US 20170054754A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2017/0054754 A1**

SAHER et al. (43) **Pub. Date: Feb. 23, 2017**

(54) **MALWARE AND EXPLOIT CAMPAIGN DETECTION SYSTEM AND METHOD**

(71) Applicant: **NSS LABS, INC.**, Austin, TX (US)

(72) Inventors: **Mohamed SAHER**, Austin, TX (US); **Jayendra PATHAK**, Austin, TX (US); **Ahmed ELGARHY**, Austin, TX (US)

(21) Appl. No.: **15/346,358**

(22) Filed: **Nov. 8, 2016**

Related U.S. Application Data

(63) Continuation-in-part of application No. 14/482,696, filed on Sep. 10, 2014.

(60) Provisional application No. 61/876,704, filed on Sep. 11, 2013.

Publication Classification

(51) **Int. Cl.**

H04L 29/06 (2006.01)

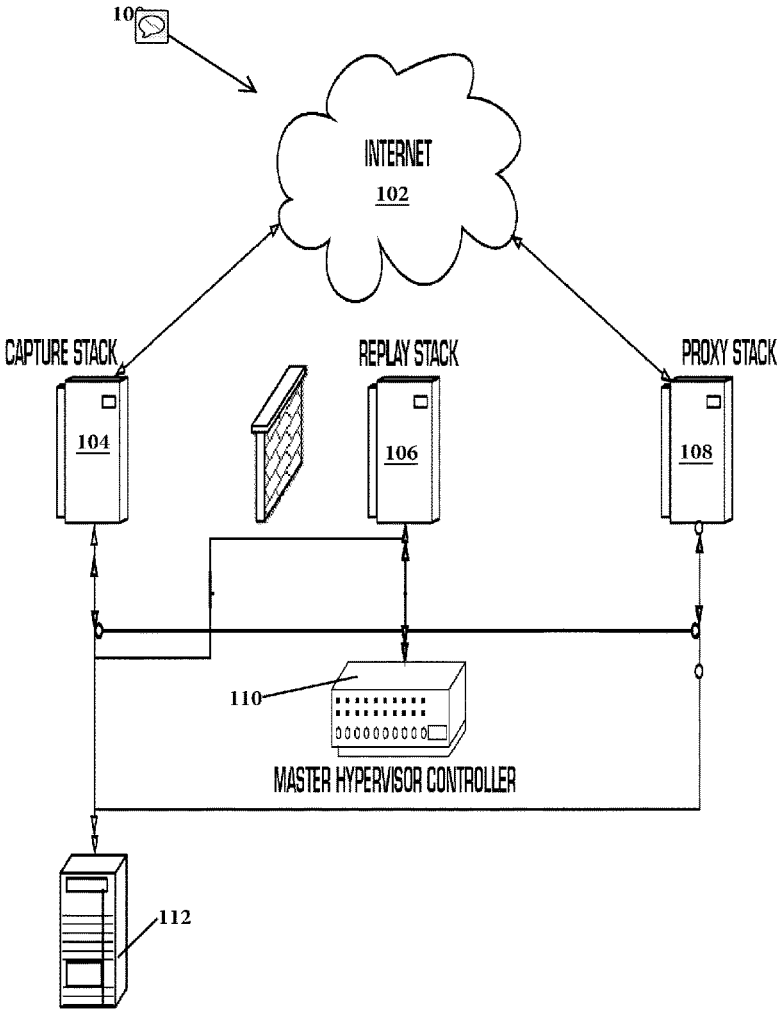
G06F 17/30 (2006.01)

(52) **U.S. Cl.**

CPC ***H04L 63/1491*** (2013.01); ***H04L 63/1416*** (2013.01); ***G06F 17/30864*** (2013.01); ***H04L 63/0272*** (2013.01)

(57) **ABSTRACT**

A malware and exploit campaign detection system and method are provided that cannot be detected by the malware or exploit campaign. The system may provide threat feed data to the vendors that produce in-line network security and end point protection (anti virus) technologies. The system may also be used as a testing platform for 3rd party products. Due to the massive footprint of the system's cloud infrastructure and disparate network connections and geo-location obfuscation techniques, NSS can locate and monitor malware across the globe and provide detailed threat analysis for each specific region, as they often support and host different malware/cybercrime campaigns.



