorization of workspace creation, deployment, and management is configured via access control lists based on the distinguished name and attributes of Grid

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3454
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

entities.

4. Experiences with Virtual Workspaces

The performance impact of virtual machines on applications has been shown to be small (typically under 5% of slowdown) for different application classes. In our preliminary evaluation, we explored the performance impact of different ways of using VMs as part of Grid infrastructure. We also conducted a preliminary evaluation of VM usage with applications. To explore the best ways of using VMs within the Grid infrastructure, we timed the process of starting up a workspace and running a program in it on a remote node in different startup configurations. We repeated those experiments for both of our workspace implementations (Xen and VMware Workstation) and compared them with the time of job startup through a call to GT4 GRAM. In all the experiments we assume that the necessary data had already been staged to the node (i.e., the executable and input data for GRAM and a VM image in the case of workspace deployment). All experiments were run on a dual 2.2 GHz Xeon server configured to run single- CPU guest VMs. The same VM image configuration (Debian Sarge) was used for both Xen and VMware, and the same test application was used for all experiments. Time was measured on the server side only, by using wall clock time.

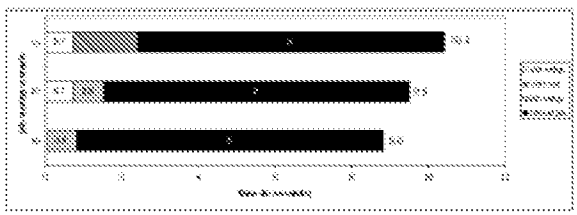


Figure 2: Xen-based VWs versus job invocation using GRAM

Figure 2(a) shows time elapsed from the moment GRAM received a job startup request till the end of the job: as per GRAM default the job is run in a user account. Figures 2 (b-c) show the combined time of deploying a workspace implemented as a Xen VM and starting a job in it under two different scenarios. In 2(b) we take advantage of the serialization property of VMs: we prepare a "hot image" (a paused image) with a hosting environment already started up. A call to GRAM is placed as soon as the environment is available. In 2(c), instead of configuring the workspace to start up a GT4 hosting environment, we start the job directly. Unless the VM is partially configured at boot time, this scenario could also be optimized by pausing a booted VM. For comparison, we also ran a similar test on VMware Workstation, which is hard to time given its non-programmatic, opaque interface (VMware ESX/GSX tools provide more efficient and direct interfaces). The results are summarized in Figure 3; a significant amount of time is spent in a controller adapter to the Workstation version.

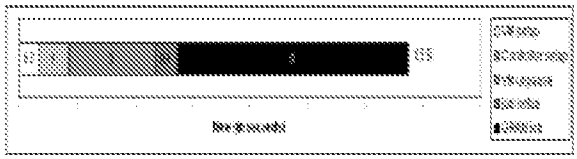


Figure 3: VM deployment time using VMware workstation

Our preliminary experiments show that a Xen VM could be effective for a range of configurations with addition to job deployment time that can easily be absorbed by latency in the Grid environment. Startup costs of deploying a VM image are comparable to starting a job; the trade-off is that while GRAM implements a flexible job deployment strategy the VM is already pre-configured with the job to be started. On the other hand, the advantages gained by using VMs are significant. Deploying hot images ready to process requests offers the fastest solution (note that this method could be used

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3455
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

to eliminate application initialization time as well). However, this may not always be possible; in such cases using a VM preconfigured to start a specific job is also an efficient alternative (currently no credential is delegated to such a job; the startup is based on authorization of the client that submits the VM). To obtain a preliminary assessment of the usefulness of this infrastructure to Grid applications, we experimented with the EMBOSS suite for bioinformatics applications. The most important effect was facilitating deployment: while the EMBOSS installation took roughly 45 minutes, starting a preconfigured VMware Workstation workspace took on average 6 minutes and 23 seconds (including staging), and the process itself eliminated installation errors (as per our results above, Xen could be used for even better results). Another noticeable consequence was a more flexible use of resources, especially in a heterogeneous resource environment: we no longer had to require a homogenous resource base. While the determination is application specific, our results suggest there are situations where using virtual machine workspace implementations would not be useful. Jobs that require 100% resource utilization do not require specific environments, or that each run for less time than the time required to stage, boot or unpause a virtual machine would not be good candidates for the infrastructure.

5. Related Work

Due to its potential, the use of virtual machines in Grid computing is attracting increasing attention. The In-Vigo project made substantial progress in this direction while the Virtuoso and VIOLIN projects explored networking issues arising from use of VMs in this setting. Our approach differs in that it focuses on technology-neutral virtual workspaces, using virtual machines as only one of their potential implementations. Further, recognizing that a workspace may be deployed many times we distinguish between workspace creation and its deployment introducing a different architecture for its support. Attempts to capture requirements of an execution environment to some extent and automate their deployment have also been made before: for example, the virtual appliance project uses virtual machines configured based on descriptions in a configuration language to ease administrative burden, and the Cluster on Demand (COD) project allows a user to choose from a database of configurations to configure a partition of a cluster. We differ from these projects by our focus on Grid computing and technology-independent approach. Finally, the Xenoserver project is building an infrastructure for wide-area distributed computing based on virtual machines similar to our long-term goals. Here, we differ by building within the established framework of Grid computing rather than providing new infrastructure.

6. Conclusions and Future Work

We have described the abstraction of a virtual workspace, a customizable execution environment capable of being deployed on a variety of platforms in the Grid. Workspaces are defined in terms of client requirements, such as software and hardware requirements, and are implemented in terms of technologies providing an isolated execution environment, quality of service at the granularity of a workspace (as opposed to a single process), customized software installation, and, in the case of VMs, execution serialization and migration. We showed how workspaces can be integrated into the existing Grid infrastructure. The integration entails relatively small changes, but introduces substantial flexibility of use. The use of Grid protocols allows us to fully leverage this

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3456
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

flexibility and create, deploy, and shut down workspaces dynamically based on policy-driven provisioning decisions. By virtue of their properties, workspaces are a promising vehicle for implementing policy-driven Grid usage. To evaluate the feasibility of our workspace implementations, we compared the performance of GT4 GRAM, a widely used job startup service for Grid applications, to the process of starting up workspaces implemented as virtual machines in a variety of scenarios. In conjunction with the small performance impact on applications, our results show workspaces to be a promising abstraction for Grid computing. In addition, our experiments identified a number of job startup scenarios relevant in the workspace context, showing how they may be used in practice. Preliminary application evaluation of workspaces also proved satisfactory and fully realized our expectations for more flexible resource usage. More work is needed in order to fully assess the usefulness of these ideas. In the short term, we will focus on the privacy and integrity of migrating workspaces. Workspace distribution (delivery to host) can require transferring images of a few gigabytes; this task can be handled by using the presence of an image on a node as a matching criterion as proposed in or by transferring only partial images as in. In addition to performance impact on individual applications, we are considering scalability issues that can be addressed by either using a lighter-weight workspace implementation such as or mapping groups of jobs to one workspace.

References

1. Czajkowski, K., D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, The WS-Resource Framework. 2004: www.globus.org/wsrf.

2. Andrieux, A., K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, Web Services Agreement Specification (WSAgreement) Draft 20. 2004: https://forge.gridforum.org/projects/graap-wg/.

3. Figueiredo, R., P. Dinda, and J. Fortes. A Case for Grid Computing on Virtual Machines. in 23rd International Conference on Distributed Computing Systems. 2003.

4. Keahey, K., K. Doering, and I. Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. in 5th International Workshop in Grid Computing. 2004.

5. Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. in ACM Symposium on Operating Systems Principles (SOSP).

6. VMware: http://www.vmware.com.

7. Czajkowski, K., I. Foster, V. Sander, C. Kesselman, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. in 8th Workshop on Job Scheduling Strategies for Parallel Processing. 2002. Edinburgh, Scotland.

8. Raman, R., M. Livny, and M. Solomon, Matchmaking: An Extensible Framework for Distributed Resource Management. Cluster Computing: The Journal of Networks, Software Tools and Applications, 1999. 2: p. 129-138.

9. Workspace Management Service: http://www.mcs.anl.gov/workspace/. 10. Keahey, K., M. Ripeanu, and K. Doering. Dynamic Creation and Management of Runtime Environments in the Grid. in Workshop on Designing and Building Web

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3457
IPR2025-00379 (IBM v. Croga)

ISSN (online): 2250-141X
Vol. 2 Issue 2, April 2012

Services. 2003. Chicago, IL.
11. McNab, A., Grid-Based Access Control for Unix Environments, Filesystems and Web Sites. Proceeings of the CHEP 2003 conference, 2003.
12. Youssef, S., Pacman: A Package Manager. 2004: http://physics.bu.edu/~youssef/pacman/.
13. Goldberg, R., Survey of Virtual Machine Research. IEEE Computer, 1974. 7(6): p. 34-45.
14. Czajkowski, K., I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, A Resource Management Architecture for Metacomputing Systems, in 4th Workshop on Job Scheduling Strategies for Parallel Processing. 1998, Springer-Verlag. p. 62-82.
15. Krsul, I., A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. in SC04. 2004. Pittsburgh, PA.
16. Allcock, W., GridFTP: Protocol Extensions to FTP for the Grid. 2003, Global Grid Forum.
17. Butler, R., D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, Design and Deployment of a National-Scale Authentication Infrastructure. IEEE Computer, 2000. 33(12)

© Copyright – IJST 2011

IBM - Ex. 1002-Part 10 of 10, Page 3458
IPR2025-00379 (IBM v. Croga)

2009 30th IEEE Symposium on Security and Privacy

Native Client: A Sandbox for Portable, Untrusted x86 Native Code

Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth,
Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar
Google Inc.

Abstract

This paper describes the design, implementation and evaluation of Native Client, a sandbox for untrusted x86 native code. Native Client aims to give browser-based applications the computational performance of native applications without compromising safety. Native Client uses software fault isolation and a secure runtime to direct system interaction and side effects through interfaces managed by Native Client. Native Client provides operating system portability for binary code while supporting performance-oriented features generally absent from web application programming environments, such as thread support, instruction set extensions such as SSE, and use of compiler intrinsics and hand-coded assembler. We combine these properties in an open architecture that encourages community review and 3rd-party tools.

1. Introduction

As an application platform, the modern web browser brings together a remarkable combination of resources, including seamless access to Internet resources, high-productivity programming languages such as JavaScript, and the richness of the Document Object Model (DOM) [64] for graphics presentation and user interaction. While these strengths put the browser in the forefront as a target for new application development, it remains handicapped in a critical dimension: computational performance. Thanks to Moore's Law and the zeal with which it is observed by the hardware community, many interesting applications get adequate performance in a browser despite this handicap. But there remains a set of computations that are generally infeasible for browser-based applications due to performance constraints, for example: simulation of Newtonian physics, computational fluid-dynamics, and high-resolution scene rendering. The current environment also tends to preclude use of the large bodies of high-quality code developed in languages other than JavaScript.

Modern web browsers provide extension mechanisms such as ActiveX [15] and NPAPI [48] to allow native code to be loaded and run as part of a web application. Such architectures allow plugins to circumvent the security mechanisms otherwise applied to web content, while giving them access to full native performance, perhaps as a secondary consideration. Given this organization, and the absence of effective technical measures to constrain these plugins, browser applications that wish to use native-code must rely on non-technical measures for security; for example, manual establishment of trust relationships through pop-up dialog boxes, or manual installation of a console application. Historically, these non-technical measures have been inadequate to prevent execution of malicious native code, leading to inconvenience and economic harm [10], [54]. As a consequence we believe there is a prejudice against native code extensions for browser-based applications among experts and distrust among the larger population of computer users.

While acknowledging the insecurity of the current systems for incorporating native-code into web applications, we also observe that there is no fundamental reason why native code should be unsafe. In Native Client, we separate the problem of safe native execution from that of extending trust, allowing each to be managed independently. Conceptually, Native Client is organized in two parts: a constrained execution environment for native code to prevent unintended side effects, and a runtime for hosting these native code extensions through which allowable side effects may occur safely.

The main contributions of this work are:

- an infrastructure for OS and browser-portable sandboxed x86 binary modules,
- support for advanced performance capabilities such as threads, SSE instructions [32], compiler intrinsics and hand-coded assembler,
- an open system designed for easy retargeting of new compilers and languages, and
- refinements to CISC software fault isolation, using x86 segments for improved simplicity and reduced overhead.

We combine these features in an infrastructure that supports safe side effects and local communication. Overall, Native Client provides sandboxed execution of native code and portability across operating systems, delivering native code performance for the browser.

The remainder of the paper is organized as follows. Section 1.1 describes our threat model. Section 2 develops some essential concepts for the NaCl[1] system architecture and

1. We use "NaCl" as an adjective reference to the Native Client system.

1081-6011/09 $25.00 © 2009 IEEE
DOI 10.1109/SP.2009.25

IEEE computer society

IBM - Ex. 1002-Part 10 of 10, Page 3459
IPR2025-00379 (IBM v. Croga)

Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

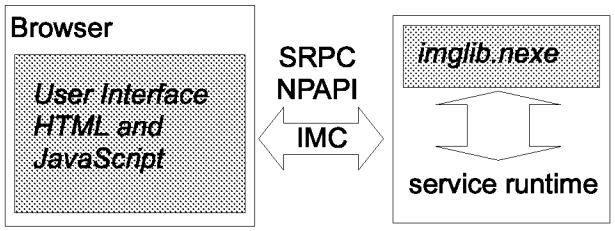


Figure 1: Hypothetical NaCl-based application for editing and sharing photos. Untrusted modules have a grey background.

programming model. Section 3 gives additional implementation details, organized around major system components. Section 4 provides a quantitative evaluation of the system using more realistic applications and application components. In Section 5 we discuss some implications of this work. Section 6 discusses relevant prior and contemporary systems. Section 7 concludes.

1.1. Threat Model

Native Client should be able to handle untrusted modules from any web site with comparable safety to accepted systems such as JavaScript. When presented to the system, an untrusted module may contain *arbitrary* code and data. A consequence of this is that the NaCl runtime must be able to confirm that the module conforms to our validity rules (detailed below). Modules that don't conform to these rules are rejected by the system.

Once a conforming NaCl module is accepted for execution, the NaCl runtime must constrain its activity to prevent unintended side effects, such as might be achieved via unmoderated access to the native operating system's system call interface. The NaCl module may arbitrarily combine the entire variety of behaviors permitted by the NaCl execution environment in attempting to compromise the system. It may execute any reachable instruction block in the validated text segment. It may exercise the NaCl application binary interface to access runtime services in any way: passing invalid arguments, etc. It may also send arbitrary data via our intermodule communication interface, with the communicating peer responsible for validating input. The NaCl module may allocate memory and spawn threads up to resource limits. It may attempt to exploit race conditions in subverting the system.

We argue below that our architecture and code validity rules constrain NaCl modules within our sandbox.

2. System Architecture

A NaCl application is composed of a collection of trusted and untrusted components. Figure 1 shows the structure of a

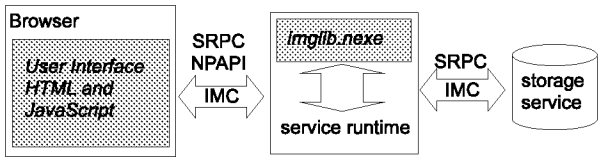


Figure 2: The hypothetical photo application of Figure 1 with a trusted storage service.

hypothetical NaCl-based application for managing and sharing photos. It consists of two components: A user interface, implemented in JavaScript and executing in the web browser, and an image processing library (imglib.nexe), implemented as a NaCl module. In this hypothetical scenario, the user interface and image processing library are part of the application and therefore untrusted. The browser component is constrained by the browser execution environment and the image library is constrained by the NaCl container. Both components are portable across operating systems and browsers, with native code portability enabled by Native Client. Prior to running the photo application, the user has installed Native Client as a browser plugin. Note that the NaCl browser plugin itself is OS and browser specific. Also note it is trusted, that is, it has full access to the OS system call interface and the user trusts it to not be abusive.

When the user navigates to the web site that hosts the photo application, the browser loads and executes the application JavaScript components. The JavaScript in turn invokes the NaCl browser plugin to load the image processing library into a NaCl container. Observe that the native code module is loaded silently—no pop-up window asks for permission. Native Client is responsible for constraining the behavior of the untrusted module.

Each component runs in its own private address space. Inter-component communication is based on Native Client's reliable datagram service, the IMC (Inter-Module Communications). For communications between the browser and a NaCl module, Native Client provides two options: a Simple RPC facility (SRPC), and the Netscape Plugin Application Programming Interface (NPAPI), both implemented on top of the IMC. The IMC also provides shared memory segments and shared synchronization objects, intended to avoid messaging overhead for high-volume or high-frequency communications.

The NaCl module also has access to a "service runtime" interface, providing for memory management operations, thread creation, and other system services. This interface is analogous to the system call interface of a conventional operating system.

In this paper we use "NaCl module" to refer to untrusted native code. Note however that applications can use multiple NaCl modules, and that both trusted and untrusted components may use the IMC. For example, the user of the photo

IBM - Ex. 1002-Part 10 of 10, Page 3460
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

application might optionally be able to use a (hypothetical) trusted NaCl service for local storage of images, illustrated in Figure 2. Because it has access to local disk, the storage service must be installed as a native browser plugin; it can't be implemented as a NaCl module. Suppose the photo application has been designed to optionally use the stable storage service; the user interface would check for the stable storage plugin during initialization. If it detected the storage service plugin, the user interface would establish an IMC communications channel to it, and pass a descriptor for the channel to the image library, enabling the image library and the storage service to communicate directly via IMC-based services (SRPC, shared memory, etc.). In this case the NaCl module will typically be statically linked against a library that provides a procedural interface for accessing the storage service, hiding details of the IMC-level communications such as whether it uses SRPC or whether it uses shared memory. Note that the storage service must assume that the image library is untrusted. The service is responsible for insuring that it only services requests consistent with the implied contract with the user. For example, it might enforce a limit on total disk used by the photo application and might further restrict operations to only reference a particular directory.

Native Client is ideal for application components requiring pure computation. It is not appropriate for modules requiring process creation, direct file system access, or unrestricted access to the network. Trusted facilities such as storage should generally be implemented outside of Native Client, encouraging simplicity and robustness of the individual components and enforcing stricter isolation and scrutiny of all components. This design choice echoes microkernel operating system design [2], [12], [25].

With this example in mind we will now describe the design of key NaCl system components in more detail.

2.1. The Inner Sandbox

Native Client is built around an x86-specific intra-process "inner sandbox." We believe that the inner sandbox is robust; regardless, to provide defense in depth [13], [16] we have also developed a second "outer sandbox" that mediates system calls at the process boundary. The outer sandbox is substantially similar to prior structures (systrace [50] and Janus [24]) and we will not discuss it in detail in this paper.

