US 20060021029A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2006/0021029 A1**

Brickell et al. (43) **Pub. Date: Jan. 26, 2006**

(54) **METHOD OF IMPROVING COMPUTER SECURITY THROUGH SANDBOXING**

(76) Inventors: **Ernie F. Brickell**, Portland, OR (US); **Clifford D. Hall**, Orangevale, CA (US); **Joseph F. Cihula**, Hillsboro, OR (US); **Richard Uhlig**, Hillsboro, OR (US)

Correspondence Address:
BLAKELY SOKOLOFF TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1030 (US)

(21) Appl. No.: **10/881,602**

(22) Filed: **Jun. 29, 2004**

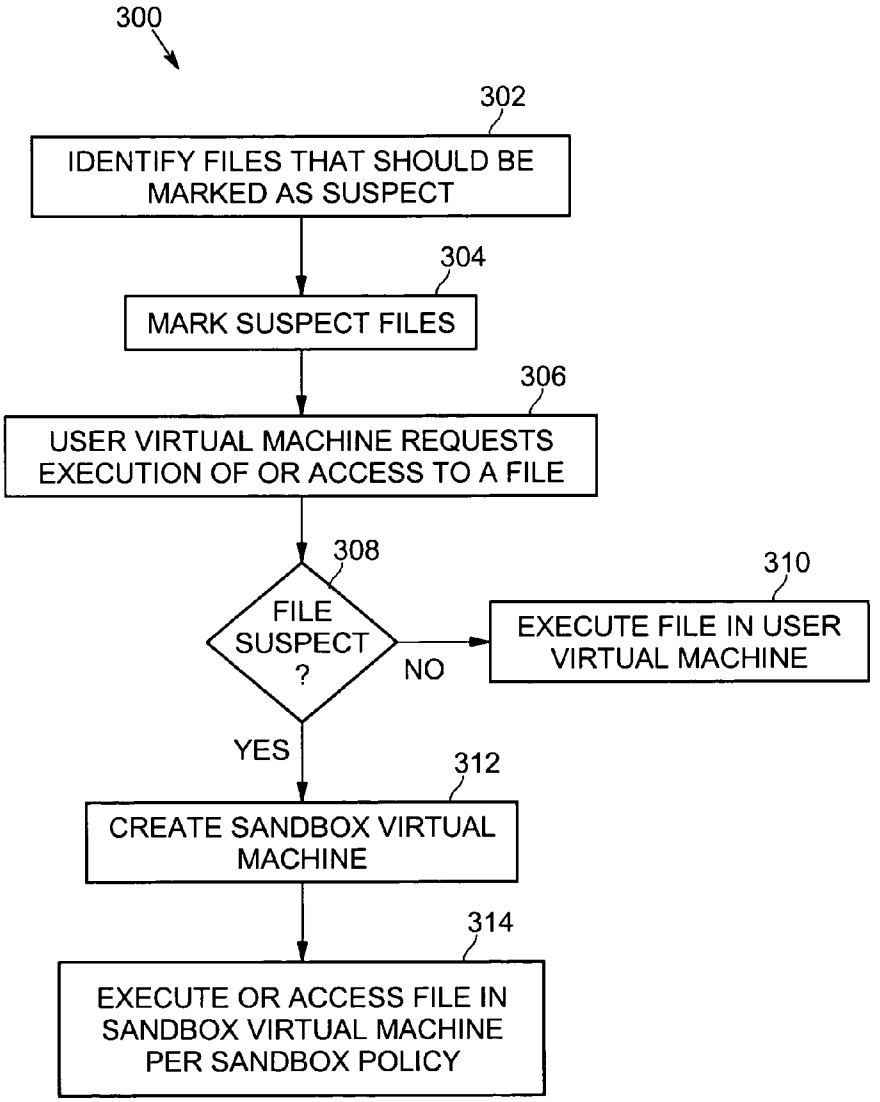
Publication Classification

(51) **Int. Cl.**
G06F 11/30 (2006.01)

(52) **U.S. Cl.** **726/22**

(57) **ABSTRACT**

Improving security of a processing system may be accomplished by at least one of executing and accessing a suspect file in a sandbox virtual machine.