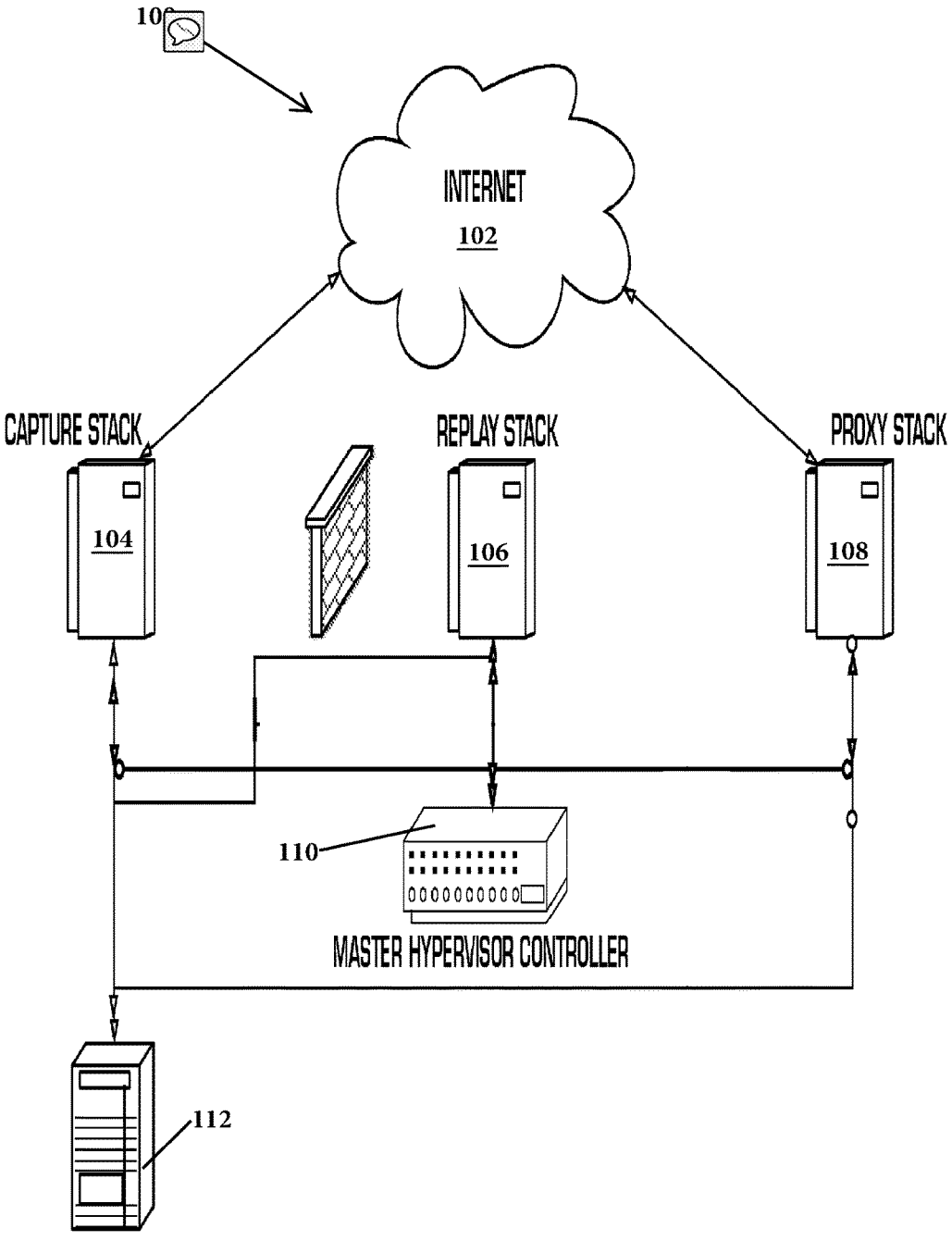

10
INTERNET
102
CAPTURE STACK
REPLAY STACK
PROXY STACK
104
106
108
110
MASTER HYPERVISOR CONTROLLER
112

FIGURE 1

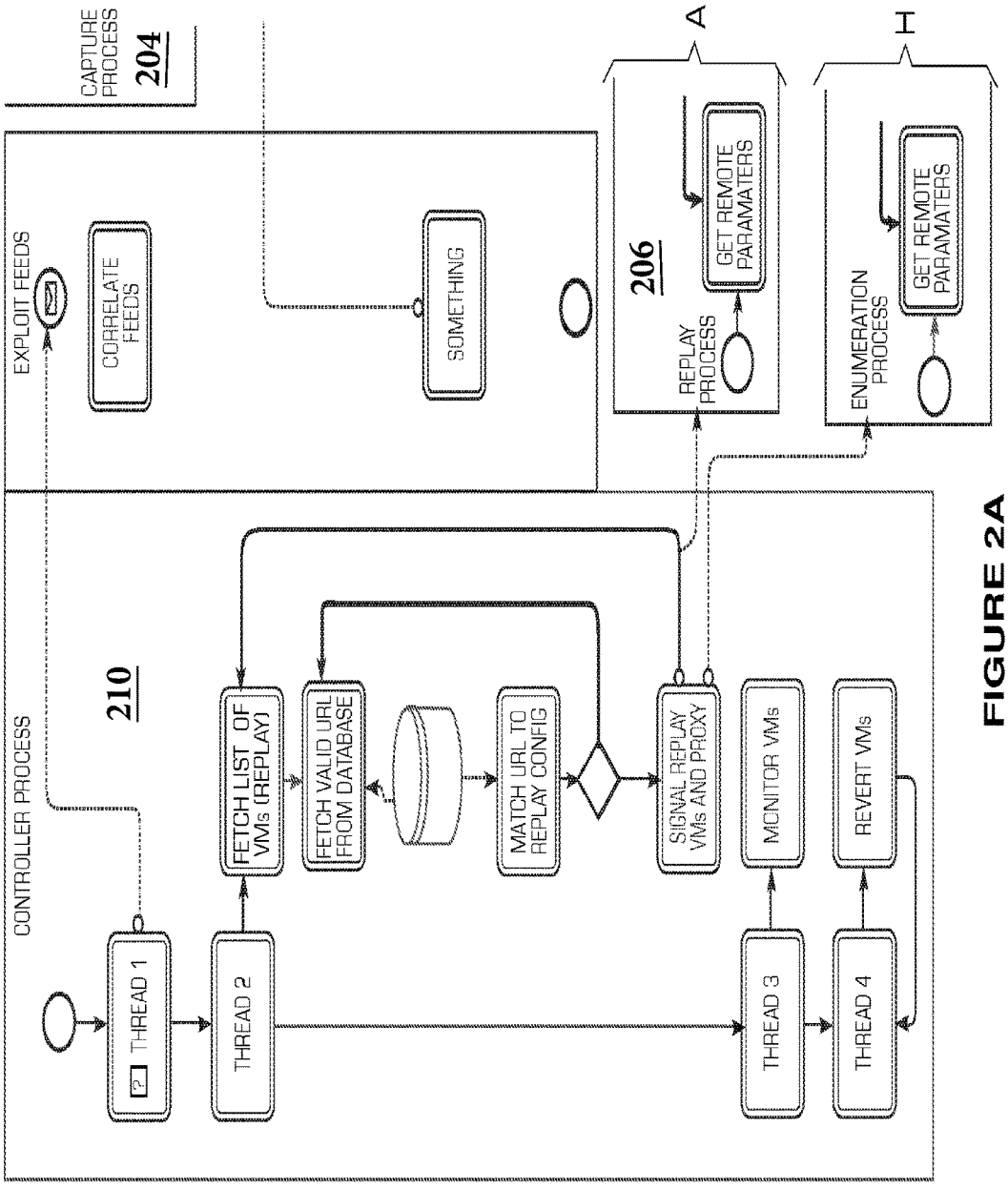

CAPTURE PROCESS
204
EXPLOIT FEEDS
CORRELATE FEEDS
SOMETHING
206
REPLAY PROCESS
GET REMOTE PARAMATERS
ENUMERATION PROCESS
GET REMOTE PARAMATERS
A
I
CONTROLLER PROCESS
210
THREAD 1
THREAD 2
FETCH LIST OF VMs (REPLAY)
FETCH VALID URL FROM DATABASE