The inner sandbox uses static analysis to detect security defects in untrusted x86 code. Previously, such analysis has been challenging for arbitrary x86 code due to such practices as self-modifying code and overlapping instructions. In Native Client we disallow such practices through a set of alignment and structural rules that, when observed, insure that the native code module can be disassembled reliably, such that all reachable instructions are identified during disassembly. With reliable disassembly as a tool, our validator can then insure that the executable includes only the subset of legal instructions, disallowing unsafe machine instructions.

The inner sandbox further uses x86 segmented memory to constrain both data and instruction memory references. Leveraging existing hardware to implement these range checks greatly simplifies the runtime checks required to constrain memory references, in turn reducing the performance impact of safety mechanisms.

This inner sandbox is used to create a security subdomain within a native operating system process. With this organization we can place a trusted service runtime subsystem within the same process as the untrusted application module, with a secure trampoline/springboard mechanism to allow safe transfer of control from trusted to untrusted code and vice-versa. Although in some cases a process boundary could effectively contain memory and system-call side effects, we believe the inner sandbox can provide better security. We generally assume that the operating system is not defect free, such that the process barrier might have defects, and further that the operating system might deliberately map resources such as shared libraries into the address space of all processes, as occurs in Microsoft Windows. In effect our inner sandbox not only isolates the system from the native module, but also helps to isolate the native module from the operating system.

2.2. Runtime Facilities

The sandboxes prevent unwanted side effects, but some side effects are often necessary to make a native module useful. For interprocess communications, Native Client provides a reliable datagram abstraction, the "Inter-Module Communications" service or IMC. The IMC allows trusted and untrusted modules to send/receive datagrams consisting of untyped byte arrays along with optional "NaCl Resource Descriptors" to facilitate sharing of files, shared memory objects, communication channels, etc., across process boundaries. The IMC can be used by trusted or untrusted modules, and is the basis for two higher-level abstractions. The first of these, the Simple Remote Procedure Call (SRPC) facility, provides convenient syntax for defining and using subroutines across NaCl module boundaries, including calls to NaCl code from JavaScript in the browser. The second, NPAPI, provides a familiar interface to interact with browser state, including opening URLs and accessing the DOM, that conforms to existing constraints for content safety. Either of these mechanisms can be used for general interaction with conventional browser content, including content modifications, handling mouse and keyboard activity, and fetching additional site content; substantially all the resources commonly available to JavaScript.

As indicated above, the service runtime is responsible for providing the container through which NaCl modules

IBM - Ex. 1002-Part 10 of 10, Page 3461
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

interact with each other and the browser. The service runtime provides a set of system services commonly associated with an application programming environment. It provides sysbrk() and mmap() system calls, primitives to support malloc()/free() interface or other memory allocation abstractions. It provides a subset of the POSIX threads interface, with some NaCl extensions, for thread creation and destruction, condition variables, mutexes, semaphores, and thread-local storage. Our thread support is sufficiently complete to allow a port of Intel's Thread Building Blocks [51] to Native Client. The service runtime also provides the common POSIX file I/O interface, used for operations on communications channels as well as web-based read-only content. As the name space of the local file system is not accessible to these interfaces, local side effects are not possible.

To prevent unintended network access, network system calls such as connect() and accept() are simply omitted. NaCl modules can access the network via JavaScript in the browser. This access is subject to the same constraints that apply to other JavaScript access, with no net effect on network security.

The NaCl development environment is largely based on Linux open source systems and will be familiar to most Linux and Unix developers. We have found that porting existing Linux libraries is generally straightforward, with large libraries often requiring no source changes.

2.3. Attack Surface

Overall, we recognize the following as the system components that a would-be attacker might attempt to exploit:

- inner sandbox: binary validation
- outer sandbox: OS system-call interception
- service runtime binary module loader
- service runtime trampoline interfaces
- IMC communications interface
- NPAPI interface

In addition to the inner and outer sandbox, the system design also incorporates CPU and NaCl module black-lists. These mechanisms will allow us to incorporate layers of protection based on our confidence in the robustness of the various components and our understanding of how to achieve the best balance between performance, flexibility and security.

In the next section we hope to demonstrate that secure implementations of these facilities are possible and that the specific choices made in our own implementation work are sound.

3. Native Client Implementation

3.1. Inner Sandbox

In this section we explain how NaCl implements software fault isolation. The design is limited to explicit control flow, expressed with calls and jumps in machine code. Other types of control flow (e.g. exceptions) are managed in the NaCl service runtime, external to the untrusted code, as described with the NaCl runtime implementation below.

C1	Once loaded into the memory, the binary is not writable, enforced by OS-level protection mechanisms during execution.
C2	The binary is statically linked at a start address of zero, with the first byte of text at 64K.
C3	All indirect control transfers use a `nacljmp` pseudo-instruction (defined below).
C4	The binary is padded up to the nearest page with at least one `hlt` instruction (0xf4).
C5	The binary contains no instructions or pseudo-instructions overlapping a 32-byte boundary.
C6	All *valid* instruction addresses are reachable by a fall-through disassembly that starts at the load (base) address.
C7	All direct control transfers target valid instructions.

Table 1: Constraints for NaCl binaries.

Our inner sandbox uses a set of rules for reliable disassembly, a modified compilation tool chain that observes these rules, and a static analyzer that confirms that the rules have been followed. This design allows for a small trusted code base (TCB) [61], with the compilation tools outside the TCB, and a validator that is small enough to permit thorough review and testing. Our validator implementation requires less than 600 C statements (semicolons), including an x86 decoder and `cpuid` decoding. This compiles into about 6000 bytes of executable code (Linux optimized build) of which about 900 bytes are the `cpuid` implementation, 1700 bytes the decoder, and 3400 bytes the validator logic.

To eliminate side effects the validator must address four sub-problems:

- Data integrity: no loads or stores outside of data sandbox
- Reliable disassembly
- No unsafe instructions
- Control flow integrity

To solve these problems, NaCl builds on previous work on CISC fault isolation. Our system combines 80386 segmented memory [14] with previous techniques for CISC software fault isolation [40]. We use 80386 segments to constrain data references to a contiguous subrange of the virtual 32-bit address space. This allows us to effectively implement a data sandbox without requiring sandboxing of load and store instructions. VX32 [20], [21] implements its data sandbox in a similar fashion. Note that NaCl modules are 32-bit x86 executables. The more recent 64-bit executable model is not supported.

Table 1 lists the constraints Native Client requires of untrusted binaries. Together, constraints C1 and C6 make disassembly reliable. With reliable disassembly as a tool, detection of unsafe instructions is straightforward. A partial list of opcodes disallowed by Native Client includes:

- `syscall` and `int`. Untrusted code cannot invoke the

IBM - Ex. 1002-Part 10 of 10, Page 3462
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

operating system directly.

- all instructions that modify x86 segment state, including `lds`, far calls, etc.
- `ret`. Returns are implemented with a sandboxing sequence that ends with an indirect jump.

Apart from facilitating control sandboxing, excluding `ret` also prevents a vulnerability due to a race condition if the return address were checked on the stack. A similar argument requires that we disallow memory addressing modes on indirect `jmp` and `call` instructions. Native Client does allow the `hlt` instruction. It should never be executed by a correct instruction stream and will cause the module to be terminated immediately. As a matter of hygiene, we disallow all other privileged/ring-0 instructions, as they are never required in a correct user-mode instruction stream. We also constrain x86 prefix usage to only allow known useful instructions. Empirically we have found that this eliminates certain denial-of-service vulnerabilities related to CPU errata.

The fourth problem is control flow integrity, insuring that all control transfers in the program text target an instruction identified during disassembly. For each direct branch, we statically compute the target and confirm it is a valid instruction as per constraint C6. Our technique for indirect branches combines 80386 segmented memory with a simplified sandboxing sequence. As per constraint C2 and C4, we use the `CS` segment to constrain executable text to a zero-based address range, sized to a multiple of 4K bytes. With the text range constrained by segmented memory, a simple constant mask is adequate to ensure that the target of an indirect branch is aligned mod 32, as per constraints C3 and C5:

```
and         %eax, 0xffffffe0
jmp         *%eax
```

We will refer to this special two instruction sequence as a `nacljmp`. Encoded as a three-byte `and` and a two-byte `jmp` it compares favorably to previous implementations of CISC sandboxing [40], [41], [56]. Without segmented memory or zero-based text, sandboxed control flow typically requires two six-byte instructions (an `and` and an `or`) for a total of fourteen bytes.

Considering the pseudo-code in Figure 3, we next assert and then prove the correctness of our design for control-flow integrity. Assuming the text in question was validated without errors, let S be the set of instructions addresses from the list StartAddr.

Theorem: S contains all addresses that can be reached from an instruction with address in S.

Proof: By contradiction. Suppose an address IP not in S is reached during execution from a predecessor instruction A with address in S. Because execution is constrained by x86 segmentation, IP must trivially be in [0:TextLimit). So IP can only be reached in one of three ways.

```
// TextLimit = the upper text address limit
// Block(IP) = 32-byte block containing IP
// StartAddr = list of inst start addresses
// JumpTargets = set of valid jump targets

// Part 1: Build StartAddr and JumpTargets
IP = 0; icount = 0; JumpTargets = { }
while IP <= TextLimit:
  if inst_is_disallowed(IP):
    error "Disallowed instruction seen"
  StartAddr[icount++] = IP
  if inst_overlaps_block_size(IP):
    error "Block alignment failure"
  if inst_is_indirect_jump_or_call(IP):
    if !is_2_inst_nacl_jmp_idiom(IP) or
      icount < 2 or
      Block(StartAddr[icount-2]) != Block(IP):
      error "Bad indirect control transfer"
  else
    // Note that indirect jmps are inside
    // a pseudo-inst and bad jump targets
    JumpTargets = JumpTargets + { IP }
  // Proceed to the fall-through address
  IP += InstLength(IP)

// Part 2: Detect invalid direct transfers
for I = 0 to length(StartAddr)-1:
  IP = StartAddr[I]
  if inst_is_direct_jump_or_call(IP):
    T = direct_jump_target(IP)
    if not(T in [0:TextLimit))
       or not(T in JumpTargets):
      error "call/jmp to invalid address"
```

Figure 3: Pseudo-code for the NaCl validator.

case 1: IP is reached by falling through from A. This implies that IP is InstAddr(A) + InstLength(A). But this address would have been in S from part 1 of the construction. Contradiction.

case 2: IP is reached by a direct jump or call from an instruction A in S. Then IP must be in JumpTargets, a condition checked by part 2 of the construction. Observe that JumpTargets is a subset of S, from part 1 of the construction. Therefore IP must be in S. Contradiction.

case 3: IP is reached by an indirect transfer from an instruction at A in S. Since the instruction at A is an indirect call or jump, any execution of A always immediately follows the execution of an `and`. After the `and` the computed address is aligned 0 mod 32. Since no instruction can straddle a 0 mod 32 boundary, every 0 mod 32 address in [0, TextLimit) must be in S. Hence IP is in S. Contradiction.

Hence any instruction reached from an instruction in S is also in S. ■

Note that this analysis covers explicit, synchronous control flow only. Exceptions are discussed in Section 3.2.

If the validator were excessively slow it might discourage people from using the system. We find our validator can check code at approximately 30MB/second (35.7 MB in 1.2

IBM - Ex. 1002-Part 10 of 10, Page 3463
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

seconds, measured on a MacBook Pro with MacOS 10.5, 2.4GHz Core 2 Duo CPU, warm file-system cache). At this speed, the compute time for validation will typically be very small compared to download time, and so is not a performance issue.

We believe this inner sandbox needs to be extremely robust. We have tested it for decoding defects using random instruction generation as well as exhaustive enumeration of valid x86 instructions. We also have used "fuzzing" tests to randomly modify test executables. Initially these tests exposed critical implementation defects, although as testing continues no defects have been found in the recent past. We have also tested on various x86 microprocessor implementations, concerned that processor errata might lead to exploitable defects [31], [38]. We did find evidence of CPU defects that lead to a system "hang" requiring a power-cycle to revive the machine. This occurred with an earlier version of the validator that allowed relatively unconstrained use of x86 prefix bytes, and since constraining it to only allow known useful prefixes, we have not been able to reproduce such problems.

3.2. Exceptions

Hardware exceptions (segmentation faults, floating point exceptions) and external interrupts are not allowed, due in part to distinct and incompatible exception models in Linux, MacOS and Windows. Both Linux and Windows rely on the x86 stack via `%esp` for delivery of these events. Regrettably, since NaCl modifies the `%ss` segment register, the stack appears to be invalid to the operating system, such that it cannot deliver the event and the corresponding process is immediately terminated. The use of x86 segmentation for data sandboxing effectively precludes recovery from these types of exceptions. As a consequence, NaCl untrusted modules apply a failsafe policy to exceptions. Each NaCl module runs in its own OS process, for the purpose of exception isolation. NaCl modules cannot use exception handling to recover from hardware exceptions and must be correct with respect to such error conditions or risk abrupt termination. In a way this is convenient, as there are very challenging security issues in delivering these events safely to untrusted code.

Although we cannot currently support hardware exceptions, Native Client does support C++ exceptions [57]. As these are synchronous and can be implemented entirely at user level there are no implementation issues. Windows Structured Exception Handling [44] requires non-portable operating support and is therefore not supported.

3.3. Service Runtime

The service runtime is a native executable invoked by an NPAPI plugin that also supports interaction between the service runtime and the browser. It supports a variety of web browsers on Windows, MacOS and Linux. It implements the dynamic enforcement that maintains the integrity of the inner sandbox and provides resource abstractions to isolate the NaCl application from host resources and operating system interface. It contains trusted code and data that, while sharing a process with the contained NaCl module, are accessible only through a controlled interface. The service runtime prevents untrusted code from inappropriate memory accesses through a combination of x86 memory segment and page protection.

Platform	"null" Service Runtime call time
Linux, Ubuntu 6.06 Intel™ Core™ 2 6600 2.4 GHz	156
Mac OSX 10.5 Intel™ Xeon™ E5462 2.8 GHz	148
Windows XP Intel™ Core™ 2 Q6600 2.4 GHz	123

Table 2: Service runtime context switch overhead. The runtimes are measured in nanoseconds. They are obtained by averaging the measurements of 10 runs of a NaCl module which measured the time required to perform 10,000,000 "null" service runtime calls.

When a NaCl module is loaded, it is placed in a segment-isolated 256MB region within the service runtime's address space. The first 64 KB of the NaCl module's address space (NaCl "user" address space) is reserved for initialization by the service runtime. The first 4 KB is read and write protected to detect NULL pointers. The remaining 60 KB contains trusted code that implements our "trampoline" call gate and "springboard" return gate. Untrusted NaCl module text is loaded immediately after this 64 KB region. The `%cs` segment is set to constrain control transfers from the zero base to the end of the NaCl module text. The other segment registers are set to constrain data accesses to the 256 MB NaCl module address space.

Because it originates from and is installed by the trusted service runtime, trampoline and springboard code is allowed to contain instructions that are forbidden elsewhere in untrusted executable text. This code, patched at runtime as part of the NaCl module loading process, uses segment register manipulation instructions and the `far call` instruction to enable control transfers between the untrusted user code and the trusted service runtime code. Since every *0 mod 32* address in the first 64 KB of the NaCl user space is a potential computed control flow target, these are our entry points to a table of system-call trampolines. One of these entry points is blocked with a `hlt` instruction, so that the remaining space may be used for code that can only be invoked from the service runtime. This provides space for the springboard return gate.

IBM - Ex. 1002-Part 10 of 10, Page 3464
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

Invocation of a trampoline transfers control from untrusted code to trusted code. The trampoline sequence resets `%ds` and then uses a `far call` to reset the `%cs` segment register and transfer control to trusted service handlers, reestablishing the conventional flat addressing model expected by the code in the service runtime. Once outside the NaCl user address space, it resets other segment registers such as `%fs`, `%gs`, and `%ss` to re-establish the native-code threading environment, fully disabling the inner sandbox for this thread, and loads the stack register `%esp` with the location of a trusted stack for use by the service runtime. Note that the per-thread trusted stack resides outside the untrusted address space, to protect it from attack by other threads in the untrusted NaCl module.

Just as trampolines permit crossing from untrusted to trusted code, the springboard enables crossing in the other direction. The springboard is used by the trusted runtime

- to transfer control to an arbitrary untrusted address,
- to start a new POSIX-style thread, and
- to start the main thread.

Alignment ensures that the springboard cannot be invoked directly by untrusted code. The ability to jump to an arbitrary untrusted address is used in returning from a service call. The return from a trampoline call requires popping an unused trampoline return addresses from the top of the stack, restoring the segment registers, and finally aligning and jumping to the return address in the NaCl module.

Table 2 shows the overhead of a "null" system call. The Linux overhead of 156 ns is slightly higher than that of the Linux 2.6 getpid syscall time, on the same hardware, of 138 ns (implemented via the vsyscall table and using the `sysenter` instruction). We note that the user/kernel transfer has evolved continuously over the life of the x86 architecture. By comparison, the segment register operations and far calls used by the NaCl trampoline are somewhat less common, and may have received less consideration over the history of the x86 architecture.

3.4. Communications

The IMC is the basis of communications into and out of NaCl modules. The implementation is built around a *NaCl socket*, providing a bi-directional, reliable, in-order datagram service similar to Unix domain sockets [37]. An untrusted NaCl module receives its first NaCl socket when it is created, accessible from JavaScript via the Document-Object Model object used to create it. The JavaScript uses the socket to send messages to the NaCl module, and can also share it with other NaCl modules. The JavaScript can also choose to connect the module to other services available to it by opening and sharing NaCl sockets as NaCl descriptors. NaCl descriptors can also be used to create shared memory segments.

Number of Descriptor	Linux	OSX	Windows
1	3.3	31.5	38
2	5.3	38.6	51
3	6.6	47.9	64
4	8.2	50.9	77
5	9.7	54.1	90
6	11.1	60.0	104
7	12.6	63.7	117
8	14.2	66.2	130

Table 3: NaCl resource descriptor transfer cost. The times are in microseconds. In this test, messages carrying zero data bytes and a varying number of I/O descriptors are transferred from a client NaCl module to a server NaCl module. On OSX, a request/ack mechanism is needed as a bug workaround in the OSX implementation of sendmsg. On Windows, a DuplicateHandle() system call is required per I/O object transferred.

Using NaCl messages, Native Client's SRPC abstraction is implemented entirely in untrusted code. SRPC provides a convenient syntax for declaring procedural interfaces between JavaScript and NaCl modules, or between two NaCl modules, supporting a few basic types (int, float, char) as well as arrays in addition to NaCl descriptors. More complex types and pointers are not supported. External data representation strategies such as XDR [18] or Protocol Buffers [26] can easily be layered on top of NaCl messages or SRPC.