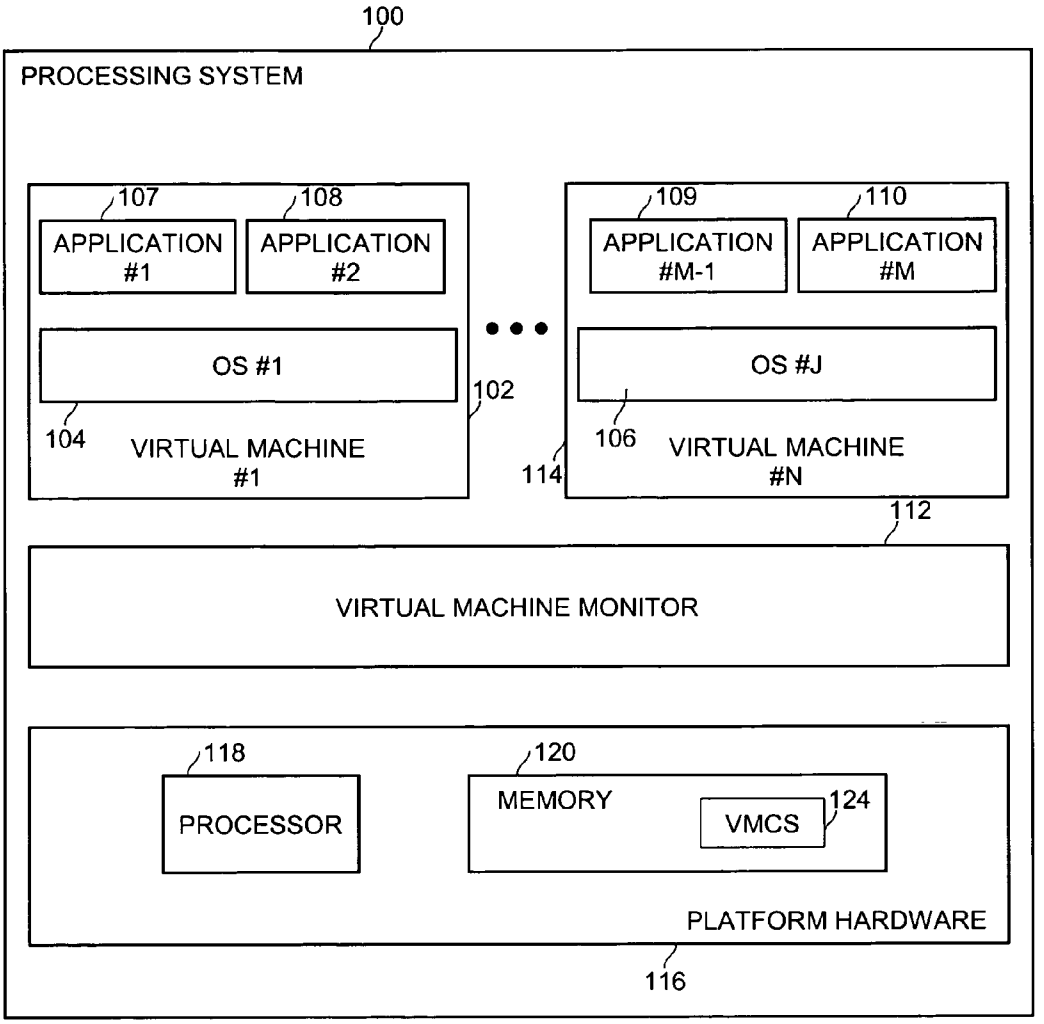

100
PROCESSING SYSTEM
107
APPLICATION #1
108
APPLICATION #2
OS #1
104
VIRTUAL MACHINE #1
102
109
APPLICATION #M-1
110
APPLICATION #M
OS #J
106
114
VIRTUAL MACHINE #N
112
VIRTUAL MACHINE MONITOR
118
PROCESSOR
120
MEMORY
124
VMCS
PLATFORM HARDWARE
116