MATCH URL TO REPLAY CONFIG
SIGNAL REPLAY VMs AND PROXY
MONITOR VMs
REVERT VMs
THREAD 3
THREAD 4

FIGURE 2A

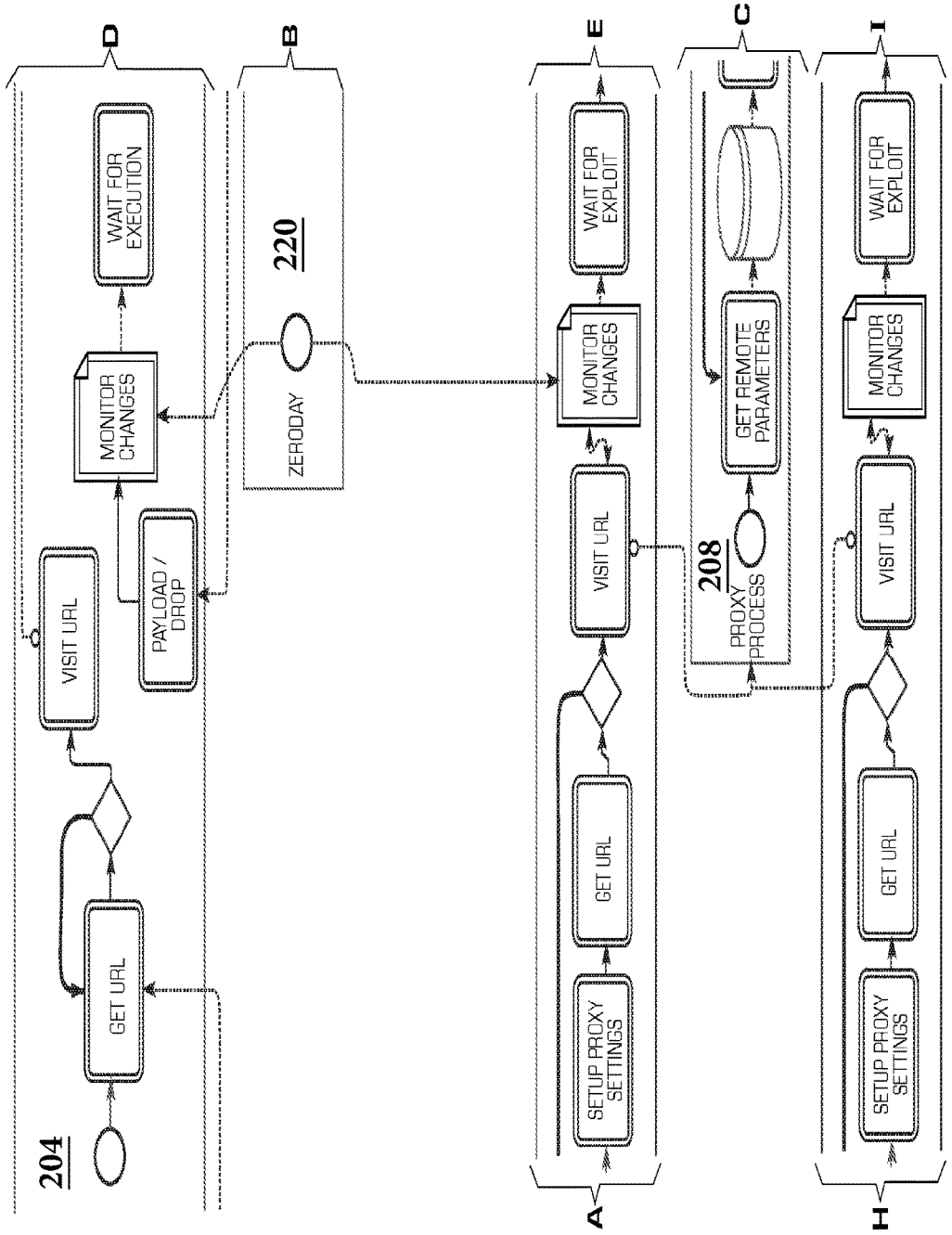

204
GET URL
VISIT URL
PAYLOAD / DROP
MONITOR CHANGES
WAIT FOR EXECUTION
D
B
220
ZERODAY
E
SETUP PROXY SETTINGS
GET URL
VISIT URL
MONITOR CHANGES
WAIT FOR EXPLOIT
A
C
208
PROXY PROCESS
GET REMOTE PARAMETERS
I
SETUP PROXY SETTINGS
GET URL
VISIT URL
MONITOR CHANGES
WAIT FOR EXPLOIT
H