Our NPAPI implementation is also layered on top of the IMC and supports a subset of the common NPAPI interface. Specific requirements that shaped the current implementation are the ability read, modify and invoke properties and methods on the script objects in the browser, support for simple raster graphics, provide the createArray() method and the ability to open and use a URL like a file descriptor. The currently implemented NPAPI subset was chosen primarily for expedience, although we will likely constrain and extend it further as we improve our understanding of related security considerations and application requirements.

3.5. Developer Tools

3.5.1. Building NaCl Modules. We have modified the standard GNU tool chain, using version 4.2.2 of the gcc collection of compilers [22], [29] and version 2.18 of binutils [23] to generate NaCl-compliant binaries. We have built a reference binary from newlib[2] using the resulting tool chain, rehosted to use the NaCl trampolines to implement system services (e.g., read(), brk(), gettimeofday(), imc_sendmsg()). Native Client supports an insecure "debug" mode that allows additional file-system interaction not otherwise allowed for secure code.

We modified gcc for Native Client by changing the alignment of function entries (`-falign-functions`) to

2. See http://sourceware.org/newlib/

IBM - Ex. 1002-Part 10 of 10, Page 3465
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

32 bytes and by changing the alignment of the targets branches (`-falign-jumps`) to 32 bytes. We also changed gcc to use `nacljmp` for indirect control transfers, including indirect calls and all returns. We made more significant changes to the assembler, to implement Native Client's block alignment requirements. To implement returns, the assembler ensures that call instructions always appear in the final bytes of a 32 byte block. We also modified the assembler to implement indirect control transfer sequences by expanding the `nacljmp` pseudo-instruction as a properly aligned consecutive block of bytes. To facilitate testing we added support to use a longer `nacljmp` sequence, align the text base, and use an `and` and `or` that uses relocations as masks. This permits testing applications by running them on the command line, and has been used to run the entire gcc C/C++ test suite. We also changed the linker to set the base address of the image as required by the NaCl loader (64K today).

Apart from their direct use the tool chain also serves to document by example how to modify an existing tools chain to generate NaCl modules. These changes were achieved with less than 1000 lines total to be patched in gcc and binutils, demonstrating the simplicity of porting a compiler to Native Client.

3.5.2. Profiling and Debugging. Native Client's open source release includes a simple profiling framework to capture a complete call trace with minimal performance overhead. This support is based on gcc's `-finstrument-functions` code generation option combined with the `rdtsc` timing instruction. This profiler is portable, implemented entirely as untrusted code. In our experience, optimized builds profiled in this framework have performance somewhere between `-O0` and `-O2` builds. Optionally, the application programmer can annotate the profiler output with methods similar to printf, with output appearing in the trace rather than stdout.

Native Client does not currently support interactive debugging of NaCl binary modules. Commonly we debug NaCl module source code by building with standard tools and a library that exports all the interfaces to the NaCl service runtime, allowing us to build debug and NaCl modules from identical source. Over time we hope to improve our support for interactive debugging of release NaCl binaries.

4. Experience

Unless otherwise noted, performance measurements in this section are made without the NaCl outer sandbox. Sandbox overhead depends on how much message-passing and service runtime activity the application requires. At this time we do not have realistic applications of Native Client to stress this aspect of the system.

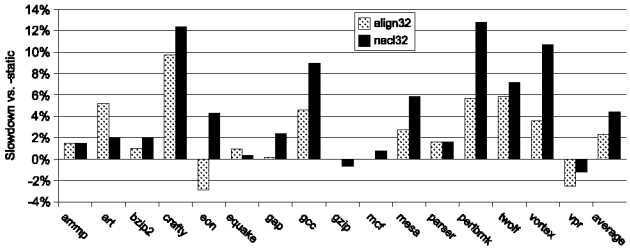


Figure 4: SPEC2000 performance. "Static" results are for statically linked binaries; "align32" results are for binaries aligned in 32-byte blocks, and "nacl32" results are for NaCl binaries.

	static	aligned	NaCl	increase
ammp	200	203	203	1.5%
art	46.3	48.7	47.2	1.9%
bzip2	103	104	104	1.9%
crafty	113	124	127	12%
eon	79.2	76.9	82.6	4.3%
equake	62.3	62.9	62.5	0.3%
gap	63.9	64.0	65.4	2.4%
gcc	52.3	54.7	57.0	9.0%
gzip	149	149	148	-0.7%
mcf	65.7	65.7	66.2	0.8%
mesa	87.4	89.8	92.5	5.8%
parser	126	128	128	1.6%
perlbmk	94.0	99.3	106	13%
twolf	154	163	165	7.1%
vortex	112	116	124	11%
vpr	90.7	88.4	89.6	-1.2%

Table 4: SPEC2000 performance. Execution time is in seconds. All binaries are statically linked.

4.1. SPEC2000

A primary goal of Native Client is to deliver substantially all of the performance of native code execution. NaCl module performance is impacted by alignment constraints, extra instructions for indirect control flow transfers, and the incremental cost of NaCl communication abstractions.

We first consider the overhead of making native code side effect free. To isolate the impact of the NaCl binary constraints (Table 1), we built the SPEC2000 CPU benchmarks using the NaCl compiler, and linked to run as a standard Linux binary. The worst case for NaCl overhead is CPU bound applications, as they have the highest density of alignment and sandboxing overhead. Figure 4 and Table 4 show the overhead of NaCl compilation for a set of benchmarks from SPEC2000. The worst case performance overhead is crafty at about 12%, with other benchmarks averaging about 5% overall. Hardware performance counter measurements indicate that the largest slowdowns are due to instruction cache misses. For crafty, the instruction fetch unit is stalled during 83% of cycles for the NaCl build, compared to 49% for the default build. Gcc and vortex are also significantly impacted by instruction cache misses.

As our current alignment implementation is conservative,

IBM - Ex. 1002-Part 10 of 10, Page 3466
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

	static	aligned	NaCl	increase
ammp	657	759	766	16.7%
art	469	485	485	3.3%
bzip2	492	525	526	7.0%
crafty	756	885	885	17.5%
eon	1820	2016	2017	10.8%
equake	465	475	475	2.3%
gap	1298	1836	1882	45.1%
gcc	2316	3644	3646	57.5%
gzip	492	537	537	9.2%
mcf	439	452	451	2.8%
mesa	1337	1758	1769	32.3%
parser	641	804	802	25.2%
perlbmk	1167	1752	1753	50.2%
twolf	773	937	936	21.2%
vortex	1019	1364	1351	32.6%
vpr	668	780	780	16.8%

Table 5: Code size for SPEC2000, in kilobytes.

aligning some instructions that are not indirect control flow targets, we hope to make incremental code size improvement as we refine our implementation. "NaCl" measurements are for statically linked binaries, 32-byte block alignment, and using the `nacljmp` instruction for indirect control flow transfers. To isolate the impact of these three constraints, Figure 4 also shows performance for static linking only, and for static linking and alignment. These comparisons make it clear that alignment is the main factor in cases where overhead is significant. Impact from static linking and sandboxing instruction overhead is small by comparison.

The impact of alignment is not consistent across the benchmark suite. In some cases, alignment appears to improve performance, and in others it seems to make things worse. We hypothesize that alignment of branch targets to 32-byte boundaries sometimes interacts favorably with caches, instruction prefetch buffers, and other facets of processor microarchitecture. These effects are curious but not large enough to justify further investigation. In cases where alignment makes performance worse, one possible factor is code size, as mentioned above. Table 5 shows that increases in NaCl code size due to alignment can be significant, especially in benchmarks like gcc with a large number of static call sites. Similarly, benchmarks with a large amount of control flow branching (e.g., crafty, vortex) have a higher code size growth due to branch target alignment. The incremental code size increase of sandboxing with `nacljmp` is consistently small.

Overall, the performance impact of Native Client on these benchmarks is on average less than 5%. At this level, overhead compares favorably to untrusted native execution.

4.2. Compute/Graphics Performance Tests

We implemented three simple compute+animation benchmarks to test and evaluate our CPU performance for threaded code.[3] They are:

- Earth: a ray-tracing workload, projecting a flat image of the earth onto a spinning globe
- Voronoi: a brute force Voronoi tessellation[4]
- Life: cellular automata simulation of Conway's Game of Life

Sample	Native Client	Linux Executable
Voronoi	12.4	13.9
Earth	14.4	12.6
Life	21.9	19.4

Table 6: Compute/graphics performance tests. Times are user time in seconds.

Executable	1 thread	2 threads	4 threads
Native Client	42.16	22.04	12.4
Linux Binary	46.29	24.53	13.9

Table 7: Voronoi thread performance. Times are user time in seconds.

These workloads have helped us refine and evaluate our thread implementation, in addition to providing a benchmark against standard native compilation.

We used the Linux `time` command to launch and time standalone vs. NaCl release build executables. All measurements are for a Ubuntu Dapper Drake Linux system with a 2.4GHz Intel Q6600 quad core processor. VSYNC was disabled.[5] The normal executables were built using g++ version 4.0.3, the NaCl versions with nacl-g++ version 4.2.2. All three samples were built with `-O3 -mfpmath=sse -msse -fomit-frame-pointer`.

Voronoi used four worker threads and ran for 1000 frames. Earth ran with four worker threads for 1000 frames. Life ran as a single thread, for 5000 frames. Table 6 shows the average for three consecutive runs.

Voronoi ran faster as a NaCl application than as a normal executable. The other two tests, Earth and Life, ran faster as normal executables than their Native Client counterparts. Overall these preliminary measurements suggest that, for these simple test cases, the NaCl thread implementation behaves reasonably compared to Linux. Table 7 shows a comparison of threaded performance between Native Client and a normal Linux executable, using the Voronoi demo. Comparing Native Client to Linux, performance scales comparably with increased thread count.

4.3. H.264 Decoder

We ported an internal implementation of H.264 video decoding to evaluate the difficulty of the porting effort.

3. These benchmarks will be included in our open source distribution.

4. See http://en.wikipedia.org/wiki/Voronoi

5. It is important to disable VSYNC when benchmarking rendering applications. If VSYNC is enabled, the application's rendering thread may be put to sleep until the next vertical sync occurs on the display.

IBM - Ex. 1002-Part 10 of 10, Page 3467
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

The original application converted H.264 video into a raw file format, implemented in about 11K lines of C for the standard GCC environment on Linux. We modified it to play video. The port required about twenty lines of additional C code, more than half of which was error checking code. Apart from rewriting the Makefile, no other modifications were required. This experience is consistent with our general experience with Native Client; legacy Linux libraries that don't inherently require network and disk generally port with minimal effort. Performance of the original and NaCl versions were comparable and limited by video frame-rate.

4.4. Bullet

Bullet [8] is an open source physics simulation system. It has accuracy and modeling features that make it appropriate for real-time applications like computer games. As a complex, performance sensitive legacy code base it is representative of a type of system that we would like to support with Native Client.

The effort required to build Bullet for Native Client was non-trivial but generally straightforward. We used Bullet v2.66 for our experiments which is configurable via autotools [5], allowing us specify use of the NaCl compiler. We also had to build the Jam build system [35], as it is required by the Bullet build. A few #defines also had to be adjusted to eliminate unsupported profiling system calls and other OS specific code. Overall it took a couple of hours of effort to get the library to build for Native Client.

Our performance test used the HelloWorld demo program from the Bullet source distribution, a simulation of a large number of spheres falling and colliding on a flat surface. We compared two builds using GCC v4.2.2 capable of generating NaCl compliant binaries. Measuring 100,000 iterations, we observed 36.5 seconds for the baseline build (-static) vs. 32-byte aligned blocks (as required by Native Client) at 36.1 seconds, or about a 1% speedup for alignment. Incorporating the additional opcode constraints required by Native Client results in runtime of 37.3 seconds, or about a 2% slowdown overall. These numbers were obtained using a two processor dual-core Opteron 8214 with 8GB of memory.

4.5. Quake

We profiled sdlquake-1.0.9 (from www.libsdl.org) using the built-in "timedemo demo1" command. Quake was run at 640x480 resolution on a Ubuntu Dapper Drake Linux box with a 2.4GHz Intel Q6600 quad core CPU. The video system's vertical sync (VSYNC) was disabled. The Linux executable was built using gcc version 4.0.3, and the Native Client version with nacl-gcc version 4.2.2, both with `-O2` optimization.

With Quake, the differences between Native Client and the normal executable are, for practical purposes, indistinguishable. See Table 8 for the comparison. We observed very little non-determinism between runs. The test plays the same sequence of events regardless of frame rate. Slight variances in frame rate can still occur due to the OS thread scheduler and pressure applied to the shared caches from other processes. Although Quake uses software rendering, the performance of the final bitmap transfer to the user's desktop may depend on how busy the video device is.

Run #	Native Client	Linux Executable
1	143.2	142.9
2	143.6	143.4
3	144.2	143.5
Average	143.7	143.3

Table 8: Quake performance comparison. Numbers are in frames per second.

5. Discussion

As described above, Native Client has inner and outer sandboxes, redundant barriers to protect native operating system interfaces. Additional measures such as a CPU blacklist and NaCl module blacklist will also be deployed, and we may deploy whitelists if we determine they are needed to secure the system. We have also considered more elaborate measures, although as they are speculative and unimplemented we don't describe them here. We see public discussion and open feedback as critical to hardening this technology, and informing our decisions about what security mechanisms to include in the system.

We expect Native Client to be well suited to simple, computationally intensive extensions for web applications, specifically in domains such as physical simulation, language processing, and high-performance graphics rendering. Over time, if we can provide convenient DOM access, we hope to enable web-based applications that run primarily in native code, with a thin JavaScript wrapper. There are also applications of this technology outside of the browser; these are beyond our current focus.

We have developed and tested Native Client on Ubuntu Linux, MacOS and Microsoft Windows XP. Overall we are satisfied with the interaction of Native Client with these operating systems. That being said, there are a few areas where operating system support might helpful. Popular operating systems generally require all threads to use a flat addressing model in order to deliver exceptions correctly. Use of segmented memory prevents these systems from interpreting the stack pointer and other essential thread state. Better segment support in the operating system might allow us to resolve this problem and allow for better hardware exception support in untrusted code. If the OS recognized a distinguished thread to receive all exceptions, that would allow Native Client to receive exceptions in a trusted thread.

Native Client would also benefit from more consistent enabling of LDT access across popular x86 operating sys-

IBM - Ex. 1002-Part 10 of 10, Page 3468
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

tems. As an interesting alternative to maintaining system call access as provided by most current systems, a system call for mapping the LDT directly into user space would remove a kernel system call from the path for NaCl thread creation, relevant for modules with a large number of threads.

With respect to programming languages and language implementations, we are encouraged by our initial experience with Native Client and the GNU tool chain, and are looking at porting other compilers. We have also ported two language interpreters, Lua and awk, and are aware of efforts to port other popular interpreted languages. While it would be challenging to support JITted languages such as Java, we are hopeful that Native Client might someday allow developers to use their language of choice in the browser rather than being restricted to only JavaScript.

6. Related Work

Techniques for safely executing 3rd-party code generally fall into three categories: system request moderation, fault isolation (including virtualization), and trust with authentication.

6.1. System Request Moderation

Kernel-based mechanisms such as user-id based access control, systrace [50] and ptrace [60] are familiar facilities on Unix-like systems. Many previous projects have explored use of these mechanisms for containing untrusted code [24], [33], [34], [36], [52], most recently Android [9], [27] from Google and Xax [17] from Microsoft Research. Android uses a sandbox for running 3rd party applications. Each Android application is run as a different Linux user, and a containment system partitions system call activity into permission groups such as "Network communication" and "Your personal information". User acknowledgment of required permissions is required prior to installing a 3rd party application. User separation inherently denies potentially useful intercommunication. To provide intercommunication, Android formed a permissions model atop the Binder interprocess communication mechanism, the Intent sytem and ContentProvider data access model. [9]

Xax is perhaps the most similar work to Native Client in terms of goals, although their implementation approach is quite different, using system call interception based on ptrace on Linux and a kernel device driver on Windows. We considered such a kernel-based approach very early in our work but rejected it as impractical due to concerns about supportability. In particular we note that the Xax Windows implementation requires a kernel-mode device driver that must be updated for each supported Windows build, a scheme we imagine onerous even if implemented by the OS vendor themselves. There are known defects in ptrace containment[6] that Xax does not address. Although the Xax authors do recognize one such issue in their paper, a simple search at Mitre's Common Vulnerabilities and Exposures site[7] documents forty-one different ptrace-related issues. Because of its pure user-space inner sandbox, Native Client is less vulnerable to these difficult kernel issues. Xax is also vulnerable to denial-of-service attacks based on x86 errata that can cause a machine to hang or reboot [31], [38]. Because Native Client examines every instruction and rejects modules with instructions it finds suspect, it significantly reduces the attack surface with respect to invalid instructions, and further it includes relevant mechanism for defending against new exploits should they be found.

Because the Xax sandbox functions at the process boundary, it fails to isolate untrusted code when shared application components such as DLLs are involuntarily injected by the operating system, an issue both for security and for portability of untrusted code. In contrast, the Native Client inner sandbox creates a security sub-domain within a native operating system process. Apart from these security differences we note that Xax does not support threading, which we considered essential given the trend towards multicore CPUs.

The Linux seccomp[8] facility also constrains Linux processes at the system call interface, allowing a process to enter a mode where only _exit(), read(), and write() system calls are permitted.

6.2. Fault Isolation

Native Client applies concepts of software fault isolation and proof-carrying code that have been extensively discussed in the research literature. Our data integrity scheme is a straightforward application of segmented memory as implemented in the Intel 80386 [14]. Our current control flow integrity technique builds on the seminal work by Wahbe, Lucco, Anderson and Graham [62]. Like Wahbe et al., Native Client expresses sandboxing constraints directly in native machine instructions rather than using a virtual machine or other ISA-portable representation. Native Client extends this previous work with specific mechanisms to achieve safety for the x86 [4], [14], [32] ring-3 instruction set architecture (ISA), using several techniques first described by McCamant and Morrisett [40]. Native Client uses a static validator rather than a trusted compiler, similar to validators described for other systems [19], [40], [41], [49], applying the concept of proof-carrying code [46].

After the notion of software fault isolation was popularized by Wahbe et al., researchers described complementary and alternative systems. A few [1], [19], [40], [41],

6. http://www.linuxhq.com/kernel/v2.4/36-rc1/Documentation/ptrace.txt

7. For example, see http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=ptrace

8. See linux/kernel/seccomp.c

IBM - Ex. 1002-Part 10 of 10, Page 3469
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

[49], [56] work directly with x86 machine code. Others are based on intermediate program representations, such as type-safe languages [28], [45], [47], [59], abstract virtual machines [3], [20], [21], [39], or compiler intermediate representations [53]. They use a portable representation, allowing ISA portability but creating a performance obstacle that we avoid by working directly with machine code. A further advantage of expressing sandboxing directly in machine code is that it does not require a trusted compiler. This greatly reduces the size of the trusted computing base [61], and obviates the need for cryptographic signatures from the compiler. Apart from simplifying the security implementation, this has the further benefit in Native Client of opening the system to 3rd-party tool chains.