Figure 1

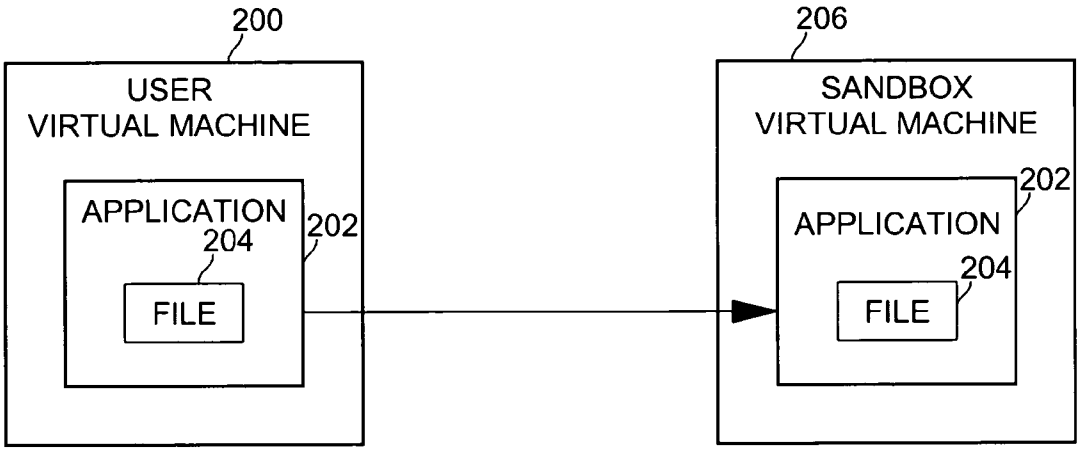

200
USER
VIRTUAL MACHINE
APPLICATION
204
202
FILE
206
SANDBOX
VIRTUAL MACHINE
202
APPLICATION
204
FILE

Figure 2

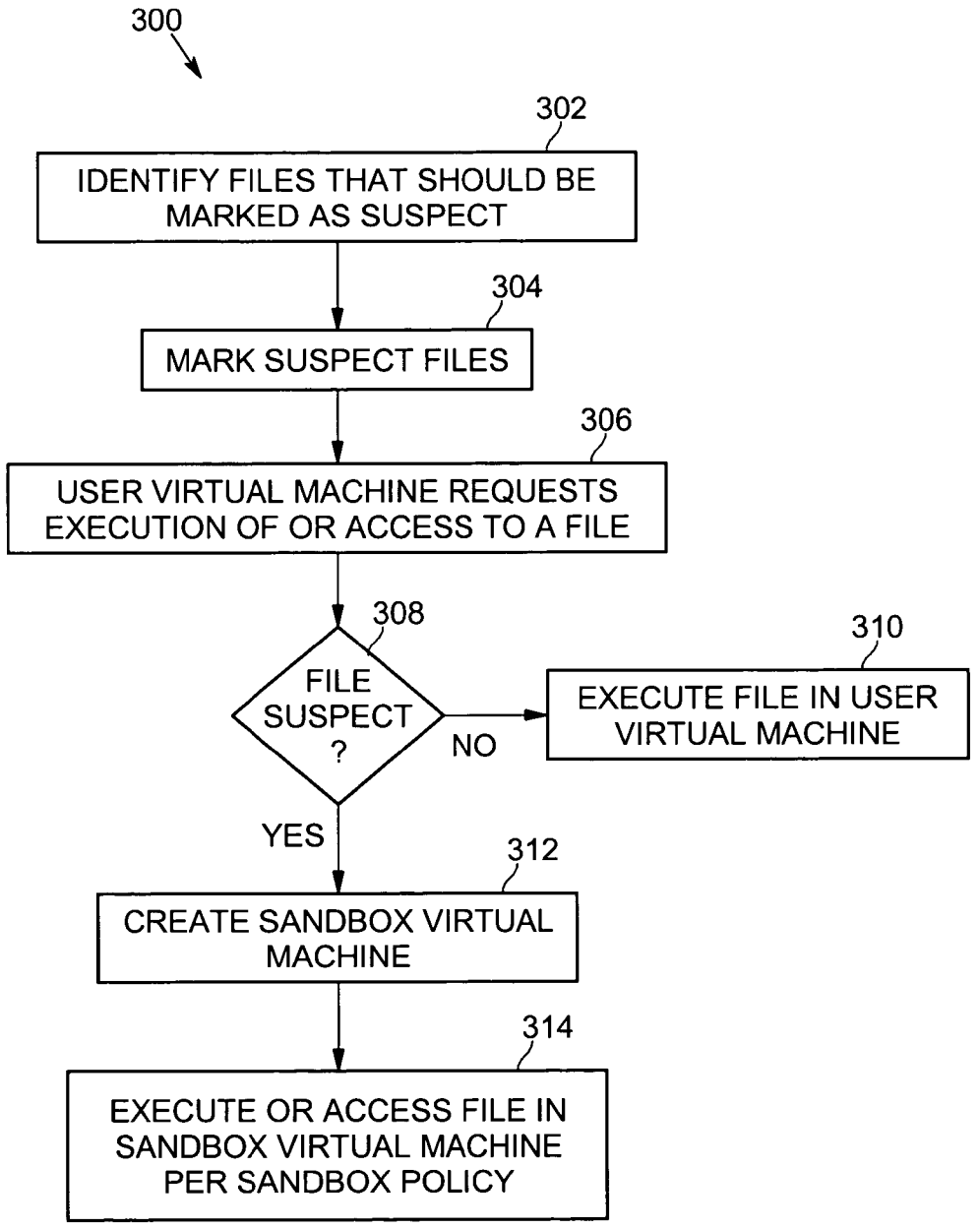

300
302
IDENTIFY FILES THAT SHOULD BE MARKED AS SUSPECT
304
MARK SUSPECT FILES
306
USER VIRTUAL MACHINE REQUESTS EXECUTION OF OR ACCESS TO A FILE
308
FILE SUSPECT ?
NO
310
EXECUTE FILE IN USER VIRTUAL MACHINE
YES
312
CREATE SANDBOX VIRTUAL MACHINE
314
EXECUTE OR ACCESS FILE IN SANDBOX VIRTUAL MACHINE PER SANDBOX POLICY