FIGURE 2B

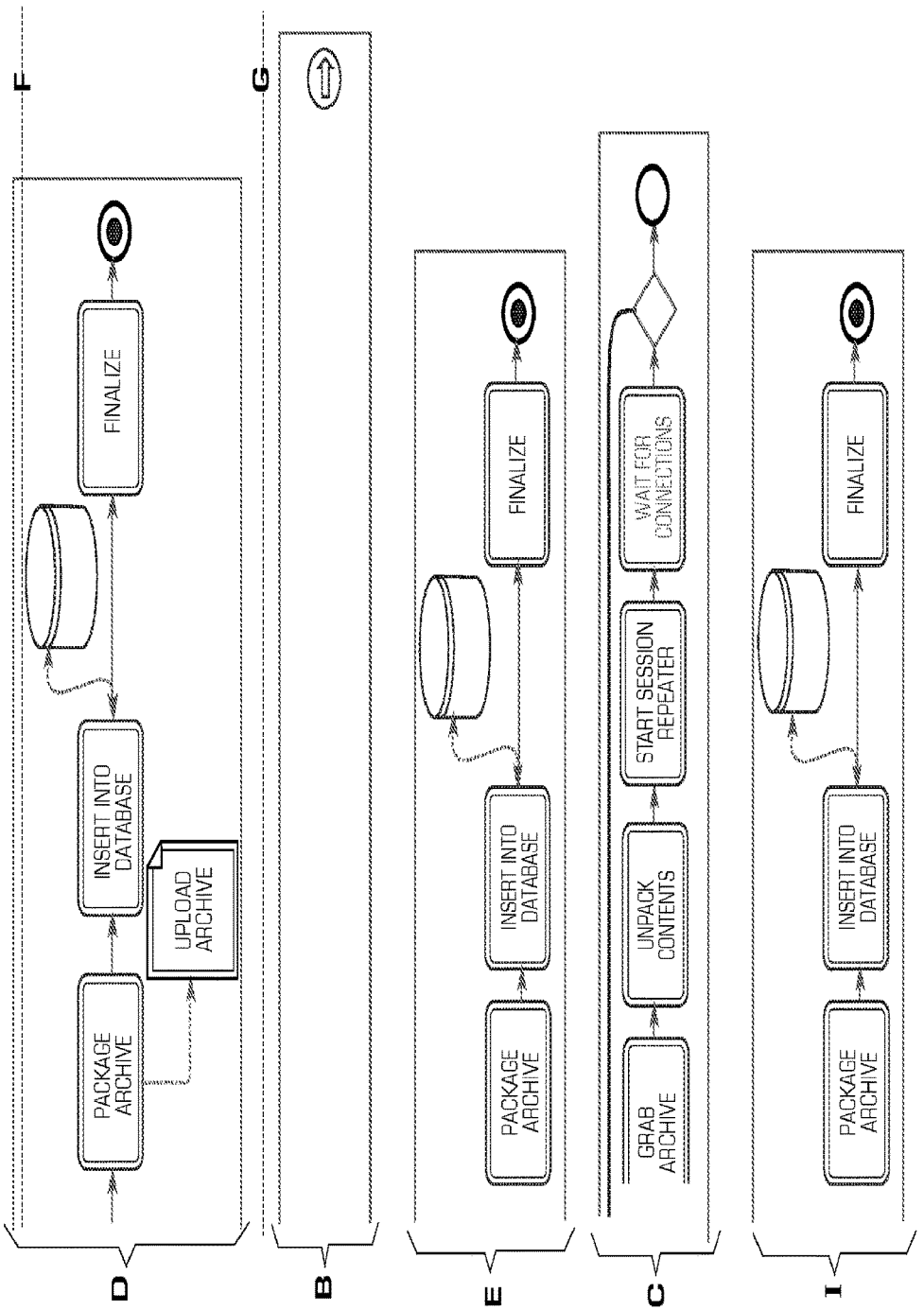

F
G
D
B
E
C
I
PACKAGE ARCHIVE
INSERT INTO DATABASE
UPLOAD ARCHIVE
FINALIZE
PACKAGE ARCHIVE
INSERT INTO DATABASE
FINALIZE
GRAB ARCHIVE
UNPACK CONTENTS
START SESSION REPEATER
WAIT FOR CONNECTIONS
PACKAGE ARCHIVE
INSERT INTO DATABASE
FINALIZE