Compared to Native Client, CFI [1] provides finer-grained control flow integrity. Whereas our system only guarantees indirect control flow will target an aligned address in the text segment, CFI can restrict a specific control transfer to a fairly arbitrary subset of known targets. While this more precise control is possibly useful in some scenarios, such as insuring integrity of translation from a high-level language, it is not useful for Native Client, since we intend to permit quite arbitrary control flow, even hand-coded assembler, as long as execution remains in known text and targets are aligned. At the same time, CFI overhead is a factor of three higher on average (15% vs. 5% on SPEC2000), not surprising since its instrumentation sequences are much longer than Native Client's, both in terms of size and instruction count. XFI [19] adds data sandboxing to CFI control flow checks, with additional overhead. By contrast Native Client gets data integrity for free from x86 segments.

Other recent systems have explored mechanisms for enabling safe side effects with measured trust. NaCl resource descriptors are analogous to capabilities in systems such as EROS [55]. Singularity channels [30] serve an analogous role. DTrace [11], Systemtap [49] and XFI [19] have related mechanisms.

A number of projects have explored isolating untrusted kernel extensions. SPIN and VINO take different approaches to implementing safety. SPIN chose a type-safe language, Modula-3 [47], together with a trusted compiler tool chain, for implementing extensions. VINO, like Native Client and the original work by Wahbe et al., used software fault isolation based on sandboxing of machine instructions. Like Native Client, VINO used a modified compilation toolchain to add sandboxing instructions to x86 machine code, using C++ for implementing extensions. Unlike Native Client, VINO had no binary validator, relying on a trusted compiler. We note that a validator for VINO would be more difficult than that of Native Client, as its validator would have had to enforce data reference integrity, achieved in Native Client with 80386 segments.

The Nooks system [58] enhances operating system kernel reliability by isolating trusted kernel code from untrusted device driver modules using a transparent OS layer called the Nooks Isolation Manager (NIM). Like Native Client, NIM uses memory protection to isolate untrusted modules. As the NIM operates in the kernel, x86 segments are not available. The NIM instead uses a private page table for each extension module. To change protection domains, the NIM updates the x86 page table base address, an operation that has the side effect of flushing the x86 Translation Lookaside Buffer (TLB). In this way, NIM's use of page tables suggests an alternative to segment protection as used by Native Client. While a performance analysis of these two approaches would likely expose interesting differences, the comparison is moot on the x86 as one mechanism is available only within the kernel and the other only outside the kernel. A critical distinction between Nooks and Native Client is that Nooks is designed only to protect against unintentional bugs, not abuse. In contrast, Native Client must be resistant to attempted deliberate abuse, mandating our mechanisms for reliable x86 disassembly and control flow integrity. These mechanisms have no analog in Nooks.

There are many environments based on a virtual-machine architecture that provide safe execution and some fraction of native performance [3], [6], [7], [20], [28], [39], [53], [63]. While recognizing the excellent fault-isolation provided by these systems, we made a deliberate choice against virtualization in Native Client, as it is generally inconsistent with, or irrelevant to, our goals of OS neutrality, browser neutrality, and peak native performance.

More recently, kernel extensions have been used for operating system monitoring. DTrace [11] incorporated a VM interpreter into the Solaris kernel for safe execution, and provided a set of kernel instrumentation points and output facilities analogous to Native Client's safe side effects. Systemtap [49] provides similar capabilities to DTrace, but uses x86 native code for extensions rather than an interpreted language in a VM.

6.3. Trust with Authentication

Perhaps the most prevalent example of using native code in web content is Microsoft's ActiveX [15]. ActiveX controls rely on a trust model to provide security, with controls cryptographically signed using Microsoft's proprietary Authenticode system [43], and only permitted to run once a user has indicated they trust the publisher. This dependency on the user making prudent trust decisions is commonly exploited. ActiveX provides no guarantee that a trusted control is safe, and even when the control itself is not inherently malicious, defects in the control can be exploited, often permitting execution of arbitrary code. To mitigate this issue, Microsoft maintains a blacklist of controls deemed unsafe [42]. In contrast, Native Client is designed to prevent such exploitation, even for flawed NaCl modules.

IBM - Ex. 1002-Part 10 of 10, Page 3470
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

7. Conclusions

This paper has described Native Client, a system for incorporating untrusted x86 native code into an application that runs in a web browser. In addition to creating a barrier against undesirable side effects, NaCl modules are portable both across operating systems and across web browsers, and supports performance-oriented features such as threading and vectorization instructions. We believe the NaCl inner sandbox is extremely robust; regardless we provide additional redundant mechanisms to provide defense-in-depth.

In our experience we have found porting existing Linux/gcc code to Native Client is straightforward, and that the performance penalty for the sandbox is small, particularly in the compute-bound scenarios for which the system is designed.

By describing Native Client here and making it available as open source, we hope to encourage community scrutiny and contributions. We believe this feedback together with our continued diligence will enable us to create a system that achieves a superior level of safety than previous native code web technologies.

Acknowledgments

Many people have contributed to the direction and the development of Native Client; we acknowledge a few of them here. The project was conceived based on an idea from Matt Papakipos. Jeremy Lau, Brad Nelson, John Grabowski, Kathy Walrath and Geoff Pike have made valuable contributions to the implementation and evaluation of the system. Thanks also to Danny Berlin, Chris DiBona, and Rebecca Ward. We thank Sundar Pichai and Henry Bridge for their role in shaping the project direction. We'd also like to thank Dick Sites for his thoughtful feedback on an earlier version of this paper.

References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity: Principles, implementations, and applications. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, November 2005.

[2] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A new kernel foundation for UNIX development. pages 93–112, 1986.

[3] A. Adl-Tabatabai, G. Langdale, S. Lucco, and R. Wahbe. Efficient and language-independent mobile programs. *SIGPLAN Not.*, 31(5):127–136, 1996.

[4] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual, Volume 1: Application Programming*. Advanced Micro Devices, September 2007. Publication number: 24592.

[5] Autoconf. http://www.gnu.org/software/autoconf/.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *19th ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[7] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447, November 1997.

[8] Bullet physics SDK. http://www.bulletphysics.com.

[9] J. Burns. Developing secure mobile applications for android. http://isecpartners.com/files/iSEC_Securing_Android_Apps.pdf, 2008.

[10] K. Campbell, L. Gordon, M. Loeb, and L. Zhou. The economic cost of publicly announced information security breaches: empirical evidence from the stock market. *Journal of Computer Security*, 11(3):431–448, 2003.

[11] B. Cantrill, M. Shapiro, and A. Leventhal. Dynamic instrumentation of production systems. In *2004 USENIX Annual Technical Conference*, June 2004.

[12] D. R. Cheriton. The V distributed system. *Communications of the ACM*, 31:314–333, 1988.

[13] F. B. Cohen. Defense-in-depth against computer viruses. *Computers and Security*, 11(6):565–584, 1993.

[14] J. Crawford and P. Gelsinger. *Programming 80386*. Sybex Inc., 1991.

[15] A. Denning. *ActiveX Controls Inside Out*. Microsoft Press, May 1997.

[16] Directorate for Command, Control, Communications and Computer Systems, U.S. Department of Defense Joint Staff. Information assurance through defense-in-depth. Technical report, Directorate for Command, Control, Communications and Computer Systems, U.S. Department of Defense Joint Staff, February 2000.

[17] J. R. Douceur, J. Elson, J. Howell, and J. R. Lorch. Leveraging legacy code to deploy desktop applications on the web. In *Proceedings of the 2008 Symposium on Operating System Design and Implementation*, December 2008.

[18] M. Eisler (editor). XDR: External data representation. Internet RFC 4506, 2006.

[19] U. Erlingsson, M. Abadi, M. Vrable, M. Budiu, and G. Necula. XFI: Software guards for system address spaces. In *OSDI '06: 7th Symposium on Operating Systems Design And Implementation*, pages 75–88, November 2006.

[20] B. Ford. VXA: A virtual architecture for durable compressed archives. In *USENIX File and Storage Technologies*, December 2005.

[21] B. Ford and R. Cox. Vx32: Lightweight user-level sandboxing on the x86. In *2008 USENIX Annual Technical Conference*, June 2008.

IBM - Ex. 1002-Part 10 of 10, Page 3471
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

[22] The GNU compiler collection. http://gcc.gnu.org.

[23] GNU binutils. http://www.gnu.org/software/binutils/.

[24] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A secure enviroment for untrusted helper applications. In *Proceedings of the 6th USENIX Security Symposium*, 1996.

[25] D. Golub, A. Dean, R. Forin, and R. Rashid. UNIX as an application program. In *Proceedings of the Summer 1990 USENIX Conference*, pages 87–95, 1990.

[26] Google Inc. Protocol buffers. http://code.google.com/p/protobuf/.

[27] Google Inc. Android—an open handset alliance project. http://code.google.com/android, 2007.

[28] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification*. Addison-Wesley, 2000.

[29] B. Gough and R. Stallman. *An Introduction to GCC*. Network Theory, Ltd., 2004.

[30] G. Hunt, J. Larus, M. Abadi, M. Aiken, P. Barham, M. Fahndrich, C. Hawblitzel, O. Hodson, S. Levi, N. Murphy, B. Steensgaard, D. Tarditi, T. Wobber, and B. Zill. An overview of the Singularity project. Technical Report MSR-TR-2005-135, Microsoft Research, October 2005.

[31] Intel Corporation. Intel Pentium processor invalid instruction errata overview. http://support.intel.com/support/processor/pentium/ppiie/index.html.

[32] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developers Manual, Volume 1: Basic Architecture*. Intel Corporation, August 2007. Order Number: 253655-024US.

[33] S. Ioannidis and S. M. Bellovin. Building a secure web browser. In *USENIX Annual Technical Conference, FREENIX Track*, pages 127–134, 2001.

[34] S. Ioannidis, S. M. Bellovin, and J. M. Smith. Sub-operating systems: A new approach to application security. In *Proceedings of the 10th ACM SIGOPS European Workshop*, 2002.

[35] Jam 2.1 user's guide. http://javagen.com/jam/.

[36] C. Jensen and D. Hagimont. Protection wrappers: a simple and portable sandbox for untrusted applications. In *Proceedings of the 8th ACM SIGOPS European Systems Conference*, pages 104–110, 1998.

[37] W. Joy, E. Cooper, R. Fabry, S. Leffler, K. McKusick, and D. Mosher. 4.2 BSD system manual. Technical report, Computer Systems Research Group, University of California, Berkeley, 1983.

[38] K. Kaspersky and A. Chang. Remote code execution through Intel CPU bugs. In *Hack In The Box (HITB) 2008 Malaysia Conference*.

[39] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Prentice Hall, 1999.

[40] S. McCamant and G. Morrisett. Efficient, verifiable binary sandboxing for a CISC architecture. Technical Report MIT-CSAIL-TR-2005-030, 2005.

[41] S. McCamant and G. Morrisett. Evaluating SFI for a CISC architecture. In *15th USENIX Security Symposium*, August 2006.

[42] Microsoft Corporation. The kill-bit faq - part 1 of 3. *Microsoft Security Vulnerability Research and Defense (Blog)*.

[43] Microsoft Corporation. Signing and checking code with Authenticode. http://msdn.microsoft.com/en-us/library/ms537364(VS.85).aspx.

[44] Microsoft Corporation. Structured exception handling. http://msdn.microsoft.com/en-us/library/ms680657(VS.85).aspx, 2008.

[45] G. Morrisett, K. Crary, N. Glew, and D. Walker. Stack-based typed assembly language. *Journal of Functional Programming*, 12(1):43–88, 2002.

[46] G. Necula. Proof carrying code. In *Principles of Programming Languages*, 1997.

[47] G. Nelson (editor). *System Programming in Modula-3*. Prentice-Hall, 1991.

[48] Netscape Corporation. Gecko plugin API reference. http://developer.mozilla.org/en/docs/Gecko_Plugin_API_Reference.

[49] V. Prasad, W. Cohen, F. Eigler, M. Hunt, J. Keniston, and J. Chen. Locating system problems using dynamic instrumentation. In *2005 Ottawa Linux Symposium*, pages 49–64, July 2005.

[50] N. Provos. Improving host security with system call policies. In *USENIX Security Symposium*, August 2003.

[51] J. Reinders. *Intel Thread Building Blocks*. O'Reilly & Associates, 2007.

[52] J. G. Richard West. User-level sandboxing: a safe and efficient mechanism for extensibility. Technical Report TR-2003-014, Boston University, Computer Science Department, Boston, MA, 2003.

[53] J. Richter. *CLR via C#, Second Edition*. Microsoft Press, 2006.

[54] M. Savage. Cost of computer viruses top $10 billion already this year. *ChannelWeb*, August 2001.

[55] J. Shapiro, J. Smith, and D. Farber. EROS: a fast capability system. In *Symposium on Operating System Principles*, pages 170–185, 1999.

[56] C. Small. MiSFIT: A tool for constructing safe extensible C++ systems. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies*, June 1997.

[57] B. Stroustrup. *The C++ Programming Language: Second Edition*. Addison-Wesley, 1997.

IBM - Ex. 1002-Part 10 of 10, Page 3472
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

[58] M. Swift, M. Annamalai, B. Bershad, and H. Levy. Recovering device drivers. In *6th USENIX Symposium on Operating Systems Design and Implementation*, December 2004.

[59] D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper, and P. Lee. TIL: a type-directed optimizing compiler for ML. In *PLDI '96: Proceedings of the ACM SIGPLAN 1996 conference on Programming language design and implementation*, pages 181–192, New York, NY, USA, 1996. ACM.

[60] W. Tarreau. ptrace documentation. http://www.linuxhq.com/kernel/v2.4/36-rc1/Documentation/ptrace.txt, 2007.

[61] U. S. Department of Defense, Computer Security Center. Trusted computer system evaluation criteria, December 1985.

[62] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient software-based fault isolation. *ACM SIGOPS Operating Systems Review*, 27(5):203–216, December 1993.

[63] C. Waldspurger. Memory resource management in VMware ESX Server. In *5th Symposium on Operating Systems Design and Implementation*, December 2002.

[64] *Document Object Model (DOM) Level 1 Specification*. Number REC-DOM-Level-1-19981001. World Wide Web Consortium, October 1998.

IBM - Ex. 1002-Part 10 of 10, Page 3473
IPR2025-00379 (IBM v. Croga)
Authorized licensed use limited to: United States Patent and Trademark Office. Downloaded on March 11,2021 at 13:18:50 UTC from IEEE Xplore. Restrictions apply.

DOCKET NO.: LCOM_ENT_CLOUD_US01 **PATENT**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: **L3 Technologies, Inc.**	**Confirmation No.: 2440**
Application No.: 16/141,378	**Group Art Unit: 2492**
Filing Date: September 25, 2018	**Examiner: KORSAK, OLEG**

For: NETWORK ISOLATION WITH CLOUD NETWORKS

Filed Via EFS

SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT

Pursuant to 37 CFR § 1.56 and in accordance with 37 CFR §§ 1.97-1.98, information relating to the above-identified application is hereby disclosed. Inclusion of information in this statement is not to be construed as an admission that this information is material as that term is defined in 37 CFR § 1.56(b).

☐ **IDS Filed Under 37 CFR 1.97(b)**

In accordance with § 1.97(b), since this Information Disclosure Statement is being filed either within three months of the filing date of the above-identified application, within three months of the date of entry into the national stage of the above identified application as set forth in § 1.491, before the mailing date of a first Office Action on the merits of the above-identified application, or before the mailing date of a first Office Action after the filing of request for continued examination under § 1.114, no additional fee is required.

☒ **IDS filed Under 37 CFR 1.97(c)**

In accordance with § 1.97(c), this Information Disclosure Statement is being filed after the period set forth in § 1.97(b) above but before the mailing date of either a Final Action under § 1.116 or a Notice of Allowance under § 1.311, or before an action that otherwise closes prosecution in the application, therefore:

☐ Certification in Accordance with § 1.97(e) is attached; or

☒ The fee of **$260.00** as set forth in § 1.17(p) is attached.

IBM - Ex. 1002-Part 10 of 10, Page 3474
IPR2025-00379 (IBM v. Croga)

☐ **IDS filed Under 37 CFR 1.97(d)**

In accordance with § 1.97(d), this Information Disclosure Statement is being filed after the mailing date of either a Final Action under § 1.113 or a Notice of Allowance under § 1.311 but before, or simultaneously with, the payment of the Issue Fee, therefore included are: Certification in Accordance with § 1.97(e); and the submission fee of **$240.00** as set forth in § 1.17(p).

CONTENT OF IDS PURSUANT TO 37 CFR 1.98

☒ Copies of reference numbers 36-125 listed on the attached Form PTO-1449 are not required to be submitted pursuant to 37 CFR § 1.98(a)(2)(ii).

☒ Copies of reference numbers 126-139 listed on the attached Form PTO-1449 are enclosed herewith.

☐ Copy of reference number is not being submitted because it was previously cited by or submitted to the U.S. Patent and Trademark Office in patent application number , filed for which a claim for priority under 35 U.S.C. § 120 has been made in the instant application.

☐ The month of publication for reference numbers is not available. However, the year of publication for these references is sufficiently earlier than the effective US filing date and any foreign priority date so that the particular month of publication is not in issue pursuant to 37 CFR § 1.98(b).

REFERENCES IN A LANGUAGE OTHER THAN ENGLISH

☐ The following documents are not in the English language. Accordingly, a concise explanation of the relevance of the document was incorporated in the specification passages identified below, the document was identified in a foreign communication as identified below or an English language counterpart application has been provided as indicated below.

Foreign Language Document	Cite No.	Pages of Reference in Specification or Relevance of Document

Foreign Language Document	Cite No.	English Language Counterpart	Cite No.

IBM - Ex. 1002-Part 10 of 10, Page 3475
IPR2025-00379 (IBM v. Croga)

☐ **CERTIFICATION IN ACCORDANCE WITH § 1.97(e)**

I hereby certify that:

☐ Each item of information contained in this information disclosure statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this information disclosure statement.

☐ No item of information contained in this information disclosure statement was cited in a communication from a foreign patent office in a counterpart foreign application, and, to the knowledge of the person signing the certification after making reasonable inquiry, no item of information contained in this information disclosure statement was known to any individual designated in § 1.56(c) more than three months prior to the filing of this information disclosure statement.

Please charge any deficiency or credit any overpayment to Deposit Account No. 50-5519.