Figure 3

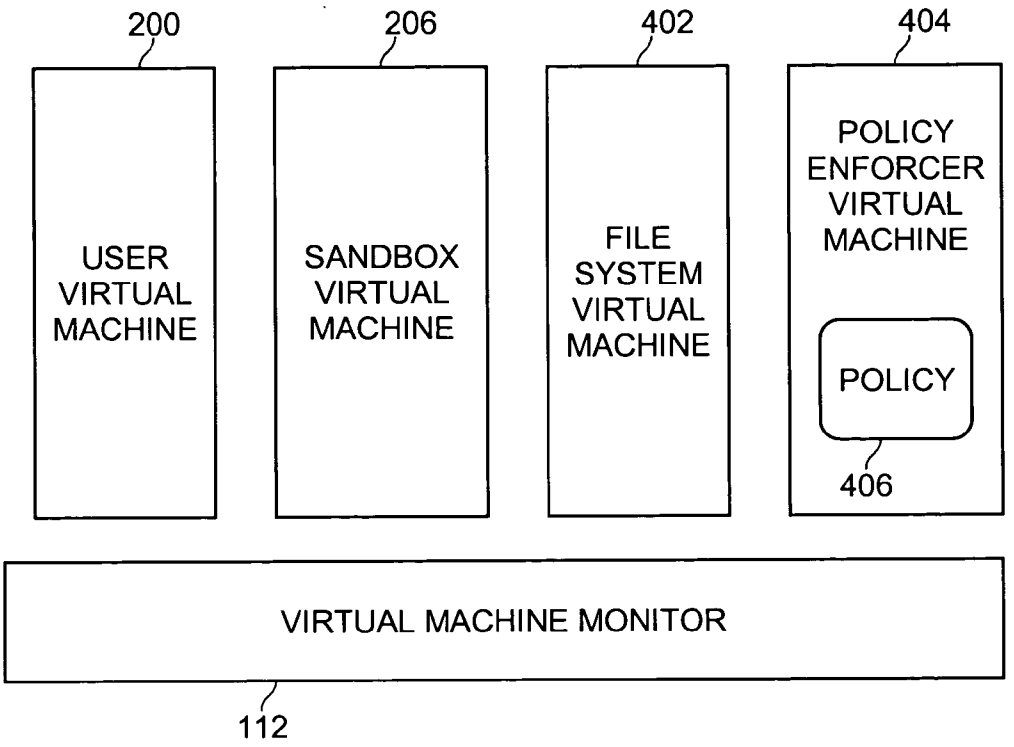

200
206
402
404
USER VIRTUAL MACHINE
SANDBOX VIRTUAL MACHINE
FILE SYSTEM VIRTUAL MACHINE
POLICY ENFORCER VIRTUAL MACHINE
POLICY
406
VIRTUAL MACHINE MONITOR
112

Figure 4

METHOD OF IMPROVING COMPUTER SECURITY THROUGH SANDBOXING

BACKGROUND

[0001] 1. Field

[0002] The present invention relates generally to computer security and, more specifically, to using virtualization techniques to improve the security of a computing platform.

[0003] 2. Description

[0004] Computer viruses are a common problem for computer users. One typical mode of attack is to send an electronic mail message (e-mail) containing a file attachment to an unsuspecting user's computer. The file attachment contains malicious attack code, and the e-mail may contain some inducement for the user to launch the file attachment. When the user clicks on the file attachment, the attack code embedded in the file is executed. The attack code accesses an address book and sends the file attachment in an e-mail to addresses found in the address book. The attack code may then try to modify files on the user's computer or obtain other files and mail them back to the attackers.

[0005] Propagation of such an attack is rapid. Once one unsuspecting user launches the file attachment, the virus quickly spreads to other unsuspecting users, who then perpetuate the problem. Such viruses have been known to overwhelm computer networks and cause millions of dollars of damage to network operators, companies, and users.

[0006] Techniques exist to detect viruses and purge them from affected computers. However, such techniques often are used only after the virus has been detected and many computers have been infected. New methods are desired that will slow down the propagation of computer viruses and other malicious code, thus allowing the virus detectors to detect and delete the viruses before the damage becomes widespread.

[0007] Along with improving the ability to detect and slow such attacks it is also desired to prevent or limit damage to users' systems and access to users' data. The ideal world in which users would never run suspicious files will never exist, so a practical solution must recognize this and attempt to prevent or limit the program from damaging the user's system and accessing the user's data.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

[0009] **FIG. 1** illustrates one embodiment of a virtual machine environment, in which one embodiment of the present invention may operate;

[0010] **FIG. 2** is a diagram of a user virtual machine interacting with a sandbox virtual machine according to an embodiment of the present invention;

[0011] **FIG. 3** is a flow diagram illustrating usage of a sandbox virtual machine according to an embodiment of the present invention; and

[0012] **FIG. 4** is a diagram illustrating a virtual machine environment according to an embodiment of the present invention.

DETAILED DESCRIPTION

[0013] An embodiment of the present invention is a method of using sandboxing to improve the security of a computing platform. Recent advances in virtualization give a computing platform the ability to run multiple virtual machines of protected computing environments so that execution of one environment will not interfere with the execution of another environment. Embodiments of the present invention use virtualization to create a sandbox virtual machine that is isolated from the rest of the computing platform. The sandbox may be used to open suspect files or execute suspect application programs so that if there is attack code in the suspect files or application programs, the attack will be contained in the sandbox. The attack code may then be dealt with according to prescribed policies. By accessing the suspect files only in the sandbox, further propagation of the attack code may be diminished and the attack may be more easily detected.

[0014] References in the specification to "one embodiment" or "an embodiment" of the present invention mean that a particular feature, structure or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrase "in one embodiment" appearing in various places throughout the specification are not necessarily all referring to the same embodiment.