FIGURE 2C

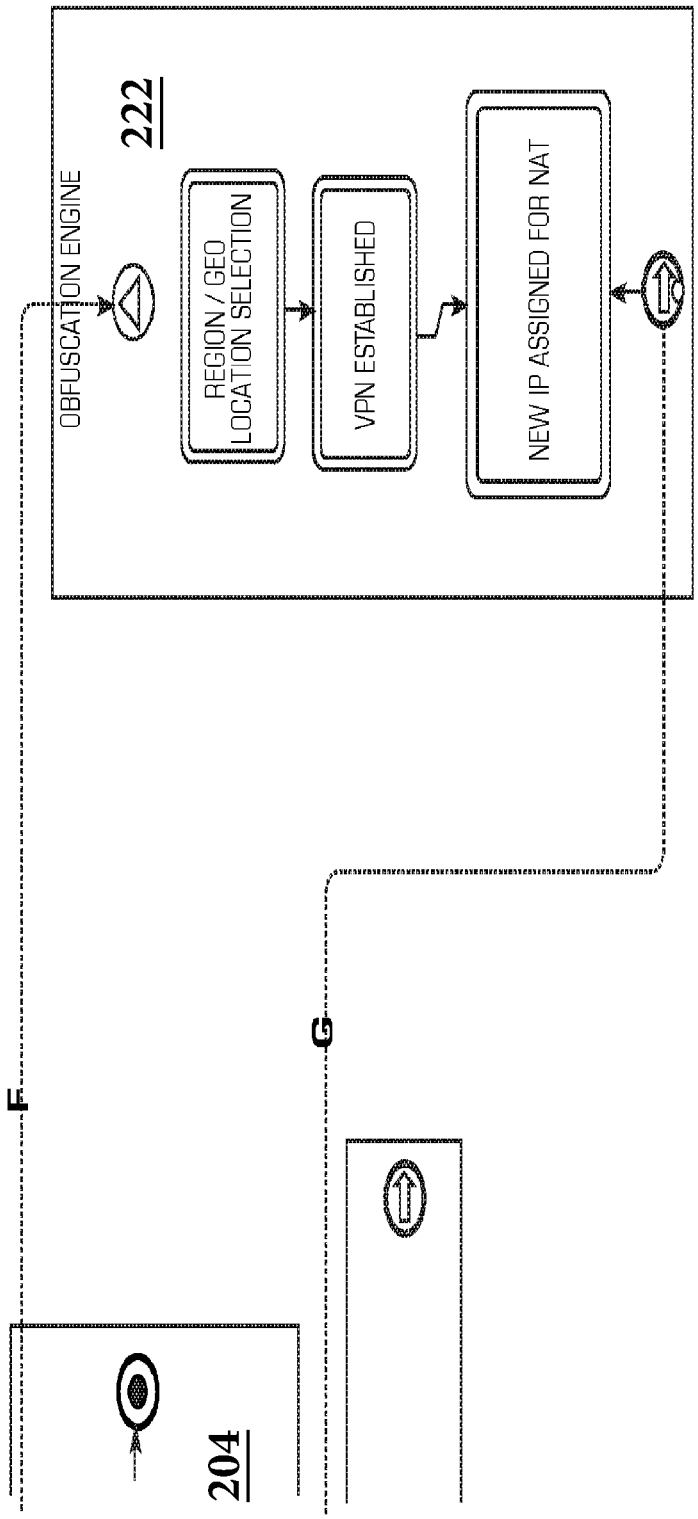

222
OBFUSCATION ENGINE
REGION / GEO LOCATION SELECTION
VPN ESTABLISHED
NEW IP ASSIGNED FOR NAT
F
G
204

FIGURE 2D


BAITNET WEB
HOME
REPORTS
DASHBOARD
CAPTURE REPORTS
LIST CAPTURES
CAPTURE GELOCATION
OS REPORTS
PLATFORM REPORTS
REPLAY REPORTS
SOURCE REPORTS
LIST OF CAPTURES (WITH ACTIVE SEARCH CRITERIA)
CAPTURE DATE
SOURCE
OPERATING SYSTEM
PLATFORM
VALID?
5/25/2013 3:51:00 AM
http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd1&cad=rja&ved=0CDEQFjAA&url=http%3A%2F%2Fwww.iei.gouv.fr%2F&ei=vgGeUYigJPeo4APO1YGADw&usg=AFQjCNEBf7AZ9KCPPa9MCu7io3AKPOLSw&sig2=w4Upgu8JEKKrEgb_mhE_Ew
MICROSOFT WINDOWS 7
JAVA 6 UPDATE 27
5/24/2013 4:04:00 PM
http://188.190.101.64/sea/vendor-regarding_mile_topic.htm
MICROSOFT WINDOWS XP
ADOBE READER 9.3
5/24/2013 4:04:00 PM
http://188.190.101.64/sea/vendor-regarding_mile_topic.htm
MICROSOFT WINDOWS XP
ADOBE READER 9.0
5/24/2013 3:26:00 PM
http://www.safetyins.net/brXUFKITSSK/CONNRQKOX/1662/1347/2951/21/2099937/index.htm
MICROSOFT WINDOWS 7
JAVA 6 UPDATE 27
5/23/2013 5:36:00 PM
http://xexec-news.com/t/4LG-1ITALI-A3LC8D-LN370-1/C.aspx
MICROSOFT WINDOWS XP
ADOBE READER 9.0
5/23/2013 4:19:00 PM
http://newssearch020.ru/flow10.php
MICROSOFT WINDOWS 7
JAVA 7

FIGURE 3


BAITNET WEB
HOME
REPORTS
DASHBOARD
CAPTURE REPORTS
OS REPORTS
PLATFORM REPORTS
REPLAY REPORTS
SOURCE REPORTS
MASTER DETAILS
CAPTURE DATE 5/25/2013 3:51:00 AM
ARCHIVE FILE NAME NSSID-234063-WIN7IE9-JREV2-5-25-2013-3-50-58-AM.zip
ARCHIVE HASH F940995CDDDB847AC5E7A1CFDC85FD98
VALID?
SOURCE DETAILS
ORIGIN DATE 5/23/2013 12:00:00 AM
URI http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CDEQFjAA&url=http%3A%2F%2Fwww.fei.gouv.fr%2F&ei=vg6eUYjgJPeo4APO1YGADw&usg=AFQjCNEB
TYPE URLQUERY
HOST IP ADDRESS 74.125.227.146
HOSTING COUNTRY UNITED STATES
OPERATING SYSTEM DETAILS
FULL NAME MICROSOFT WINDOWS 7
SHORT NAME N/A
FAMILY NAME MICROSOFT WINDOWS
PLATFORM DETAILS
FULL NAME JAVA 6 UPDATE 27
SHORT NAME N/A
FAMILY NAME JAVA