Date: September 13, 2021

/Jeung Huh/
Jeung (Jacob) Huh
Registration No. 76,913

CONDO ROCCIA KOPTIW LLP
1800 JFK Blvd., Suite 1700
Philadelphia, PA 19103
Telephone: (215) 558-5740
Facsimile: (215) 558-5676

Electronic Patent Application Fee Transmittal

Application Number:	16141378
Filing Date:	25-Sep-2018
Title of Invention:	NETWORK ISOLATION WITH CLOUD NETWORKS
First Named Inventor/Applicant Name:	Glenn Coleman
Filer:	Jeung Huh/Melissa Inzaris
Attorney Docket Number:	LCOM_ENT_CLOUD_US01

Filed as Large Entity

Filing Fees for Utility under 35 USC 111(a)

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Extension - 3 months with $0 paid	1253	1	1480	1480
Miscellaneous:				
SUBMISSION- INFORMATION DISCLOSURE STMT	1806	1	260	260
		Total in USD ($)		**1740**

Electronic Acknowledgement Receipt

EFS ID:	43757581
Application Number:	16141378
International Application Number:	
Confirmation Number:	2440
Title of Invention:	NETWORK ISOLATION WITH CLOUD NETWORKS
First Named Inventor/Applicant Name:	Glenn Coleman
Customer Number:	158266
Filer:	Jeung Huh/Melissa Inzaris
Filer Authorized By:	Jeung Huh
Attorney Docket Number:	LCOM_ENT_CLOUD_US01
Receipt Date:	14-SEP-2021
Filing Date:	25-SEP-2018
Time Stamp:	11:40:46
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	CARD
Payment was successfully received in RAM	$1740
RAM confirmation Number	E20219DB41132600
Deposit Account	
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		IMS-MT-C5-0149_LCOM_ENT_CLOUD_US01_AS_FILED_Reply_to_NFOA_dated_03_23_21.pdf	159427 70226b05d417fb20deaf3d2ea395ad2ce9316005	yes	9

Multipart Description/PDF files in .zip description

Document Description	Start	End
Amendment/Req. Reconsideration-After Non-Final Reject	1	1
Claims	2	6
Applicant Arguments/Remarks Made in an Amendment	7	9

Warnings:

Information:

2	Information Disclosure Statement (IDS) Form (SB08)	LCOM_ENT_CLOUD_US01_AS_FILED_SIDS_1449.pdf	214564 24a28970da2e34631ddf20bc24c141a718b030cd	no	12

Warnings:

Information:

This is not an USPTO supplied IDS fillable form

3	Extension of Time	LCOM_ENT_CLOUD_US01_AS_FILED_EOT.pdf	139462 d7cf43d580b26c6f8047eeb87e282305317b3ef8	no	2

Warnings:

Information:

4	Non Patent Literature	126_NPL_search_terms.pdf	72923 3675ea9e84950a1edabd47edbee8a96125232bc9	no	1

Warnings:

Information:

5	Non Patent Literature	127_Azab_SICE_A_Hardware_Level_Strongly_Isolated_Computing_Environment.pdf	2210763 01861d95ef60fa762557861c9f9bbcc547c6a943	no	14
Warnings:					
Information:					
6	Non Patent Literature	128_Georgia_computes_Improving_the_Computing_Education_Pipeline.pdf	690914 03ea8d54cfac04d1e23f31e3a42ad20dcf1137a3	no	5
Warnings:					
Information:					
7	Non Patent Literature	129_Gupta_A_secure_architecture_design_based_on_application_isolation_code_minimization.pdf	1068617 14992c6c9f87abd39edafe65f8d45470b38fc847	no	7
Warnings:					
Information:					
8	Non Patent Literature	130_Ul_haq_Design_and_Implementation_of_Sandbox_Technique_for_Isolated_Applications.pdf	979755 8fd27b89a78fca963dd7ef6be9b955fe7dbc584a	no	5
Warnings:					
Information:					
9	Non Patent Literature	131_Hsu_Enforcing_Least_Privilege_Memory_Views_for_Multithreaded_Applications.pdf	2093183 a47957c275df633ac6beac606c799ea8ea2710c6	no	13
Warnings:					
Information:					
10	Non Patent Literature	132_Jauernig_Trusted_Execution_Environments_Properties_Application_and_Challenges.pdf	1071996 3517ac05fb2214722003ac05234bc9c413bbf70a	no	5
Warnings:					
Information:					
11	Non Patent Literature	133_Jung_AnArchitectureFor_2014_8pp.pdf	4131513 86d957682c785cffccb5f541eecf4100f1274a5b	no	8
Warnings:					
Information:					

12	Non Patent Literature	134_Kostiainen_Dedicated_Security_Chips_in_the_Age_of_Secure_Enclaves.pdf	2053916 112425b0f77472a9c34b1858cf96ec351be764c2	no	9
Warnings:					
Information:					
13	Non Patent Literature	135_Phuoc_Computing_the_Real_Isolated_Points_of_an_Algebraic_Hypersurface.pdf	554329 1425b14844c22b3ae6c2a71794dabde164de255e	no	12
Warnings:					
Information:					
14	Non Patent Literature	136_Liu_Thwarting_Memory_Disclosure_with_Efficient_Hypervisor-enforced_Intra-domain_Isolation.pdf	2178885 3c293a7a38a87146ea80e11a74d184108dcd0f5b	no	13
Warnings:					
Information:					
15	Non Patent Literature	137_Srivastava_TamperResistant_2008_20pp.pdf	3825847 ec43b95997b9976b1112f8c70a22ce7d6d722da8	no	20
Warnings:					
Information:					
16	Non Patent Literature	138_Yadav_Impact_and_Analysis_of_Virtual_Workspace_on_Grid_Computing.pdf	1001633 946b652326a094345171bed50771768303207678	no	9
Warnings:					
Information:					
17	Non Patent Literature	139_Yee_Native_Client_A_Sandbox_for_Portable_Untrusted_x86_Native_Code.pdf	1855308 d51b0912dbef3bcc4f6570496b801637929484c4	no	15
Warnings:					
Information:					
18	Transmittal Letter	LCOM_ENT_CLOUD_US01_AS_FILED_SIDS_Transmittal.pdf	93132 06d25519496d073c07b0be5be75303e66ba08a9e	no	3
Warnings:					
Information:					

19	Fee Worksheet (SB06)	fee-info.pdf	40589 081aceee5259756ab0d94f1be8cb2b4292d770dc	no	2

Warnings:	
Information:	
Total Files Size (in bytes):	24436756

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (09-11)
Approved for use through 1/31/2014. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875	Application or Docket Number 16/141,378	Filing Date 09/25/2018	☐ To be Mailed

ENTITY: ☑ LARGE ☐ SMALL ☐ MICRO

APPLICATION AS FILED - PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA		RATE ($)	FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A		N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A		N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A		N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*		x $100 =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*		x $460 =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $310 ($155 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).				
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))					
* If the difference in column 1 is less than zero, enter "0" in column 2.				TOTAL	

APPLICATION AS AMENDED - PART II

		(Column 1)		(Column 2)	(Column 3)			
AMENDMENT	09/14/2021	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	* 19	Minus	** 20	= 0		x $100 =	0
	Independent (37 CFR 1.16(h))	* 3	Minus	*** 3	= 0		x $480 =	0
	☐ Application Size Fee (37 CFR 1.16(s))							
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							
							TOTAL ADD'L FEE	0

		(Column 1)		(Column 2)	(Column 3)			
AMENDMENT		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=		x $0 =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=		x $0 =	
	☐ Application Size Fee (37 CFR 1.16(s))							
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							
							TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.	LIE
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".	/PEARLIE A FENNELL/
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".	
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.	

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
16/141,378	09/25/2018	Glenn Coleman	LCOM_ENT_CLOUD_US01	2440

158266 7590 09/20/2021

Condo Roccia Koptiw LLP
1800 JFK Boulevard
Suite 1700
Philadelphia, PA 19103

EXAMINER	
KORSAK, OLEG	

ART UNIT	PAPER NUMBER
2492	

MAIL DATE	DELIVERY MODE
09/20/2021	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of Non-Compliant Amendment (37 CFR 1.121)	Application No. 16/141,378	Applicant(s) Coleman et al.	
	Examiner OLEG KORSAK	Art Unit 2492	AIA (FITF) Status Yes

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

The amendment document filed on 09/14/2021 is considered non-compliant because it has failed to meet the requirements of 37 CFR 1.121 or 1.4. In order for the amendment document to be compliant, correction of the following item(s) is required.

THE FOLLOWING MARKED (X) ITEM(S) CAUSE THE AMENDMENT DOCUMENT TO BE NON-COMPLIANT:

- ☐ 1. Amendments to the specification:
  - ☐ A. Amended paragraph(s) do not include markings.
  - ☐ B. New paragraph(s) should not be underlined.
  - ☐ C. Other _____ .
- ☐ 2. Abstract:
  - ☐ A.Not presented on a separate sheet. 37 CFR 1.72.
  - ☐ B.Other _____ .
- ☐ 3. Amendments to the drawings:
  - ☐ A. The drawings are not properly identified in the top margin as "Replacement Sheet", "New Sheet", or "Annotated Sheet" as required by 37 CFR 1.121(d).
  - ☐ B. The practice of submitting proposed drawing correction has been eliminated. Replacement drawings showing amended figures, without markings, in compliance with 37 CFR 1.84 are required.
  - ☐ C. Other _____ .
- ☑ 4. Amendments to the claims:
  - ☐ A. A complete listing of all of the claims is not present.
  - ☐ B. The listing of claims does not include the text of all pending claims (including withdrawn claims)
  - ☑ C. Each claim has not been provided with the proper status identifier, and as such, the individual status of each claim cannot be identified. Note: the status of every claim must be indicated after its claim number by using one of the following status identifiers: (Original), (Currently amended), (Canceled), (Previously presented), (New), (Not entered), (Withdrawn) and (Withdrawn-currently amended).
  - ☐ D. The claims of this amendment paper have not been presented in ascending numerical order.
  - ☑ E. Other: See Continuation Sheet .
- ☐ 5. Other (e.g., the amendment is unsigned or not signed in accordance with 37 CFR 1.4):

_____

For further explanation of the amendment format required by 37 CFR 1.121, see MPEP § 714.

TIME PERIODS FOR FILING A REPLY TO THIS NOTICE:

1. Applicant is given **no new time period if the non-compliant amendment is an** after-final amendment or an amendment filed after allowance. If applicant wishes to resubmit the non-compliant after-final amendment with corrections, the **entire corrected amendment** must be resubmitted.

2. Applicant is given **two months** from the mail date of this notice to supply the correction, if the non-compliant amendment is one of the following: a preliminary amendment, a non-final amendment (including a submission for a request for continued examination (RCE) under 37 CFR 1.114), a supplemental amendment filed within a suspension period under 37 CFR 1.103(a) or (c), and an amendment filed in response to a *Quayle* action. If any of above boxes 1. to 4. are checked, the correction required is only the **corrected section** of the non-compliant amendment in compliance with 37 CFR 1.121.

   Extensions of time are available under 37 CFR 1.136(a) only if the non-compliant amendment is a non-final amendment or an amendment filed in response to a *Quayle* action.

   Failure to timely respond to this notice will result in:
   Abandonment of the application if the non-compliant amendment is a non-final amendment or an amendment filed in response to a *Quayle* action; or
   Non-entry of the amendment if the non-compliant amendment is a preliminary amendment or supplemental amendment.

/OLEG KORSAK/ Primary Examiner, Art Unit 2492	

Continuation of 4(e) Other: Applicant stated that no claims have been amended, however Claim 19 with status identifier (Previously Presented) has limitation "in order" with line through.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: **L3 Technologies, Inc.**	**Confirmation No.: 2440**
Application No.: 16/141,378	**Group Art Unit: 2492**
Filing Date: September 25, 2018	**Examiner: KORSAK, OLEG**

For: NETWORK ISOLATION WITH CLOUD NETWORKS

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

REPLY PURSUANT TO NOTICE OF NON-COMPLIANT AMENDMENT 37 CFR § 1.121

In response to the Official Action dated **September 20, 2021**, reconsideration is respectfully requested in view of the amendments and/or remarks as indicated below:

☐ **Amendments to the Specification** begin on page N/A of this paper.

☒ **Amendments to the Claims** are reflected in the listing of the claims which begins on page 2 of this paper.

☐ **Amendments to the Drawings** begin on page N/A of this paper and include an attached replacement sheet.

☒ **Remarks** begin on page 7 of this paper.

☒ The Commissioner is hereby authorized to charge any fee deficiency, charge any additional fees, or credit any overpayment of fees, associated with this application in connection with this filing, or any future filing, submitted to the U.S. Patent and Trademark Office during the pendency of this application, to Deposit Account No. 50-5519.

IBM - Ex. 1002-Part 10 of 10, Page 3488
IPR2025-00379 (IBM v. Croga)

This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Previously Presented) A host computer system:

a memory; and

a processor configured to:

implement a workspace, wherein the workspace is configured to enable operation of a first set of one or more applications or processes via a first memory space;

implement an isolated computing environment, the isolated computing environment using a host operating system and comprising a sandboxed computing environment that uses a second memory space to enable operation of a second set of one or more applications or processes, wherein the isolated computing environment is configured to access an Internet-based cloud service via at least one application of the second set of one or more applications or processes;

isolate the isolated computing environment from the workspace using an internal isolation firewall;

authenticate the isolated computing environment with an authentication device; and

communicate with a proxy server to access the Internet-based cloud service to allow communication between the at least one application and the Internet-based cloud service when the isolated computing environment has been authenticated.

2. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to access the Internet-based cloud service via one or more of a local area network (LAN), a wide area network (WAN), a cellular network, a Wi-Fi network, a digital subscribers line (DSL) network, a cloud computing network, a data center, or a satellite network.

IBM - Ex. 1002-Part 10 of 10, Page 3489
IPR2025-00379 (IBM v. Croga)

3. (Previously Presented) The host computer system of claim 1, wherein the Internet-based cloud service comprises one or more of an Infrastructure as a Service (IaaS), a Software as a Service (SaaS), or a Platform as a Service (PaaS).

4. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to select the proxy server from a plurality of proxy servers, the plurality of proxy servers comprising one or more of an Internet-based proxy server or a local network-based proxy server.

5. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to determine whether to send web traffic to a local network-based proxy server or an Internet-based proxy server based on a location of the host computer system.

6. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to determine whether to send web traffic to a local network-based proxy server or an Internet-based proxy server based on an identity of a network to which the host computer system is connected.

7. (Previously Presented) The host computer system of claim 1, wherein the processor is configured to determine whether to send web traffic to a local network-based proxy server or an Internet-based proxy server based on an identity of the at least one application running in the isolated computing environment.

8. (Previously Presented) The host computer system of claim 1, wherein the internal isolation firewall is configured to segregate the first memory space from the second memory space, and wherein the processor is configured to enforce policies for communication between the first memory space and the second memory space using the internal isolation firewall.

IBM - Ex. 1002-Part 10 of 10, Page 3490
IPR2025-00379 (IBM v. Croga)

9. (Previously Presented) The host computer system of claim 8, wherein the internal isolation firewall is further configured to generate a message to prompt a user of the host computer system, the message indicating whether to allow communication between the first memory space and the second memory space.

10. (Previously Presented)A cloud computing system comprising:

a cloud computing network comprising a server, wherein the server is configured to:

implement a workspace configured to enable operation of a first set of one or more applications or processes;

implement an isolated computing environment using a host operating system and comprising a sandboxed computing environment and being configured to enable operation of a second set of one or more applications or processes;

isolate the isolated computing environment from the workspace using an internal isolation firewall; and

authenticate the isolated computing environment with an authentication device; and

a proxy configured to control communication between the server and an Internet resource, wherein the proxy is configured to allow communication between the second set of one or more applications or processes running in the isolated computing environment and the Internet resource when the isolated computing environment has been authenticated and to block communication between the first set of one or more applications or processes running in the workspace and the Internet resource.

11. (Previously Presented) The cloud computing system of claim 10, wherein the server is configured to perform at least one processing task on behalf of a client device.

IBM - Ex. 1002-Part 10 of 10, Page 3491
IPR2025-00379 (IBM v. Croga)

12. (Previously Presented) The cloud computing system of claim 11, wherein the proxy is a first proxy, and the server is configured to determine whether to send traffic to the first proxy or a second proxy.

13. (Previously Presented) The cloud computing system of claim 12, wherein the server is configured to determine whether to send traffic to the first proxy or the second proxy based on user-provided credentials.

14. (Previously Presented) The cloud computing system of claim 10, further comprising a border firewall configured to segregate the cloud computing network from one or more Internet destinations.

15. (Previously Presented) The cloud computing system of claim 10, wherein the internal isolation firewall is further configured to allow communication between the workspace and the isolated computing environment on a condition that the communication is in a predetermined format.

16. (Previously Presented) The cloud computing system of claim 11, wherein the server is further configured to receive input data from the client device and send one or more data communications to the client device.

17. (Previously Presented) The cloud computing system of claim 11, wherein the at least one processing task performed on behalf of the client device comprises a processing task for an Infrastructure as a Service (IaaS), a Software as a Service (SaaS), or a Platform as a Service (PaaS).

18. (Canceled)

IBM - Ex. 1002-Part 10 of 10, Page 3492
IPR2025-00379 (IBM v. Croga)

19. (Previously Presented) A method implemented in a host computer system, the method comprising:

implementing a workspace, wherein the workspace is configured to enable operation of a first set of one or more applications or processes via a first memory space;

implementing an isolated computing environment, the isolated computing environment using a host operating system and comprising a sandboxed computing environment using a second memory space to enable operation of a second set of one or more applications or processes, wherein the isolated computing environment is configured to access an Internet-based cloud service via at least one application of the second set of one or more applications or processes;

isolating the isolated computing environment from the workspace using an internal isolation firewall;

authenticate the isolated computing environment with an authentication device; and

communicating with a proxy server to access the Internet-based cloud service to allow communication between the at least one application and the Internet-based cloud service when the isolated computing environment has been authenticated.

20. (Original) The method of claim 19, wherein the Internet-based cloud service comprises one or more of an Infrastructure as a Service (IaaS), a Software as a Service (SaaS), or a Platform as a Service (PaaS).

IBM - Ex. 1002-Part 10 of 10, Page 3493
IPR2025-00379 (IBM v. Croga)

REMARKS

In response to the Examiner's Notice of Non-Compliant Amendment dated September 20, 2021, the undersigned is resubmitting this response. In the previous response dated September 13, 2021, due to a good faith clerical error in updating the claims to reflect the amendments of March 8, 2021, claim 19 had a typographical error, in that the term "in order" with line through was not removed. This has been corrected and no claims have been amended and are indicated as allowable over prior art.

Claims 1-17, 19, and 20 are pending in the application. No claims have been amended, canceled, or added. No new matter has been added.

Allowable Subject Matter

Applicant thanks Examiner Korsak for indicating that claims 1-17, 19, and 20 are allowable over prior art.