[0015] Some portions of the detailed descriptions that follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer system's registers or memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0016] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or the like, may refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer-system memories or registers or other such information storage, transmission or display devices.

[0017] In the following detailed description of the embodiments, reference is made to the accompanying drawings that show, by way of illustration, specific embodiments in which the invention may be practiced. In the drawings, like numer-

IPR2025-01397
CrowdStrike EX1025 Page 6

als describe substantially similar components throughout the several views. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, and electrical changes may be made without departing from the scope of the present invention. Moreover, it is to be understood that the various embodiments of the invention, although different, are not necessarily mutually exclusive. For example, a particular feature, structure, or characteristic described in one embodiment may be included within other embodiments. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims, along with the full scope of equivalents to which such claims are entitled.

[0018] In some embodiments, the present invention may be provided as a computer program product or software which may include a machine or computer-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. In other embodiments, steps of the present invention might be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

[0019] Thus, a machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer), but is not limited to, floppy diskettes, optical disks, Compact Disc, Read-Only Memory (CD-ROMs), and magneto-optical disks, Read-Only Memory (ROMs), Random Access Memory (RAM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), magnetic or optical cards, flash memory, a transmission over the Internet, electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.) or the like.

[0020] **FIG. 1** illustrates one embodiment of a virtual machine environment for a processing system **100**, in which the present invention may operate. In this embodiment, bare platform hardware **116** comprises a computing platform, which may be capable, for example, of executing a standard operating system (OS) and a virtual machine monitor (VMM), such as a VMM **112**.

[0021] The VMM **112**, though typically implemented in software, may emulate and export a bare machine interface to higher level software. Such higher level software may comprise a standard or real-time OS, or may be a highly stripped down operating environment with limited operating system functionality that may not include traditional OS facilities, etc. The software may also include a version of firmware such as a BIOS. Alternatively, for example, the VMM **112** may be run within, on top of, or in parallel with another VMM. VMMs may be implemented, for example, in hardware, software, or firmware, or by a combination of various techniques.

[0022] The processing system **100** may be any one of a personal computer (PC), mainframe, handheld device, portable computer, set-top box, or any other computing system. The platform hardware **116** includes a processor **118** and memory **120**.

[0023] Processor **118** can be any type of processor capable of executing software, such as a microprocessor, digital signal processor, microcontroller, or the like. The processor **118** may include microcode, programmable logic or hard-coded logic for performing the execution of method embodiments of the present invention. Although **FIG. 1** shows only one such processor **118**, there may be one or more processors in the system, and each processor may run the same VMM, different VMMs, multiple VMMs, or no VMMs.

[0024] Memory **120** can be a hard disk, a floppy disk, random access memory (RAM), read only memory (ROM), flash memory, any combination of the above devices, or any other type of machine medium readable by processor **118**. Memory **120** may store instructions and/or data for performing the execution of method embodiments of the present invention.

[0025] The VMM **112** presents to other software (i.e., "guest" software) the abstraction of one or more virtual machines (VMs), which may provide the same or different abstractions to the various guests. **FIG. 1** shows two VMs, virtual machine #**1102** and virtual machine #N **114**, although in any given system any number of virtual machines may be implemented. The guest software running on each VM may include a guest OS such as guest OS #**1104** or OS #J **106** and various guest software applications **107**, **108**, **109** and **110**. Each of the guest OSs **104** and **106** expect to access physical resources (e.g., processor registers, memory and I/O devices) within the VMs **102** and **114** on which the guest OS **104** or **106** is running and to perform other functions. For example, the guest OS may expect to have access to all registers, caches, structures, I/O devices, memory and the like, according to the architecture of the processor and platform presented in the VM. The resources that can be accessed by the guest software may either be classified as "privileged" or "non-privileged." For privileged resources, the VMM **112** facilitates functionality desired by guest software while retaining ultimate control over these privileged resources. Non-privileged resources do not need to be controlled by the VMM **112** and can be accessed by guest software.

[0026] Further, each guest OS expects to handle various events such as exceptions (e.g., page faults, general protection faults, etc.), interrupts (e.g., hardware interrupts, software interrupts), and platform events (e.g., initialization (INIT) and system management interrupts (SMIs)). Some of these events are "privileged" because they must be handled by the VMM **112** to ensure proper operation of VMs **102** and **114** and for protection from and among guest software.

[0027] When a privileged event occurs or guest software attempts to access a privileged resource, control may be transferred to the VMM **112**. The transfer of control from guest software to the VMM **112** is referred to herein as a VM exit. After facilitating the resource access or handling the event appropriately, the VMM **112** may return control to guest software. The transfer of control from the VMM **112** to guest software is referred to as a VM entry.

[0028] In one embodiment, the processor **118** controls the operation of the VMs **102** and **114** in accordance with data stored in a virtual machine control structure (VMCS) **124**. The VMCS **124** is a structure that may contain the state of guest software, the state of the VMM **112**, execution control information indicating how the VMM **112** wishes to control

IPR2025-01397
CrowdStrike EX1025 Page 7

operation of guest software, information controlling transitions between the VMM **112** and a VM, etc. The processor **118** reads information from the VMCS **124** to determine the execution environment of the VM and to constrain its behavior. In one embodiment, the VMCS is stored in memory **120**. In some embodiments, multiple VMCS structures are used to support multiple VMs.