FIGURE 4

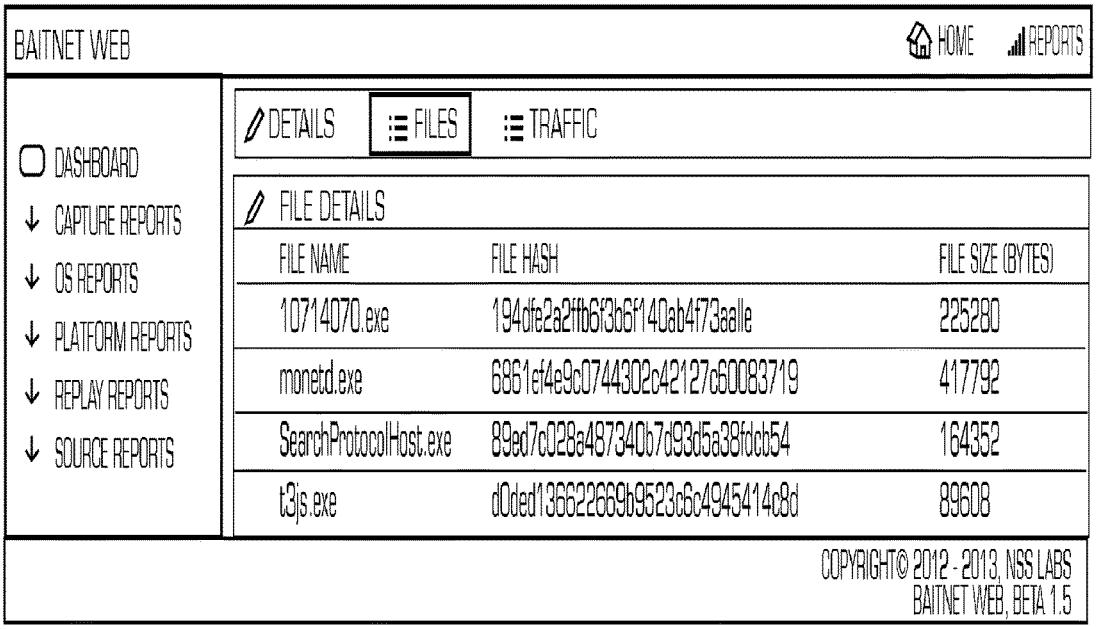

BAITNET WEB
HOME
REPORTS
DASHBOARD
CAPTURE REPORTS
OS REPORTS
PLATFORM REPORTS
REPLAY REPORTS
SOURCE REPORTS
DETAILS
FILES
TRAFFIC
FILE DETAILS
FILE NAME
FILE HASH
FILE SIZE (BYTES)
10714070.exe
194dfe2a2ffb6f3b6f140ab4f73aa11e
225280
monetd.exe
6861ef4e9c0744302c42127c60083719
417792
SearchProtocolHost.exe
89ed7c028a487340b7d93d5a38fdcb54
164352
t3js.exe
d0ded136622669b9523c6c4945414c8d
89608
COPYRIGHT© 2012 - 2013, NSS LABS
BAITNET WEB, BETA 1.5