Double Patenting Rejections over U.S. Patent No. 10,544,475 and 10,588,798

Claims 1-17, 19, and 20 are provisionally rejected under the judicially created doctrine of obviousness-type double patenting as allegedly being obvious over claims 1-20 of U.S. Patent No. 10,554,475 and over claims 1-20 of U.S. Patent No. 10,558,798.[1] However, Applicant is submitting an electronic terminal disclaimer to disclaim claims 1-20 of U.S. Patent No. 10,554,475 and over claims 1-20 of U.S. Patent No. 10,558,798. So, Applicant requests that the double patenting rejection of claims 1-17, 19, and 20 be withdrawn.

[1] Office Action dated 03/23/2021 at pages 3-5.

IBM - Ex. 1002-Part 10 of 10, Page 3494
IPR2025-00379 (IBM v. Croga)

Double Patenting Rejections over co-pending Applications

Claims 1-20 are also provisionally rejected under the judicially created doctrine of obviousness-type double patenting as allegedly being obvious over claims of various co-pending applications.[2]

As discussed above, all pending claims are allowable over prior art and an electronic terminal disclaimer is being filed herewith for Patent No. 10,554,475 and U.S. Patent No. 10,558,798. Consequently, the only rejection remaining in this application is the double patenting rejections over following twelve co-pending applications:

1. 16/141,535 – earliest filing date is 09/28/2017
2. 16/144,609 – earliest filing date 09/28/2017
3. 16/144,761 – earliest filing date 09/28/2017
4. 16/146,149 – earliest filing date 09/28/2017
5. 16/142,810 – earliest filing date 09/28/2017
6. 16/146,490 – earliest filing date 09/28/2017
7. 16/163,796 – earliest filing date 10/23/2017
8. 16/163,884 – earliest filing date 10/23/2017
9. 16/164,342 – earliest filing date 10/23/2017
10. 16/059,381
11. 16/109,275
12. 16/138,696

First, the double patenting rejections that are based on co-pending applicants 1-9 should be withdrawn under § MPEP 804. MPEP § 804 states that "[i]f a provisional nonstatutory double patenting rejection is the only rejection remaining in an application having the earliest effective U.S. filing date … compared to the reference application(s), the examiner should withdraw the rejection in the application having the earliest effective U.S. filing date and permit that application to issue as a patent…." The present application was filed on September 25, 2018 and

[2] *Id.*

IBM - Ex. 1002-Part 10 of 10, Page 3495
IPR2025-00379 (IBM v. Croga)

claims priority to U.S. Provisional Application No. 62/563,237, filed September 26, 2017. Consequently, co-pending applications 1-9 have later U.S. filing date than the present application. So, the double patenting rejections that are based on co-pending applicants 1-9 should be withdrawn under § MPEP 804.

Second, the double patenting rejections that are based on co-pending applications 10-12 should be withdrawn because the Office Action has not demonstrated how each element of claims 1-20 may be found in co-pending applications 10-12. Specifically, MPEP § 804 provides that for an anticipation analysis, "the examiner typically should explain why it would have been obvious to select the claimed species or sub-genus given the genus claimed in the potentially conflicting patent or application." However, in the present case, no such explanation was provided. Rather, the Office Action merely stated that "[a]lthough the claims at issue are not identical, they are not patentably distinct from each other because each element of the claims of the present application is anticipated by the claims of cited applicants and as such are unpatentable for obviousness-type double patenting...."[3] Consequently, Applicant requests that the double patenting rejections based on co-pending applications 10-12 be withdrawn.

For at least the reasons above, Applicant requests reconsideration and withdrawal of the double patenting rejections.

[3] Office Action dated 03/23/2021 at p. 4-5.

IBM - Ex. 1002-Part 10 of 10, Page 3496
IPR2025-00379 (IBM v. Croga)

CONCLUSION

Applicant submits that the present application is in condition for allowance and a Notice of Allowance is requested.

Date: September 20, 2021

/Jeung Huh/
Jeung (Jacob) Huh
Registration No. 76,913

Condo Roccia Koptiw LLP
1800 JFK Blvd., Suite 1700
Philadelphia, PA 19103
Telephone: (215) 558-5740
Facsimile: (215) 558-5676

IBM - Ex. 1002-Part 10 of 10, Page 3497
IPR2025-00379 (IBM v. Croga)

Electronic Acknowledgement Receipt

EFS ID:	43811264
Application Number:	16141378
International Application Number:	
Confirmation Number:	2440
Title of Invention:	NETWORK ISOLATION WITH CLOUD NETWORKS
First Named Inventor/Applicant Name:	Glenn Coleman
Customer Number:	158266
Filer:	Jeung Huh/melissa overline
Filer Authorized By:	Jeung Huh
Attorney Docket Number:	LCOM_ENT_CLOUD_US01
Receipt Date:	20-SEP-2021
Filing Date:	25-SEP-2018
Time Stamp:	16:02:39
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		LCOM_ENT_CLOUD_US01_AS_FILED_Reply_to_Non_Compliance_Dated_09_20_21.pdf	187095 2a34950591b75b6d1b287beb5d4bec8e64e0dba5	yes	10

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Amendment/Req. Reconsideration-After Non-Final Reject	1	1
	Claims	2	6
	Applicant Arguments/Remarks Made in an Amendment	7	10
Warnings:			
Information:			
	Total Files Size (in bytes):	187095	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (09-11)
Approved for use through 1/31/2014. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875	Application or Docket Number 16/141,378	Filing Date 09/25/2018	☐ To be Mailed

ENTITY: ☑ LARGE ☐ SMALL ☐ MICRO

APPLICATION AS FILED - PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA		RATE ($)	FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A		N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A		N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A		N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*		x $100 =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*		x $460 =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $310 ($155 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).				
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))					
* If the difference in column 1 is less than zero, enter "0" in column 2.				TOTAL	

APPLICATION AS AMENDED - PART II

		(Column 1)		(Column 2)	(Column 3)			
AMENDMENT	09/20/2021	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	* 19	Minus	** 20	= 0		x $100 =	0
	Independent (37 CFR 1.16(h))	* 3	Minus	*** 3	= 0		x $480 =	0
	☐ Application Size Fee (37 CFR 1.16(s))							
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							
							TOTAL ADD'L FEE	0

		(Column 1)		(Column 2)	(Column 3)			
AMENDMENT		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=		x $0 =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=		x $0 =	
	☐ Application Size Fee (37 CFR 1.16(s))							
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							
							TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.	LIE
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".	/NICOLE L NICHOLSON/
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".	
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.	

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

158266 7590 10/05/2021

Condo Roccia Koptiw LLP
1800 JFK Boulevard
Suite 1700
Philadelphia, PA 19103

EXAMINER	
KORSAK, OLEG	
ART UNIT	PAPER NUMBER
2492	

DATE MAILED: 10/05/2021

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
16/141,378	09/25/2018	Glenn Coleman	LCOM_ENT_CLOUD_US01	2440

TITLE OF INVENTION: NETWORK ISOLATION WITH CLOUD NETWORKS

APPLN. TYPE	ENTITY STATUS	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	UNDISCOUNTED	$1200	$0.00	$0.00	$1200	01/05/2022

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the ENTITY STATUS shown above. If the ENTITY STATUS is shown as SMALL or MICRO, verify whether entitlement to that entity status still applies.

If the ENTITY STATUS is the same as shown above, pay the TOTAL FEE(S) DUE shown above.

If the ENTITY STATUS is changed from that shown above, on PART B - FEE(S) TRANSMITTAL, complete section number 5 titled "Change in Entity Status (from status indicated above)".

For purposes of this notice, small entity fees are 1/2 the amount of undiscounted fees, and micro entity fees are 1/2 the amount of small entity fees.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Maintenance fees are due in utility patents issuing on applications filed on or after Dec. 12, 1980. It is patentee's responsibility to ensure timely payment of maintenance fees when due. More information is available at www.uspto.gov/PatentMaintenanceFees.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), by mail or fax, or via EFS-Web.

By mail, send to: Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

By fax, send to: (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

158266 7590 10/05/2021

Condo Roccia Koptiw LLP
1800 JFK Boulevard
Suite 1700
Philadelphia, PA 19103

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being transmitted to the USPTO via EFS-Web or by facsimile to (571) 273-2885, on the date below.

(Typed or printed name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
16/141,378	09/25/2018	Glenn Coleman	LCOM_ENT_CLOUD_US01	2440

TITLE OF INVENTION: NETWORK ISOLATION WITH CLOUD NETWORKS

APPLN. TYPE	ENTITY STATUS	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	UNDISCOUNTED	$1200	$0.00	$0.00	$1200	01/05/2022

EXAMINER	ART UNIT	CLASS-SUBCLASS
KORSAK, OLEG	2492	726-012000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-09 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) The names of up to 3 registered patent attorneys or agents OR, alternatively,
(2) The name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 _____
2 _____
3 _____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document must have been previously recorded, or filed for recordation, as set forth in 37 CFR 3.11 and 37 CFR 3.81(a). Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) : ❑ Individual ❑ Corporation or other private group entity ❑ Government

4a. Fees submitted: ❑ Issue Fee ❑ Publication Fee (if required) ❑ Advance Order - # of Copies _____

4b. Method of Payment: *(Please first reapply any previously paid fee shown above)*

❑ Electronic Payment via EFS-Web ❑ Enclosed check ❑ Non-electronic payment by credit card (Attach form PTO-2038)

❑ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment to Deposit Account No. _____

5. **Change in Entity Status** (from status indicated above)

❑ Applicant certifying micro entity status. See 37 CFR 1.29

❑ Applicant asserting small entity status. See 37 CFR 1.27

❑ Applicant changing to regular undiscounted fee status.

NOTE: Absent a valid certification of Micro Entity Status (see forms PTO/SB/15A and 15B), issue fee payment in the micro entity amount will not be accepted at the risk of application abandonment.
NOTE: If the application was previously under micro entity status, checking this box will be taken to be a notification of loss of entitlement to micro entity status.
NOTE: Checking this box will be taken to be a notification of loss of entitlement to small or micro entity status, as applicable.

NOTE: This form must be signed in accordance with 37 CFR 1.31 and 1.33. See 37 CFR 1.4 for signature requirements and certifications.

Authorized Signature _____ Date _____

Typed or printed name _____ Registration No. _____

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
16/141,378	09/25/2018	Glenn Coleman	LCOM_ENT_CLOUD_US01	2440

158266 7590 10/05/2021

Condo Roccia Koptiw LLP
1800 JFK Boulevard
Suite 1700
Philadelphia, PA 19103

EXAMINER	
KORSAK, OLEG	
ART UNIT	PAPER NUMBER
2492	

DATE MAILED: 10/05/2021

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(Applications filed on or after May 29, 2000)

The Office has discontinued providing a Patent Term Adjustment (PTA) calculation with the Notice of Allowance.

Section 1(h)(2) of the AIA Technical Corrections Act amended 35 U.S.C. 154(b)(3)(B)(i) to eliminate the requirement that the Office provide a patent term adjustment determination with the notice of allowance. See Revisions to Patent Term Adjustment, 78 Fed. Reg. 19416, 19417 (Apr. 1, 2013). Therefore, the Office is no longer providing an initial patent term adjustment determination with the notice of allowance. The Office will continue to provide a patent term adjustment determination with the Issue Notification Letter that is mailed to applicant approximately three weeks prior to the issue date of the patent, and will include the patent term adjustment on the patent. Any request for reconsideration of the patent term adjustment determination (or reinstatement of patent term adjustment) should follow the process outlined in 37 CFR 1.705.

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

OMB Clearance and PRA Burden Statement for PTOL-85 Part B

The Paperwork Reduction Act (PRA) of 1995 requires Federal agencies to obtain Office of Management and Budget approval before requesting most types of information from the public. When OMB approves an agency request to collect information from the public, OMB (i) provides a valid OMB Control Number and expiration date for the agency to display on the instrument that will be used to collect the information and (ii) requires the agency to inform the public about the OMB Control Number's legal significance in accordance with 5 CFR 1320.5(b).

The information collected by PTOL-85 Part B is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 30 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450. Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

Notice of Allowability	Application No. 16/141,378	Applicant(s) Coleman et al.	
	Examiner OLEG KORSAK	Art Unit 2492	AIA (FITF) Status Yes

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☑ This communication is responsive to 09/20/2021.
   ☐ A declaration(s)/affidavit(s) under **37 CFR 1.130(b)** was/were filed on _____.

2. ☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

3. ☑ The allowed claim(s) is/are 1-17 and 19-20 . As a result of the allowed claim(s), you may be eligible to benefit from the **Patent Prosecution Highway** program at a participating intellectual property office for the corresponding application. For more information , please see **http://www.uspto.gov/patents/init_events/pph/index.jsp** or send an inquiry to **PPHfeedback@uspto.gov.**

4. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

   Certified copies:

   a) ☐ All b) ☐ Some* c) ☐ None of the:

   1. ☐ Certified copies of the priority documents have been received.
   2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
   3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____ .

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____ .

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☐ Notice of References Cited (PTO-892)
2. ☑ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date 09/14/2021.
3. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material _____.
4. ☐ Interview Summary (PTO-413), Paper No./Mail Date. _____.
5. ☐ Examiner's Amendment/Comment
6. ☑ Examiner's Statement of Reasons for Allowance
7. ☐ Other _____.

/OLEG KORSAK/
Primary Examiner, Art Unit 2492

DETAILED ACTION

1. This action is in response to amendment filed September 20, 2021.

2. By this amendment, claims 1-17 and 19-20 are presented for further consideration.

3. New IDS have been submitted on 09/14/2021 and was considered and signed by Examiner.

Notice of Pre-AIA or AIA Status

The present application, filed on or after March 16, 2013, is being examined under the first inventor to file provisions of the AIA.

Terminal Disclaimer

The terminal disclaimer filed on September 20, 2021 disclaiming the terminal portion of any patent granted on this application which would extend beyond the expiration date of US Patent No 10,558,798 and US Patent No 10,554,475 has been reviewed and is accepted. The terminal disclaimer has been recorded.

Response to Arguments

4. Applicant's arguments regarding the amendment submitted on September 20, 2021 have been considered and are persuasive. Also, please see NFOAM (03/21/2021, Claims 1-17 and 19-20 were indicated as allowable over prior art, and applicant filed a terminal disclaimer to overcome DP over of US Patent No 10,558,798 and US Patent No 10,554,475).
Further the closest relevant reference to currently claimed invention provided in IDS (09/14/2021) is not a prior art under 35 U.S.C. 102(b)(1). Therefore, all the previous pending rejections are withdrawn and the record is clear.

According to MPEP 1302.14 (I): "In most cases, the examiner's actions and the applicant's replies make evident the reasons for allowance, satisfying the "record as a whole"

proviso of the rule. This is particularly true when applicant fully complies with 37 CFR 1.111 (b) and (c) and 37 CFR 1.133(b). Thus, where the examiner's actions clearly point out the reasons for rejection and the applicant's reply explicitly presents reasons why claims are patentable over the reference, the reasons for allowance are in all probability evident from the record and no statement should be necessary."

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

Allowable Subject Matter

5. Claims 1-17 and 19-20 are allowed.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to OLEG KORSAK whose telephone number is (571)270-1938. The examiner can normally be reached on Monday-Friday 7:30am - 5:00pm EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Saleh Najjar can be reached on (571)272-4006. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the

Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/OLEG KORSAK/

Primary Examiner, Art Unit 2492

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	16/141,378	Coleman et al.
	Examiner	**Art Unit**
	OLEG KORSAK	2492

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

CLAIMS

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☑ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	05/21/2020	09/07/2020	03/22/2021	09/25/2021					
1	1	✓	✓	✓	=					
2	2	✓	✓	✓	=					
3	3	✓	✓	✓	=					
4	4	✓	✓	✓	=					
5	5	✓	✓	✓	=					
6	6	✓	✓	✓	=					
7	7	✓	✓	✓	=					
8	8	✓	✓	✓	=					
9	9	✓	✓	✓	=					
10	10	✓	✓	✓	=					
11	11	✓	✓	✓	=					
12	12	✓	✓	✓	=					
13	13	✓	✓	✓	=					
14	14	✓	✓	✓	=					
15	15	✓	✓	✓	=					
16	16	✓	✓	✓	=					
17	17	✓	✓	✓	=					
	18	✓	-	-	-					
18	19	✓	✓	✓	=					
19	20	✓	✓	✓	=					

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	16/141,378	Coleman et al.
	Examiner	Art Unit
	OLEG KORSAK	2492

CPC - Searched*		
Symbol	**Date**	**Examiner**
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	05/19/2020	OK
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	09/07/2020	OK
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	03/22/2021	OK
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	09/25/2021	OK

CPC Combination Sets - Searched*		
Symbol	**Date**	**Examiner**

US Classification - Searched*			
Class	**Subclass**	**Date**	**Examiner**

* See search history printout included with this form or the SEARCH NOTES box below to determine the scope of the search.