[**0029**] As used herein, a sandbox is an execution environment in which the code executing in the environment is restricted from some functionality. In embodiments of the present invention, a sandbox may allow code to execute, but the code may be quarantined so that the code cannot damage or access any of the computing platform outside of the sandbox. A file that is suspected of being malicious should be opened in a sandbox, so that if the file is malicious, the damage it might cause will be limited or prevented. In addition, a sandbox may be monitored for suspicious activity occurring, such as that during the opening of files or executing of code. In one embodiment, a sandbox may be implemented as a virtual machine.

[**0030**] **FIG. 2** is a diagram of a user virtual machine interacting with a sandbox virtual machine according to an embodiment of the present invention. In one example, a user may cause the execution of one or applications in a user virtual machine **200**. For example, one application **202** may be an e-mail program. The user's computer may receive an e-mail that contains a file attachment **204**. When the user clicks on the file attachment, instead of opening the file in the user virtual machine **200**, the attachment **204** may be opened in a sandbox virtual machine **206** by another copy of the application **202**, or perhaps by another program (not shown). The user may still be able to view the attachment as presented by the sandbox virtual machine, but the rest of the computing platform hosting the sandbox virtual machine may be protected from operations that occur in the sandbox. That is, the sandbox virtual machine may prevent certain specified operations from affecting files or other system resources outside of the sandbox. Thus, if the file attachment contains attack code that tries to distribute the attack to other users by accessing the address book of the e-mail application, the distribution will not be successful from the sandbox if the sandbox prevents the application from sending e-mails.

[**0031**] **FIG. 3** is a flow diagram **300** illustrating usage of a sandbox virtual machine according to an embodiment of the present invention. At block **302**, an entity of the processing system identifies files that should be marked as suspect. In one embodiment, the entity may be the virtual machine manager (VMM). The user may also participate in the decision to mark files. In another embodiment, a special virtual machine may be running to perform sandbox management operations. A suspect file may be any file that is not yet trusted by the processing system. Upon entry into a processing system, a file or application program may be evaluated according to a predetermined file identification policy as to whether it should be marked as suspect or not. One example policy is that all files or applications not created on the processing system should be marked as suspect when they are received and first stored in the processing system. Another policy may be that if the file or application was digitally signed by a trusted signature key, then the file or application would not be identified as suspect, otherwise it would be considered suspect. These policies are merely illustrative, and other methods of identifying suspect files and applications may be used without departing from the scope of the present invention.

[**0032**] At block **304**, once files are determined to be suspect, the entity marks the suspect files to denote that they are suspect. One skilled in the art will recognize that there are many ways to mark the files. One example method for marking files as suspect is to create an extension to the file system so that there is an annotation on each file indicating whether it is suspect or not. In one embodiment, this annotation may be a binary flag, where when the flag is set the file is considered suspect.

[**0033**] At some point in time after some marking activity has been performed, at block **306** software operating within a user virtual machine (such as an application program, for example) may request execution of or access to a file. For example, an e-mail attachment may be selected by the user to be executed. In another example, the user desires to have a selected application program access a file stored in the file system. At block **308**, it may be determined whether the file is currently marked as suspect. If the file is not suspect, then the file may be executed or accessed within the user virtual machine at block **310**. If the file is suspect, then the file may be processed within a sandbox virtual machine. In one embodiment, at block **312** a sandbox virtual machine may be created to process this particular file access request. In another embodiment, a permanent sandbox virtual machine may be active in the processing system to handle all such requests to access suspect files.

[**0034**] In another embodiment, the sandbox virtual machine may be created by "forking" a new virtual machine environment to construct a duplicate copy of the user virtual machine at the time that the sandbox is needed. One benefit of forking is that the original copy and the duplicate copy can begin to diverge, with the state of the original copy going completely unmodified, and hence serving as a known-good "checkpoint" of the machine state. If an attack is later observed in the duplicate copy, the system can simply revert back to the known-good checkpoint. VM "forking", therefore, goes a step beyond containing viruses or slowing their propagation, to also offering a method for recovering from an attack.

[**0035**] Once a sandbox virtual machine is active to handle the file access, at block **314** the file may be accessed or executed in the sandbox virtual machine according to the rules of a specified sandbox policy. The sandbox policy may define what actions may be taken as part of the file access. For example, software executing in a sandbox may be held to a policy that e-mail is not allowed to be sent from the sandbox. This may be enforced by having a policy checker component within the processing system capture any requests by the sandbox to send e-mail. As another example, software executing within the sandbox may not be allowed to delete or modify files in the processing system. In one embodiment this may be enforced by having all requests to modify files virtualized so that it would appear to the sandbox virtual machine that a request to modify files was being fulfilled, but in reality the request would be fulfilled using temporary files, so that no changes to permanent system files would be made. In another embodiment where the sandbox virtual machine was forked from a user virtual machine, after executing for period of time in a sandbox, the

IPR2025-01397
CrowdStrike EX1025 Page 8

forked environment may be merged back into the user environment. At this point, a decision may be made as to whether to accept the changes to files requested by the sandbox virtual machine. In one example, no changes to system files made from within a sandbox may be accepted. Alternatively, if it is detected that the forked environment has be compromised by a virus attack, it could simply be discarded, and system operation can revert back to the original VM state, which serves a known-good checkpoint.