FIGURE 5

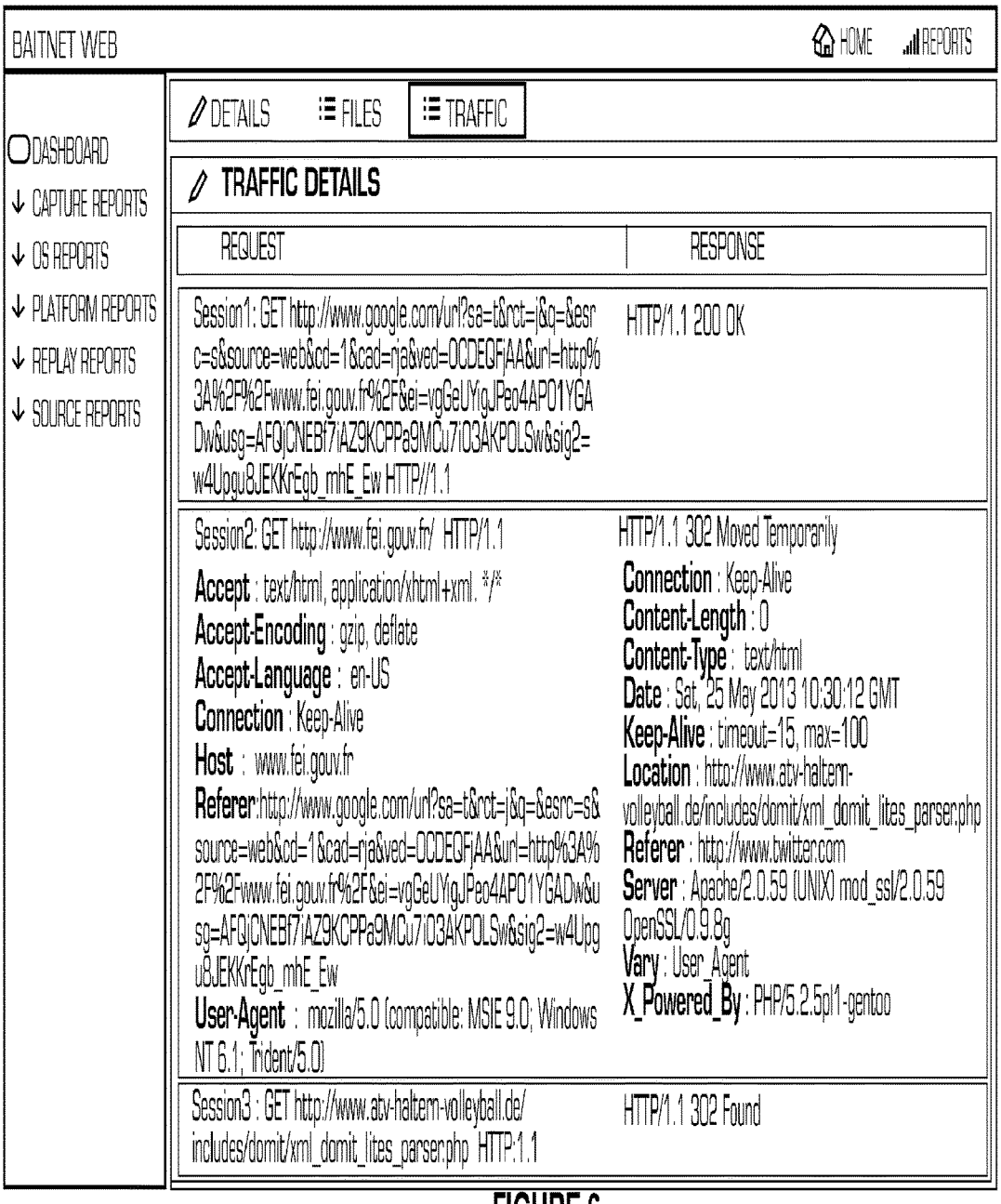

BAITNET WEB
HOME
REPORTS
DASHBOARD
CAPTURE REPORTS
OS REPORTS
PLATFORM REPORTS
REPLAY REPORTS
SOURCE REPORTS
DETAILS
FILES
TRAFFIC
TRAFFIC DETAILS
REQUEST
RESPONSE
Session1: GET http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CDEQFjAA&url=http%3A%2F%2Fwww.fei.gouv.fr%2F&ei=vgGeUYgJPeo4APO1YGADw&usg=AFQjCNEBf7iAZ9KCPPa9MCu7I03AKPOLSw&sig2=w4Upgu8JEKKrEgb_mhE_Ew HTTP//1.1
HTTP/1.1 200 OK
Session2: GET http://www.fei.gouv.fr/ HTTP/1.1
Accept : text/html, application/xhtml+xml, */*
Accept-Encoding : gzip, deflate
Accept-Language : en-US
Connection : Keep-Alive
Host : www.fei.gouv.fr
Referer:http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CDEQFjAA&url=http%3A%2F%2Fwww.fei.gouv.fr%2F&ei=vgGeUYgJPeo4APO1YGADw&usg=AFQjCNEBf7iAZ9KCPPa9MCu7I03AKPOLSw&sig2=w4Upgu8JEKKrEgb_mhE_Ew
User-Agent : mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)
HTTP/1.1 302 Moved Temporarily
Connection : Keep-Alive
Content-Length : 0
Content-Type : text/html
Date : Sat, 25 May 2013 10:30:12 GMT
Keep-Alive : timeout=15, max=100
Location : http://www.atv-haltern-volleyball.de/includes/domit/xml_domit_lites_parser.php
Referer : http://www.twitter.com
Server : Apache/2.0.59 (UNIX) mod_ssl/2.0.59 OpenSSL/0.9.8g
Vary : User_Agent
X_Powered_By : PHP/5.2.5pl1-gentoo
Session3 : GET http://www.atv-haltern-volleyball.de/includes/domit/xml_domit_lites_parser.php HTTP:1.1
HTTP/1.1 302 Found

FIGURE 6

MALWARE AND EXPLOIT CAMPAIGN DETECTION SYSTEM AND METHOD

PRIORITY CLAIMS/RELATED APPLICATIONS

[0001] This application claims priority under 35 USC 120 and is a continuation in part of U.S. patent application Ser. No. 14/482,696, filed Sep. 10, 2014 and titled "MALWARE AND EXPLOIT CAMPAIGN DETECTION SYSTEM AND METHOD" that in turn claims priority under 35 USC 120 and the benefit under 35 USC 119(e) to U.S. Provisional Patent Application Ser. No. 61/876,704 filed Sep. 11, 2013 and entitled "Malware And Exploit Campaign Detection System And Method", the entirety of both of which are incorporated herein by reference.

BACKGROUND

[0002] Intrinsically modern drive-by-exploitation and malware campaigns are growing in sophistication related to obfuscation, deployment, and execution in an effort to avoid detection and analysis by security researchers, and modern security systems and software. Anti-virus (AV) systems, such as endpoint protection platforms (EPPs), as well as breach detection services (BDS) employ virtual "sandboxes" or "honey nets" that operate in a cloud (virtual) network construct. These sandboxes attempt to identify malware and virus programs by incubating the suspect software until such time as the malware executes and its activities can be monitored and analyzed.

[0003] These systems often fail to identify previously unknown malware due to the evolution within malware development that allows the malware to recognize when it is sitting in such a system/trap. Modern malware can be considered to be "cognitive" and completely aware that it is currently being incubated within a trap (monitored system), and will continue to hibernate and therefore will not present itself as malicious software.

[0004] Thus the sandbox system will fail to identify the suspect file as being malicious, and therefore will allow all similar programs to bypass future testing. Another shortcoming is that they rely on monitoring traffic that is already affecting the victim machine or network, and because these sandbox systems are incapable of operating in-line due to performance limitations, they are unable to actually prevent the attack. The best the conventional systems can do is to inform when a breach has already occurred and thus can never be predictive.

BRIEF SUMMARY

[0005] The system and method for malware and exploit campaign detection (that may be known as "BaitNET") is different than known systems since the system has technology that prevents detection of the system by the malware/exploit. Unlike other technologies, BaitNET cannot be detected by modern malware/exploits and thus the operations/actions of the malware/exploits can be collected and analyzed without restriction. The collected malware/exploit is replayed/tested against various operating system and application configurations within BaitNET's private cloud infrastructure to determine what other system footprints are susceptible to the malware campaign. BaitNET is able to successfully incubate, track, and inventory the malware/exploit. Due to the transparency of BaitNET to the malware/exploit, BaitNET is able to perform live analysis that that can track threat actors, identify where they are truly located, and what other similar malware/exploit campaigns they have been launching and against whom. All of this is done while BaitNET produces threat forecasts that indicate viable and potential targets of the threat actors. BaitNET can also be used to measure and test the effectiveness of commercially available EPPs, AVs, in-line network security appliances, and BDS systems. This is done by injecting malware/exploits into BaitNET's construct, where these commercial products have already been installed, and then monitoring the delta between what BaitNET knows was injected, and detected itself, and what the commercial product claims to have detected. E.g., BaitNET is an advancement in technology so far beyond modern AV, EPP, and BDS that it is used to test the efficacy of these commercial products.