/OLEG KORSAK/ Primary Examiner, Art Unit 2492	

IBM - Ex. 1002-Part 10 of 10, Page 3516
IPR2025-00379 (IBM v. Croga)

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	16/141,378	Coleman et al.
	Examiner	Art Unit
	OLEG KORSAK	2492

Search Notes		
Search Notes	**Date**	**Examiner**
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	05/19/2020	OK
EAST search limited by keywords	05/19/2020	OK
GOOGLE NPL search	05/20/2020	OK
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	09/07/2020	OK
Updated EAST search limited by keywords	09/07/2020	OK
GOOGLE NPL search	09/07/2020	OK
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	03/22/2021	OK
Updated EAST search limited by keywords	03/22/2021	OK
H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10	09/25/2021	OK
EAST search limited by keywords	09/25/2021	OK

Interference Search			
US Class/CPC Symbol	**US Subclass/CPC Group**	**Date**	**Examiner**
H04L63	0209 0236 0281 029 145 20	09/25/2021	OK
G06F9	45558	09/25/2021	OK
G06F21	53	09/25/2021	OK
H04L67	10	09/25/2021	OK

/OLEG KORSAK/ Primary Examiner, Art Unit 2492	

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	16/141,378	Coleman et al.
	Examiner	**Art Unit**
	OLEG KORSAK	2492

CPC								
Symbol							**Type**	**Version**
H04L	/	63	/	0209			F	2013-01-01
H04L	/	63	/	029			I	2013-01-01
H04L	/	63	/	0281			I	2013-01-01
G06F	/	21	/	53			I	2013-01-01
G06F	/	9	/	45558			I	2013-01-01
H04L	/	63	/	20			I	2013-01-01
H04L	/	67	/	10			I	2013-01-01
H04L	/	63	/	145			I	2013-01-01
H04L	/	63	/	0236			I	2013-01-01
G06F	/	2009	/	45587			A	2013-01-01
G06F	/	2221	/	2149			A	2013-01-01

CPC Combination Sets								
Symbol					**Type**	**Set**	**Ranking**	**Version**
	/		/					

NONE		**Total Claims Allowed:**	
(Assistant Examiner)	(Date)	19	
/OLEG KORSAK/ Primary Examiner, Art Unit 2492	25 September 2021	O.G. Print Claim(s)	O.G. Print Figure
(Primary Examiner)	(Date)	1	1

U.S. Patent and Trademark Office

Part of Paper No.: 20210925

IBM - Ex. 1002-Part 10 of 10, Page 3512
IPR2025-00379 (IBM v. Croga)

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	16/141,378	Coleman et al.
	Examiner	**Art Unit**
	OLEG KORSAK	2492

INTERNATIONAL CLASSIFICATION					
CLAIMED					
H04L29/06	/	29	/	06	

NON-CLAIMED				
	/		/	

US ORIGINAL CLASSIFICATION	
CLASS	**SUBCLASS**

CROSS REFERENCES(S)						
CLASS	**SUBCLASS (ONE SUBCLASS PER BLOCK)**					

NONE		**Total Claims Allowed:**	
(Assistant Examiner)	(Date)	19	
/OLEG KORSAK/ Primary Examiner, Art Unit 2492	25 September 2021	O.G. Print Claim(s)	O.G. Print Figure
(Primary Examiner)	(Date)	1	1

U.S. Patent and Trademark Office

Part of Paper No.: 20210925

IBM - Ex. 1002-Part 10 of 10, Page 3513
IPR2025-00379 (IBM v. Croga)

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	16/141,378	Coleman et al.
	Examiner	**Art Unit**
	OLEG KORSAK	2492

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☑ T.D. ☐ R.1.47

CLAIMS

Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original
1	1	10	10	18	19										
2	2	11	11	19	20										
3	3	12	12												
4	4	13	13												
5	5	14	14												
6	6	15	15												
7	7	16	16												
8	8	17	17												
9	9		18												

NONE		**Total Claims Allowed:**	
(Assistant Examiner)	(Date)	19	
/OLEG KORSAK/ Primary Examiner, Art Unit 2492	25 September 2021	O.G. Print Claim(s)	O.G. Print Figure
(Primary Examiner)	(Date)	1	1

U.S. Patent and Trademark Office

Part of Paper No.: 20210925

IBM - Ex. 1002-Part 10 of 10, Page 3514
IPR2025-00379 (IBM v. Croga)

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	90	("10037199" | "10055231" | "10122703" | "10397273" | "10521612" | "10931669" | "20020023212" | "20020040439" | "20020069366" | "20050144467" | "20060282795" | "20070136579" | "20080016313" | "20090055924" | "20090164994" | "20090249472" | "20090265760" | "20110154431" | "20110299515" | "20120017213" | "20120265976" | "20130132948" | "20130145463" | "20140040979" | "20140059642" | "20140075535" | "20140108558" | "20140115646" | "20140282890" | "20140344912" | "20140351516" | "20150281176" | "20160057167" | "20160134549" | "20160226834" | "20160246974" | "20160248754" | "20160292694" | "20170076092" | "20170142144" | "20170180413" | "20170206351" | "20170208067" | "20170250997" | "20170317978" | "20170353496" | "20170359309" | "20180027006" | "20180115519" | "20180139178" | "20180196945" | "20180198824" | "20180203995" | "20180234422" | "20180276396" | "20180330257" | "20180367528" | "20190097970" | "20190097972" | "20190098007" | "20190098020" | "20190121961" | "20190121962" | "20190213325" | "7150040" | "7467408" | "7694328" | "7716720" | "7870153" | "7926086" | "8090797" | "8539561" | "8566398" | "8613070" | "8769268" | "8931054" | "8966464" | "9294492" | "9386021" | "9438634" | "9465734" | "9531715" | "9602524" | "9609026" | "9680873" | "9729579" | "9787639" | "9864600" | "9921860" | "9942198").PN.	US-PGPUB; USPAT	OR	OFF	2021/09/25 11:31
L2	53	(internal NEAR2 firewall$2) AND (protected NEAR2 memory)	US-PGPUB; USPAT	OR	OFF	2021/09/25 11:32
L3	1	L1 AND L2	US-PGPUB; USPAT	OR	OFF	2021/09/25 11:32

L4	1	L3 AND sandbox$3	US-PGPUB; USPAT	OR	OFF	2021/09/25 11:33
L5	1	L4 AND proxy	US-PGPUB; USPAT	OR	OFF	2021/09/25 11:33
L7	1	L5 AND cloud	US-PGPUB; USPAT	OR	OFF	2021/09/25 11:34
L8	1	L7 AND isolat$3	US-PGPUB; USPAT	OR	OFF	2021/09/25 11:34
L9	67	((("L3") near3 ("TECHNOLOGIES") near3 ("INC"))).AS,AANM.	USPAT	OR	OFF	2021/09/25 12:19
L10	46	((("COLEMAN") near3 ("Glenn")) OR (("MARTZ") near3 ("Peter")) OR (("MORITZ") near3 ("Kenneth"))).INV.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	OFF	2021/09/25 12:19
L11	17	("20020069369" | "20060070066" | "20070220187" | "20070260873" | "20080028401" | "20080201711" | "20080256536" | "20090172781" | "20090328038" | "20100138829" | "20100138830" | "20100223613" | "20100251329" | "20120023593" | "7478330" | "7698442" | "7805516").PN.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L12	100,354	(H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10).CPC.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L13	2,873	(internal NEAR2 firewall$2)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L14	6	L9 AND L13	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L15	17	L10 AND L13	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L16	0	L11 AND L13	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L17	1,006	L12 AND L13	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L18	1	L11 AND (protected NEAR2 memory)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L19	67	((("L3") near3 ("TECHNOLOGIES") near3 ("INC"))).AS,AANM.	USPAT	OR	OFF	2021/09/25 12:19
L20	46	((("COLEMAN") near3 ("Glenn")) OR (("MARTZ") near3 ("Peter")) OR (("MORITZ") near3 ("Kenneth"))).INV.	US-PGPUB; USPAT; USOCR;	OR	OFF	2021/09/25 12:19

			EPO; JPO; DERWENT			
L21	17	("20020069369" | "20060070066" | "20070220187" | "20070260873" | "20080028401" | "20080201711" | "20080256536" | "20090172781" | "20090328038" | "20100138829" | "20100138830" | "20100223613" | "20100251329" | "20120023593" | "7478330" | "7698442" | "7805516").PN.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L22	100,354	(H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10).CPC.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L23	2,873	(internal NEAR2 firewall$2)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L24	6	L19 AND L23	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L25	17	L20 AND L23	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L26	0	L21 AND L23	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L27	1,006	L22 AND L23	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L28	1	L21 AND (protected NEAR2 memory)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L29	67	((("L3") near3 ("TECHNOLOGIES") near3 ("INC"))).AS,AANM.	USPAT	OR	OFF	2021/09/25 12:19
L30	46	((("COLEMAN") near3 ("Glenn")) OR (("MARTZ") near3 ("Peter")) OR (("MORITZ") near3 ("Kenneth"))).INV.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	OFF	2021/09/25 12:19
L31	17	("20020069369" | "20060070066" | "20070220187" | "20070260873" | "20080028401" | "20080201711" | "20080256536" | "20090172781" | "20090328038" | "20100138829" | "20100138830" | "20100223613" | "20100251329" | "20120023593" | "7478330" | "7698442" | "7805516").PN.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L32	100,354	(H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10).CPC.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19

L33	2,873	(internal NEAR2 firewall$2)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L34	6	L29 AND L33	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L35	17	L30 AND L33	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L36	0	L31 AND L33	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L37	1,006	L32 AND L33	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L38	1	L31 AND (protected NEAR2 memory)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L39	67	((("L3") near3 ("TECHNOLOGIES") near3 ("INC"))).AS,AANM.	USPAT	OR	OFF	2021/09/25 12:19
L40	46	((("COLEMAN") near3 ("Glenn")) OR (("MARTZ") near3 ("Peter")) OR (("MORITZ") near3 ("Kenneth"))).INV.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	OFF	2021/09/25 12:19
L41	17	("20020069369" | "20060070066" | "20070220187" | "20070260873" | "20080028401" | "20080201711" | "20080256536" | "20090172781" | "20090328038" | "20100138829" | "20100138830" | "20100223613" | "20100251329" | "20120023593" | "7478330" | "7698442" | "7805516").PN.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L42	100,354	(H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10).CPC.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L43	2,873	(internal NEAR2 firewall$2)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L44	6	L39 AND L43	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L45	17	L40 AND L43	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L46	0	L41 AND L43	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L47	1,006	L42 AND L43	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19
L48	1	L41 AND (protected NEAR2 memory)	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:19

L50	0	((internal NEAR2 firewall$2) AND (protected NEAR2 memory)AND sandbox$3).CLM.	US-PGPUB; USPAT	OR	OFF	2021/09/25 12:20

EAST Search History (Interference)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L49	44,897	(H04L63/0209 G06F9/45558 G06F21/53 H04L63/0236 H04L63/0281 H04L63/029 H04L63/145 H04L63/20 H04L67/10).CPC.	USPAT	OR	OFF	2021/09/25 12:19
L51	0	((internal NEAR2 firewall$2) AND (protected NEAR2 memory)AND sandbox$3).CLM.	USPAT	OR	OFF	2021/09/25 12:20
L52	7	((internal NEAR2 firewall$2) AND (protected NEAR2 memory)).CLM.	USPAT	OR	OFF	2021/09/25 12:20
L53	7	L49 AND L52	USPAT	OR	OFF	2021/09/25 12:21
L54	0	L53 AND (sandbox$3).CLM.	USPAT	OR	OFF	2021/09/25 12:22

9/25/2021 12:22:42 PM
C:\Users\okorsak\Documents\e-Red Folder\16\16141378 Coleman\16141378 Coleman.wsp

Substitute for 1449/PTO				Application Number	16/141,378
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
(use as many sheets as necessary)				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	1	of	12	Attorney Docket Number	LCOM_ENT_CLOUD_US01

U.S. PATENTS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	36	7150040	B2	2006-12-12	Brownell, David	
	37	7467408	B1	2008-12-16	O'Toole Jr., James W.	
	38	7694328	B2	2010-04-06	Joshi et al.	
	39	7716720	B1	2010-05-11	Marek et al.	
	40	7870153	B2	2011-01-11	Croft et al.	
	41	7926086	B1	2011-04-12	Violleau et al.	
	42	8090797	B2	2012-01-03	Chinta et al.	
	43	8539561	B2	2013-09-17	Gupta et al.	
	44	8566398	B2	2013-10-22	Barnier et al.	
	45	8613070	B1	2013-12-17	Borzycki et al.	
	46	8769268	B2	2014-07-01	Morozov et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO				Application Number	16/141,378
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
(use as many sheets as necessary)				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	2	of	12	Attorney Docket Number	LCOM_ENT_CLOUD_US01

U.S. PATENTS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	47	8931054	B2	2015-01-06	Huynh et al.	
	48	8966464	B1	2015-02-24	Christopher et al.	
	49	9294492	B1	2016-03-22	Martini et al.	
	50	9386021	B1	2016-07-05	Pratt, Ian	
	51	9438634	B1	2016-09-06	Ross et al.	
	52	9465734	B1	2016-10-11	Myrick et al.	
	53	9531715	B1	2016-12-27	Rodgers et al.	
	54	9602524	B2	2017-03-21	Ghosh et al.	
	55	9609026	B2	2017-03-28	Ross et al.	
	56	9680873	B1	2017-06-13	Halls et al.	
	57	9729579	B1	2017-08-08	Marino et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	3	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENTS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	58	9787639	B1	2017-10-10	Sun et al.	
	59	9864600	B2	2018-01-09	Larimore et al.	
	60	9921860	B1	2018-03-20	Banga et al.	
	61	9942198	B2	2018-04-10	Hoy et al.	
	62	10037199	B2	2018-07-31	Hung et al.	
	63	10055231	B1	2018-08-21	Li et al.	
	64	10122703	B2	2018-11-06	Innes et al.	
	65	10397273	B1	2019-08-27	Stickle et al.	
	66	10521612	B2	2019-12-31	Arad et al.	
	67	10931669	B2	2021-02-23	Coleman et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO				Application Number	16/141,378
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
(use as many sheets as necessary)				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	4	of	12	Attorney Docket Number	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	68	20020023212	A1	2002-02-21	Proudler, Graeme John	
	69	20020040439	A1	2002-04-04	Kellum, Charles W.	
	70	20020069366	A1	2002-06-06	Schoettger, Chad	
	71	20050144467	A1	2005-06-30	Yamazaki, Takeshi	
	72	20060282795	A1	2006-12-14	Clark et al.	
	73	20070136579	A1	2007-06-14	Levy et al.	
	74	20080016313	A1	2008-01-17	Murotake et al.	
	75	20090055924	A1	2009-02-26	Trotter, Douglas H.	
	76	20090164994	A1	2009-06-25	Vasilevsky et al.	
	77	20090249472	A1	2009-10-01	Litvin et al.	
	78	20090265760	A1	10-22-2009	Zhu, et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	5	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	79	20110154431	A1	2011-06-23	Walsh, Daniel J.	
	80	20110299515	A1	2011-12-08	Robertson et al.	
	81	20120017213	A1	2012-01-19	Hunt et al.	
	82	20120265976	A1	2012-10-18	Spiers et al.	
	83	20130132948	A1	05-23-2013	Hari, et al.	
	84	20130145463	A1	2013-06-06	Ghosh et al.	
	85	20140040979	A1	2014-02-06	Barton et al.	
	86	20140059642	A1	2014-02-27	Deasy et al.	
	87	20140075535	A1	2014-03-13	Soffer et al.	
	88	20140108558	A1	2014-04-17	Borzycki et al.	
	89	20140115646	A1	2014-04-24	Rajgopal et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	6	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	90	20140282890	A1	2014-09-18	Li et al.	
	91	20140344912	A1	2014-11-20	Chapman et al.	
	92	20140351516	A1	2014-11-27	Larimore et al.	
	93	20150281176	A1	2015-10-01	Banfield, Bret	
	94	20160057167	A1	2016-02-25	Bach, Timothy	
	95	20160134549	A1	2016-05-12	Abel et al.	
	96	20160226834	A1	2016-08-04	Dawson, Martin	
	97	20160246974	A1	2016-08-25	Broz et al.	
	98	20160248754	A1	2016-08-25	Jahr, Jeffrey Stephen	
	99	20160292694	A1	2016-10-06	Goldschlag et al.	
	100	20170076092	A1	2017-03-16	Kashyap et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	7	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	101	20170142144	A1	2017-05-18	Weinberger et al.	
	102	20170180413	A1	2017-06-22	Petry et al.	
	103	20170206351	A1	2017-07-20	Jay et al.	
	104	20170208067	A1	2017-07-20	Sriramakrishnan, Sarma	
	105	20170250997	A1	2017-08-31	Rostamabadi et al.	
	106	20170317978	A1	2017-11-02	Diaz-Cuellar et al.	
	107	20170353496	A1	2017-12-07	Pai et al.	
	108	20170359309	A1	2017-12-14	Bolte et al.	
	109	20180027006	A1	2018-01-25	Zimmermann et al.	
	110	20180115519	A1	2018-04-26	Bonomi et al.	
	111	20180139178	A1	2018-05-17	Gan, Yongcun	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO				Application Number	16/141,378
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
(use as many sheets as necessary)				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	8	of	12	Attorney Docket Number	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	112	20180196945	A1	2018-07-12	Kornegay et al.	
	113	20180198824	A1	2018-07-12	Pulapaka et al.	
	114	20180203995	A1	2018-07-19	Yuen et al.	
	115	20180234422	A1	2018-08-16	Odom et al.	
	116	20180276396	A1	2018-09-27	Yablokov et al.	
	117	20180330257	A1	2018-11-15	Dodson et al.	
	118	20180367528	A1	2018-12-20	Schwarz et al.	
	119	20190097970	A1	2019-03-28	Coleman et al.	
	120	20190097972	A1	2019-03-28	Coleman et al.	
	121	20190098007	A1	2019-03-28	Coleman et al.	
	122	20190098020	A1	2019-03-28	Martz et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	9	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

U.S. PATENT APPLICATION PUBLICATIONS

Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	123	20190121961	A1	2019-04-25	Coleman et al.	
	124	20190121962	A1	2019-04-25	Coleman et al.	
	125	20190213325	A1	2019-07-11	McKerchar et al.	

FOREIGN PATENT DOCUMENTS

Examiner Initials*	Cite No	Foreign Document Number[3]	Country Code[2]	Kind Code[4]	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines where Relevant Passages or Relevant Figures Appear	T

NON-PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T
	126	"NPL Search Term", Web Search History, 1 page	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	10	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

NON-PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T
	127	AZAB et al., "SICE: A Hardware-Level Strongly Isolated Computing Environment for x86 Multi-Core Platforms", Proceedings of the 18th ACM Conference on Computer and Communications Security, October 2011, pp. 375-388	
	128	BRUCKMAN et al., ""Georgia Computes!": Improving the Computing Education Pipeline", Proceeding of the 40th ACM Technical Symposium on Computer Science Education, 2009, pp. 86-90	
	129	GUPTA et al., "A Secure Architecture Design Based on Application Isolation, Code Minimization and Randomization", IEEE 6th Symposium on Security Analytics and Automation 2013, 2013, pp. 423-429	
	130	HAQ et al., "Design and Implementation of Sandbox Technique for Isolated Applications", IEEE School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China, 2016, pp. 557-561	
	131	HSU et al., "Enforcing Least Privilege Memory Views for Multithreaded Applications", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, October 2016, pp. 393-405	
	132	JAUERNIG et al., "Trusted Execution Environments: Properties, Applications, and Challenges", IEEE Computer and Reliability Societies, Resilient Security, March-April 2020, pp. 56-60	
	133	JUNG et al., "An Architecture for Virtualization-Based Trusted Execution Environment on Mobile Devices", 2014 IEEE 11th International Conference on Ubiquitous Intelligence and Computing and 2014 IEEE 11th International Conference on Autonomic and Trusted Computing and 2014 IEEE 14th International Conference on Scalable Computing and Communications and, Associated Symposia/Workshops, 2014, pp. 540-547	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	11	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

NON-PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T
	134	KOSTIAINEN et al., "Dedicated Security Chips in the Age of Secure Enclaves", IEEE Computer and Reliability Societies; Hardware-Assisted Security, September-October 2020, pp. 38-46	
	135	LE et al., "Computing the Real Isolated Points of an Algebraic Hypersurface", International Symposium on Symbolic and Algebraic Computation, August 24, 2020, 12 pages	
	136	LIU et al., "Thwarting Memory Disclosure with Efficient Hypervisor-Enforced Intra-Domain Isolation", Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, October 2015, pp. 1607-1619	
	137	SRIVASTAVA et al., "Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections", School of Computer Science, Georgia Institute of Technology, 2008, pp. 39-58	
	138	YADAV et al., "Impact & Analysis of Virtual Workspace on Grid Computing", International Journal of Science & Technology (IJST), Vol. 2, Issue 2, April 2012, pp. 24-32	
	139	YEE et al., "Native Client: A Sandbox for Portable, Untrusted x86 Native Code", 2009 30th IEEE Symposium on Security and Privacy, 2009, pp. 79-93	

EXAMINER SIGNATURE			
Examiner Signature	/OLEG KORSAK/	Date Considered	09/25/2021

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

Substitute for 1449/PTO **INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(use as many sheets as necessary)*				**Application Number**	16/141,378
				Filing Date	Sep 25, 2018
				First Named Inventor	Glenn Coleman
				Art Unit	2492
				Examiner Name	Korsak, Oleg
Sheet	12	of	12	**Attorney Docket Number**	LCOM_ENT_CLOUD_US01

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /O.K/

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), by mail or fax, or via EFS-Web.