[0036] While a suspect file is executing in or being accessed within a sandbox virtual machine, an entity such as the virtual machine monitor (VMM) may be monitoring the sandbox for behavior that indicates attack code is being run. For example, the VMM may monitor the sandbox for changes to system files, for automated e-mail requests, or for attempts to access sensitive documents. If the VMM detects any of these behaviors, then the VMM could respond in one or more prescribed manners. For example, the VMM could mark the file as probable attack code, have the code deleted, notify the user, send the file to a security server for further evaluation, or perform other predetermined actions.

[0037] With embodiments of the present invention, not only can executable files be marked as suspect and executed in a sandbox, but in addition, data files can be marked as suspect and the application to access the data file can be executed in a sandbox. For example, an application might execute in the user virtual machine when it is accessing a trusted data file, but would execute in a sandbox when it is accessing a suspect data file. This accommodates the idea that code can be trusted, but when trusted code executes untrusted data, the resulting combination may still contain an attack.

[0038] Once a file has been marked as suspect, the file will remain marked as suspect unless some specific action is taken to remove the suspect marking on the file. A suspect file could be accessed multiple times and still remain marked as suspect. One policy for implementing the functionality of unmarking a suspect file is for the user to operate a virtual machine that is not a sandbox, and to request the import of the suspect file into that virtual machine. Another policy that may be used to unmark a suspect file is that after a file has been executed within a sandbox for a specified period of time without evidence of behavior indicative of attack code, then the file's suspect marking may be changed. In one embodiment, this unmarking may be implemented by unsetting the suspect flag for the file. In another embodiment, the user may selectively un-mark a file once it is believed to be trusted.

[0039] **FIG. 4** is a diagram illustrating a virtual machine environment according to an embodiment of the present invention. In this embodiment, rather than have a VMM monitor the operations of the sandbox virtual machine, the entity performing the monitoring may be policy enforcer virtual machine **404**. Alternatively, the monitoring entity could instead reside (at least in part) in another VM that performs additional actions. This virtual machine may interact with file system virtual machine **402** to ensure that attack code executing within the sandbox cannot modify files in the processing system. Requests to access the file system may be handled by the file system virtual machine. The policy enforcer virtual machine may enforce a predetermined policy **406** that describes the activities allowable by code being executed within the sandbox. In one example, the sandbox virtual machine may not be able to directly access the network so that any request by the sandbox virtual machine to access the network would be sent to the policy enforcer. The policy enforcer would then check the policy to see if the network access would be allowed. In this way, the policy enforcer could keep the sandbox virtual machine from sending email or from sending any information to the network.

[0040] Embodiments of the present invention may help to deter rapid propagation of some computer viruses. If a file attachment having attack code is opened using the present invention, the attachment would be opened in a sandbox. The sandbox may not be allowed to send out e-mail (depending on how the policy was defined), so the attack would not be replicated. The attack code might modify system files in the sandbox, but these changes would be made virtually, so that the actual system files would not be modified. After opening the file in the sandbox, a user may decide to move the file into another virtual machine. Although this is not as preferable as keeping the file in the sandbox, even this scenario may slow down virus propagation.

[0041] Although the operations may be described herein as a sequential process, some of the operations may in fact be performed in parallel or concurrently. In addition, in some embodiments the order of the operations may be rearranged without departing from the spirit of the invention.

[0042] While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.

What is claimed is:

1. A method of improving security of a processing system comprising:

at least one of executing and accessing a suspect file in a sandbox virtual machine.

2. The method of claim 1, further comprising at least one of executing and accessing the suspect file according to a sandbox policy defining allowable actions to be taken during at least one of execution and accessing of the suspect file.

3. The method of claim 2, wherein the sandbox policy includes preventing at least one of sending electronic mail messages, deleting or modifying files on the processing system, and sending messages through a network interface, all being initiated from within the sandbox virtual machine.

4. The method of claim 1, further comprising identifying a file to be marked as suspect.

5. The method of claim 4, wherein identifying a file to be marked as suspect comprises evaluating the file according to a file identification policy.

6. The method of claim 5, wherein identifying the file to be marked as suspect comprises identifying the file by a virtual machine monitor.

7. The method of claim 4, further comprising marking the suspect file to identify the file as suspect.

8. The method of claim 1, further comprising creating the sandbox virtual machine for at least one of executing and accessing the suspect file.

IPR2025-01397
CrowdStrike EX1025 Page 9

9. The method of claim 1, further comprising creating the sandbox virtual machine for processing at least one of executing and accessing one or more suspect files in the processing system.

10. The method of claim 1, further comprising creating the sandbox virtual machine by forking the sandbox virtual machine from a user virtual machine.

11. The method of claim 7, further comprising changing the marking of a suspect file when monitoring of at least one of execution and accessing of the suspect file indicates that the file is not malicious.

12. The method of claim 7, further comprising changing the marking of a suspect file from suspect to not suspect from within a user virtual machine.

13. The method of claim 1, further comprising determining if a file is marked as suspicious prior to attempting executing or accessing the file in a user virtual machine.

14. An article comprising: a storage medium having a plurality of machine readable instructions, wherein when the instructions are executed by a processor, the instructions provide for improving security of a processing system by at least one of executing and accessing a suspect file in a sandbox virtual machine.

15. The article of claim 14, further comprising instructions for at least one of executing and accessing the suspect file according to a sandbox policy defining allowable actions to be taken during at least one of execution and accessing of the suspect file.

16. The article of claim 15, wherein the sandbox policy includes preventing at least one of sending electronic mail messages, deleting or modifying files on the processing system, and sending messages through a network interface, all being initiated from within the sandbox virtual machine.

17. The article of claim 14, further comprising instructions for identifying a file to be marked as suspect.

18. The article of claim 17, wherein instructions for identifying a file to be marked as suspect comprise instructions for evaluating the file according to a file identification policy.

19. The article of claim 18, wherein instructions for identifying the file to be marked as suspect comprise instructions for identifying the file by a virtual machine monitor.

20. The article of claim 17, further comprising instructions for marking the suspect file to identify the file as suspect.

21. The article of claim 15, further comprising instructions for creating the sandbox virtual machine for at least one of executing and accessing the suspect file.

22. The article of claim 15, further comprising instructions for creating the sandbox virtual machine for processing at least one of executing and accessing all suspect files in the processing system.

23. The article of claim 15, further comprising instructions for creating the sandbox virtual machine by forking the sandbox virtual machine from a user virtual machine.

24. The article of claim 20, further comprising instructions for changing the marking of a suspect file when monitoring of at least one of execution and accessing of the suspect file indicates that the file is not malicious.

25. The article of claim 20, further comprising instructions for changing the marking of a suspect file from suspect to not suspect from within a user virtual machine.

26. The article of claim 14, further comprising instructions for determining if a file is marked as suspicious prior to attempting executing or accessing the file in a user virtual machine.

27. A processing system comprising:

a virtual machine monitor to identify and mark files that are suspect; and

a sandbox virtual machine to at least one of execute and access a suspect file.

28. The processing system of claim 27, wherein the sandbox virtual machine is adapted to at least one of execute and access the suspect file according to a sandbox policy defining allowable actions to be taken during at least one of execution and accessing of the suspect file.

29. The processing system of claim 28, wherein the sandbox policy includes preventing at least one of sending electronic mail messages with the suspect file as an attachment and deleting files on the processing system, both being initiated from within the sandbox virtual machine.

30. The processing system of claim 27, wherein the virtual machine monitor identifies a file to be marked as suspect by evaluating the file according to a file identification policy.

31. The processing system of claim 27, wherein the virtual machine monitor changes the marking of a suspect file when monitoring of at least one of execution and accessing of the suspect file by the sandbox virtual machine indicates that the file is not suspect.

32. A processing system comprising:

a sandbox virtual machine to at least one of execute and access a suspect file; and

a policy enforcer virtual machine to identify and mark files that are suspect, and to enforce a sandbox policy defining allowable actions to be taken by the sandbox virtual machine during at least one of execution and accessing of the suspect file.

33. The processing system of claim 32, wherein the sandbox policy includes preventing, by the policy enforcer virtual machine, at least one of sending electronic mail messages with the suspect file as an attachment and deleting files on the processing system, both being initiated from within the sandbox virtual machine.

34. The processing system of claim 32, wherein the policy enforcer virtual machine identifies a file to be marked as suspect by evaluating the file according to a file identification policy.

35. The processing system of claim 32, wherein the policy enforcer virtual machine changes the marking of a suspect file when monitoring of at least one of execution and accessing of the suspect file by the sandbox virtual machine indicates that the file is not suspect.

36. A method of improving security of a processing system comprising:

identifying files to be marked as suspect;

marking the identified files;

accepting a request to at least one of execute and access a file by a user virtual machine;

at least one of executing and accessing the file in the user virtual machine when the file is not marked as suspect; and

creating a sandbox virtual machine and at least one of executing and accessing the file in the sandbox virtual machine according to a policy when the file is marked as suspect.

37. The method of claim 36, wherein the policy includes preventing at least one of sending electronic mail messages with the suspect file as an attachment and deleting files on the processing system, both being initiated from within the sandbox virtual machine.

38. The method of claim 36, wherein identifying a file to be marked as suspect comprises evaluating the file according to a file identification policy.

39. The method of claim 38, wherein identifying the file to be marked as suspect comprises identifying the file by a virtual machine monitor.

* * * * *