By mail, send to: Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

By fax, send to: (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

158266 7590 10/05/2021

Condo Roccia Koptiw LLP
1800 JFK Boulevard
Suite 1700
Philadelphia, PA 19103

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being transmitted to the USPTO via EFS-Web or by facsimile to (571) 273-2885, on the date below.

(Typed or printed name)

(Signature)

(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
16/141,378	09/25/2018	Glenn Coleman	LCOM_ENT_CLOUD_US01	2440

TITLE OF INVENTION: NETWORK ISOLATION WITH CLOUD NETWORKS

APPLN. TYPE	ENTITY STATUS	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	UNDISCOUNTED	$1200	$0.00	$0.00	$1200	01/05/2022

EXAMINER	ART UNIT	CLASS-SUBCLASS
KORSAK, OLEG	2492	726-012000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-09 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) The names of up to 3 registered patent attorneys or agents OR, alternatively,
(2) The name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 Condo Roccia Koptiw LLP
2
3

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document must have been previously recorded, or filed for recordation, as set forth in 37 CFR 3.11 and 37 CFR 3.81(a). Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: L3 Technologies, Inc.

(B) RESIDENCE: (CITY and STATE OR COUNTRY): New York, New York

Please check the appropriate assignee category or categories (will not be printed on the patent) : ☐ Individual ☒ Corporation or other private group entity ☐ Government

4a. Fees submitted: ☒ Issue Fee ☐ Publication Fee (if required) ☐ Advance Order - # of Copies _____

4b. Method of Payment: *(Please first reapply any previously paid fee shown above)*

☐ Electronic Payment via EFS-Web ☐ Enclosed check ☐ Non-electronic payment by credit card (Attach form PTO-2038)

☒ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment to Deposit Account No. 50-5519

5. **Change in Entity Status** (from status indicated above)

☐ Applicant certifying micro entity status. See 37 CFR 1.29

☐ Applicant asserting small entity status. See 37 CFR 1.27

☐ Applicant changing to regular undiscounted fee status.

NOTE: Absent a valid certification of Micro Entity Status (see forms PTO/SB/15A and 15B), issue fee payment in the micro entity amount will not be accepted at the risk of application abandonment.
NOTE: If the application was previously under micro entity status, checking this box will be taken to be a notification of loss of entitlement to micro entity status.
NOTE: Checking this box will be taken to be a notification of loss of entitlement to small or micro entity status, as applicable.

NOTE: This form must be signed in accordance with 37 CFR 1.31 and 1.33. See 37 CFR 1.4 for signature requirements and certifications.

Authorized Signature /Jeung Huh/ Date October 11, 2021

Typed or printed name Jeung Huh Registration No. 76,913

Electronic Patent Application Fee Transmittal

Application Number:	16141378
Filing Date:	25-Sep-2018
Title of Invention:	NETWORK ISOLATION WITH CLOUD NETWORKS
First Named Inventor/Applicant Name:	Glenn Coleman
Filer:	Jeung Huh/Darleen Yacovone
Attorney Docket Number:	LCOM_ENT_CLOUD_US01

Filed as Large Entity

Filing Fees for Utility under 35 USC 111(a)

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
UTILITY APPL ISSUE FEE	1501	1	1200	1200

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Extension-of-Time:				
Miscellaneous:				
			Total in USD ($)	**1200**

Electronic Acknowledgement Receipt

EFS ID:	43996574
Application Number:	16141378
International Application Number:	
Confirmation Number:	2440
Title of Invention:	NETWORK ISOLATION WITH CLOUD NETWORKS
First Named Inventor/Applicant Name:	Glenn Coleman
Customer Number:	158266
Filer:	Jeung Huh/Darleen Yacovone
Filer Authorized By:	Jeung Huh
Attorney Docket Number:	LCOM_ENT_CLOUD_US01
Receipt Date:	11-OCT-2021
Filing Date:	25-SEP-2018
Time Stamp:	14:26:53
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	CARD
Payment was successfully received in RAM	$1200
RAM confirmation Number	E20210AE27102107
Deposit Account	
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Issue Fee Payment (PTO-85B)	LCOM_ENT_CLOUD_US01_Issue_Fee_Transmittal_NOA_Dated_10_05_2021_as_filed.pdf	1602479 785ba36f33cf52c0d5f6ac3a63a75e691e31b24f	no	1
Warnings:					
Information:					
2	Fee Worksheet (SB06)	fee-info.pdf	38098 eec0f08f44796716e31095abbe57a7794fb64add	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	1640577		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	ISSUE DATE	PATENT NO.	ATTORNEY DOCKET NO.	CONFIRMATION NO.
16/141,378	11/16/2021	11178104	LCOM_ENT_CLOUD_US01	2440

158266 7590 10/27/2021

Condo Roccia Koptiw LLP
1800 JFK Boulevard
Suite 1700
Philadelphia, PA 19103

ISSUE NOTIFICATION

The projected patent number and issue date are specified above.

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment is 0 day(s). Any patent to issue from the above-identified application will include an indication of the adjustment on the front page.

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Application Assistance Unit (AAU) of the Office of Patents Stakeholder Experience (OPSE), Stakeholder Support Division (SSD) at (571)-272-4200.

INVENTOR(s) (Please see PAIR WEB site http://pair.uspto.gov for additional inventors):

Glenn Coleman, Berwyn, PA;
Peter Martz, Marlton, NJ;
Kenneth Moritz, Lansdowne, PA;

APPLICANT(s) (Please see PAIR WEB site http://pair.uspto.gov for additional applicants):

L3 Technologies, Inc., New York, NY;

The United States represents the largest, most dynamic marketplace in the world and is an unparalleled location for business investment, innovation, and commercialization of new technologies. The USA offers tremendous resources and advantages for those who invest and manufacture goods here. Through SelectUSA, our nation works to encourage and facilitate business investment. To learn more about why the USA is the best country in the world to develop technology, manufacture products, and grow your business, visit SelectUSA.gov.

UNITED STATES
PATENT AND TRADEMARK OFFICE

P.O. Box 1450
Alexandria, VA 22313 - 1450
www.uspto.gov

ELECTRONIC ACKNOWLEDGEMENT RECEIPT

APPLICATION #	RECEIPT DATE / TIME	ATTORNEY DOCKET #
16/141,378	**05/17/2024 02:45:13 PM Z ET**	**LCOM_ENT_CLOUD_US01**

Title of Invention

NETWORK ISOLATION WITH CLOUD NETWORKS

Application Information

APPLICATION TYPE	Utility - Nonprovisional Application under 35 USC 111(a)	PATENT #	11178104
CONFIRMATION #	2440	FILED BY	Laura Alto
PATENT CENTER #	65574419	FILING DATE	09/25/2018
CUSTOMER #	158266	FIRST NAMED INVENTOR	Glenn Coleman
CORRESPONDENCE ADDRESS	-	AUTHORIZED BY	Antonio Papageorgiou

Documents

TOTAL DOCUMENTS: 2

DOCUMENT	PAGES	DESCRIPTION	SIZE (KB)
aia0096_template__2.pdf	3	Assignee showing of ownership per 37 CFR 3.73	118 KB
aia0081a_template__2.pdf	2	Power of Attorney	142 KB

Digest

DOCUMENT	MESSAGE DIGEST(SHA-512)
aia0096_template__2.pdf	50EDD448DA2BB55168DCAE0F362FC5D699B6E79D8AE87817 1EC4B2CEB0F112E37E3807A11FDAB3E1AAFC5D62CF35ADE2 D19C071D9342611919113 8B908060CFB

aia0081a_template__2.pdf E7579A6EAD8E84E924638D917249EB522BF42A9A0E1BD954776EACA00F7458F119A7FB9DD6BEDAC5C435E8A7D5A51AE98926B8F0AC8BEC7674B8D6046C969FC0

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/AIA/96 (08-12)
Approved for use through 01/31/2013. OMB 0651-0031
U.S. Patent and TrademarkOffice; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork ReductionAct of 1995, no personsare required to respond to a collection of information unless it displays a valid OMB control number.

STATEMENT UNDER 37 CFR 3.73(c)

Applicant/Patent Owner: CROGA INNOVATIONS LIMITED

Application No./Patent No.: 11,178,104 Filed/Issue Date: 11/16/2021

Titled: Network isolation with cloud networks

CROGA INNOVATIONS LIMITED, a Limited Liability Company

(Name of Assignee) (Type of Assignee, e.g., corporation, partnership, university, government agency, etc.)

states that, for the patent application/patent identified above, it is (choose **one** of options 1, 2, 3 or 4 below):

1. ☑ The assignee of the entire right, title, and interest.

2. ☐ An assignee of less than the entire right, title, and interest (check applicable box):

   ☐ The extent (by percentage) of its ownership interest is _____%. Additional Statement(s) by the owners holding the balance of the interest must be submitted to account for 100% of the ownership interest.

   ☐ There are unspecified percentages of ownership. The other parties, including inventors, who together own the entire right, title and interest are:

   Additional Statement(s) by the owner(s) holding the balance of the interest must be submitted to account for the entire right, title, and interest.

3. ☐ The assignee of an undivided interest in the entirety (a complete assignment from one of the joint inventors was made). The other parties, including inventors, who together own the entire right, title, and interest are:

   Additional Statement(s) by the owner(s) holding the balance of the interest must be submitted to account for the entire right, title, and interest.

4. ☐ The recipient, via a court proceeding or the like (*e.g.*, bankruptcy, probate), of an undivided interest in the entirety (a complete transfer of ownership interest was made). The certified document(s) showing the transfer is attached.

The interest identified in option 1, 2 or 3 above (not option 4) is evidenced by either (choose **one** of options A or B below):

A. ☐ An assignment from the inventor(s) of the patent application/patent identified above. The assignment was recorded in the United States Patent and Trademark Office at Reel _____, Frame _____, or for which a copy thereof is attached.

B. ☑ A chain of title from the inventor(s), of the patent application/patent identified above, to the current assignee as follows:

1. From: Inventors To: L3 Technologies, Inc.

   The document was recorded in the United States Patent and Trademark Office at

   Reel 057738, Frame 0143, or for which a copy thereof is attached.

2. From: L3 Technologies, Inc. To: Croga Innovations Limited

   The document was recorded in the United States Patent and Trademark Office at

   Reel 066001, Frame 0843, or for which a copy thereof is attached.

This collectionof information is required by 37CFR 3.73(b). The informationis required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application.Confidentiality isgoverned by 35U.S.C. 122 and37 CFR 1.11and 1.14.This collection is estimated to take 12 minutes to complete, including gathering,preparing, and submitting the completed applicationform to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should besent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETEDFORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA22313-1450** .

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

PTO/AIA/96 (08-12)
Approved for use through 01/31/2013. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

STATEMENT UNDER 37 CFR 3.73(c)

3. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at

Reel _____, Frame _____, or for which a copy thereof is attached.

4. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at

Reel _____, Frame _____, or for which a copy thereof is attached.

5. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at

Reel _____, Frame _____, or for which a copy thereof is attached.

6. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at

Reel _____, Frame _____, or for which a copy thereof is attached.

☐ Additional documents in the chain of title are listed on a supplemental sheet(s).

☑ As required by 37 CFR 3.73(c)(1)(i), the documentary evidence of the chain of title from the original owner to the assignee was, or concurrently is being, submitted for recordation pursuant to 37 CFR 3.11.

[NOTE: A separate copy (i.e., a true copy of the original assignment document(s)) must be submitted to Assignment Division in accordance with 37 CFR Part 3, to record the assignment in the records of the USPTO. See MPEP 302.08]

The undersigned (whose title is supplied below) is authorized to act on behalf of the assignee.

/Antonio Papageorgiou/	May 1, 2024
Signature	Date
Antonio Papageorgiou	53431
Printed or Typed Name	Title or Registration Number

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requiresthat you begiven certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirementsof the Act, please be advised that: (1)the general authority for the collectionof this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary;and (3) the principalpurpose for whichthe informationis used by the U.S. Patent and Trademark Office is to process and/or examine your submission related toa patent application or patent. If you do not furnishthe requested information, the U.S. Patent and Trademark Officemay not be able to process and/or examine yoursubmission, which may result in termination of proceedings orabandonment of the application or expiration of the patent.

Theinformation provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need forthe information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. Arecord related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further,a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. Arecord from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

PTO/AIA/81A (02-15)
Approved for use through 03/31/2021. OMB 0651-0035
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995 no persons are required to respond to a collection of information unless it displays a valid OMB control number

PATENT - POWER OF ATTORNEY OR REVOCATION OF POWER OF ATTORNEY WITH A NEW POWER OF ATTORNEY AND CHANGE OF CORRESPONDENCE ADDRESS

Patent Number	11,178,104
Issue Date	11/16/2021
First Named Inventor	Coleman, Glenn (Berwyn, PA, US)
Title	Network isolation with cloud networks
Attorney Docket No.	

I hereby revoke all previous powers of attorney given in the above-identified patent.

☐ A Power of Attorney is submitted herewith.

OR

☒ I hereby appoint Practitioner(s) associated with the Customer Number identified in the box at right as my/our attorney(s) or agent(s) with respect to the patent identified above, and to transact all business in the United States Patent and Trademark Office connected therewith: 199393

OR

☐ I hereby appoint Practitioner(s) named below as my/our attorney(s) or agent(s) with respect to the patent identified above, and to transact all business in the United States Patent and Trademark Office connected therewith:

Practitioner(s) Name	Registration Number

Please recognize or change the correspondence address for the above-identified patent to:

☒ The address associated with the above-identified Customer Number.

OR

☐ The address associated with the Customer Number identified in the box at right: 199393

OR

☐ Firm or Individual Name					
Address					
City		State		Zip	
Country					
Telephone		Email			

I am the:

☐ Applicant.

OR

☒ Patent owner.

Statement under 37 CFR 3.73(c) (Form PTO/AIA/96) submitted herewith or filed on _____.

SIGNATURE of Applicant or Patent Owner

Signature	/Ciaran O'Gara/	Date	May 1, 2024
Name	Ciaran O'Gara	Telephone	
Title and Company	Director CROGA INNOVATIONS LIMITED		

NOTE: Signatures of all the applicants or patent owners of the entire interest or their representative(s) are required. If more than one signature is required, submit multiple forms, check the box below, and identify the total number of forms submitted in the blank below.

☐ A total of _____ forms are submitted.

This collection of information is required by 37 CFR 1.31, 1.32, and 1.33. The information is required to obtain or retain a benefit by the public, which is to update (and by the USPTO to process) the file of a patent or reexamination proceeding. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 3 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
16/141,378	09/25/2018	Glenn Coleman	LCOM_ENT_CLOUD_US01

CONFIRMATION NO. 2440

199393
Atlantic IP
c/o Lombard Geliebter LLP
1325 Avenue of the Americas
28th Floor
New York, NY 10019

POA ACCEPTANCE LETTER

OC00000007435499

Date Mailed: 06/25/2024

NOTICE OF ACCEPTANCE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 05/17/2024.

The Power of Attorney in this application is accepted. Correspondence in this application will be mailed to the above address as provided by 37 CFR 1.33.

Questions about the contents of this notice and the requirements it sets forth should be directed to the Office of Data Management, Application Assistance Unit, at **(571) 272-4000** or **(571) 272-4200** or **1-888-786-0101**.

/qnguyen/

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
16/141,378	09/25/2018	Glenn Coleman	LCOM_ENT_CLOUD_US01

CONFIRMATION NO. 2440

158266
Condo Roccia Koptiw LLP
1800 JFK Boulevard
Suite 1700
Philadelphia, PA 19103

POWER OF ATTORNEY NOTICE

OC0000000743549934

Date Mailed: 06/25/2024

NOTICE REGARDING CHANGE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 05/17/2024.

- The Power of Attorney to you in this application has been revoked by the assignee who has intervened as provided by 37 CFR 3.71. Future correspondence will be mailed to the new address of record(37 CFR 1.33).

Questions about the contents of this notice and the requirements it sets forth should be directed to the Office of Data Management, Application Assistance Unit, at **(571) 272-4000** or **(571) 272-4200** or **1-888-786-0101**.

/qnguyen/

TO: uspto@dockettrak.com,lbrown@atlanticips.com,docket@lombardip.com
FROM: noreply@uspto.gov
CC: patentcenter_eofficeaction@uspto.gov
SUBJECT: USPTO: Patent Electronic System - Correspondence Notification for Customer Number 199393

Tue Jun 25 05:00:20 EDT 2024

Dear Patent Center Customer:

Correspondence Address:

Atlantic IP
c/o Lombard Geliebter LLP
1325 Avenue of the Americas
28th Floor
New York,NEW YORK,10019
UNITED STATES

This is a courtesy notification regarding the following USPTO patent application(s) associated with your Customer Number, 199393, that have new outgoing correspondence. This correspondence is now available for viewing in Patent Center.

The official date of notification of the outgoing correspondence will be indicated on the form (e.g., PTOL-90) accompanying the correspondence.

Disclaimer:
The list of documents shown below are provided as a courtesy and is not part of the official file wrapper. The content of the images shown in the Image File Wrapper is the official record.

Application	Document	Mailroom Date	Attorney Docket No.
16141378	N570	06/25/2024	LCOM_ENT_CLOUD_US01
16141378	N570	06/25/2024	LCOM_ENT_CLOUD_US01

To view your correspondence online, please sign in to Patent Center and then select Workbench/View correspondence. To update your email address(es), select Manage/Manage customer numbers.

If you have any questions, please contact the Patent Electronic Business Center (EBC) at ebc@uspto.gov or 866-217-9197 Monday – Friday, 6 a.m. to midnight ET.

Please do not reply to this email as it was sent from an unmonitored mailbox.

Sincerely,

The Patent Center Team