[0006] In one implementation, BaitNET is the conglomerate of a number of software applications, processes, and innovations as outlined herein which afford BaitNET the ability to shim into the operating system and the virtual machine architecture (both guest and host) enabling BaitNET to obfuscate the fact that the machine itself is a virtual/unmanned computer. The system utilizes a multitude of virtual private networks (VPNs) allowing a near-unlimited number of unique Internet IP addresses from all across the world to be used. These disparate IP addresses afford two primary advantages to BaitNET. One, in order to force re-infection, as many malware system will not "drop" (deploy) malware to the same IP address more than once, it is necessary to have BaitNET obfuscate its Internet presence. Two, many malware campaigns limit their targets by geo-location, which is often tracked via IP Address. E.g., Malware-infected servers often limit themselves to only infecting one (1) computer from any given masked IP address, and may limit the country of origin of the IP addresses that they will infect. BaitNET utilizes VPNs throughout the world to mimic dispersed geo-location and map out malware campaigns in different regions. Other techniques, while not proprietary to BaitNET, may also be used to emulate potential target qualification data points such as varying the language pack and keyboard language configuration on the host operating system.

[0007] After finding new malware, done by crawling URLs provided through various channels, BaitNET records the attack vector, payload, critical information on exploitation, and other relevant metadata and then "replays" this attack against thousands of other hosts on the BaitNET network. "Replay" is achieved through the use of BaitNET's proxy services, as outlined later in this document, and may be done against a singular image when testing the efficacy of a 3rd party security system or against limitless iterations of operating systems, application configurations, and versions of software tools when mapping the effectiveness of the exploit/malware. Each of the hosts used during the replay has a different combination of web browser, suite of installed applications, various program and operating system patch levels, installed language packages, etc. The representation of systems of nearly all possible combinations, Windows and OS X, from 2005 to present day. BaitNET is also capable of emulating mobile device operating systems, and uses the same technology to detect and inventory malware/exploits. All of this allows researchers to understand the true target landscape/scope for the malware/exploit, and the malware/exploit can be tested against anti virus (AV) and

in-line security systems such as intrusion prevention systems (IPS), next generation firewalls (NGFs), and breach detection systems (BDS.)

[0008] The BaitNET Framework is a distributed automation framework for testing URLs in real time to detect drive-by-exploitation attacks and malware dropped by said attacks, and gather data from said attacks to aid in their further analysis and prevention. URLs are tested using various operating system and application configurations within BaitNET's cloud infrastructure to determine if they are maliciously serving exploits. If a URL is found to be malicious, BaitNET is able to successfully incubate, track, and inventory the attack.

[0009] Due to the transparency of BaitNET to the exploit and any malware it drops, BaitNET is able to perform live analysis that that can track threat actors and fully enumerate their capabilities (i.e. which exploit kits they are using, which specific exploits are employed, which applications are being targeted, and full details of the exploits themselves). BaitNET therefore produces accurate predictions of which applications are being targeted in current campaigns by threat actors, providing predictive threat analysis AHEAD of any breach.

[0010] BaitNET can also be used to measure and test the effectiveness of commercially available security products, both network and host-based. This is done by replaying captured exploits using the same BaitNET infrastructure in which commercial security products have been installed. BaitNET is capable of monitoring the delta between what BaitNET detected during the initial capture process, and what the commercial product claims to have detected.

[0011] BaitNET has the concept of a Controller that acts as both a unit of work ventilator, or producer, and a lightweight in-memory message pump. Worker nodes, referred to as Victims, register themselves with the Controller to process units of work. The unit of work in BaitNET is a URL. Subscriber nodes, referred to as Notification Sinks, register themselves with the Controller to receive notifications about a URL's result as a Victim is processing it. The Controller, through a series of steps, distributes URLs to registered Victims to be processed, receives the results, and publishes them to registered Notification Sinks. BaitNET's cloud infrastructure is composed of a one or more Controllers and thousands of Victims preferably deployed in a virtualized environment. The exact number of Victims is based on the scope and scale of the testing and research being performed. Victims are machines with a unique operating system and application configuration that are responsible for testing URLs assigned to them by a Controller.

[0012] BaitNET is capable of running on "bare metal" machines however due to its nature in testing potentially malicious URLs, a virtualized environment is preferred such as that when a Victim is comprised, it can be automatically reset to a clean state. BaitNET is not limited to a specific hypervisor thanks to its modular design. Its default virtual adapter is for VMware ESXi but it was originally designed to work with Microsoft Hyper-V. Additions can be made in the form of additional adapters, which would allow it to run on any hypervisor, thus supporting multiple hypervisor communication channels, possibly within the same deployment if needed.

[0013] BaitNET provides a malware and exploit campaign detection system and method that cannot be detected by the malware or exploit campaign. The system may provide threat feed data to the vendors that produce in-line network security and endpoint protection technologies. The system may also be used as a testing platform for 3rd party products. Due to the massive footprint of the system's cloud infrastructure and disparate network connections and geolocation obfuscation techniques, NSS can locate and monitor malware across the globe and provide detailed threat analysis for each specific region, as they often support and host different malware/cybercrime campaigns.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] FIG. **1** shows the high-level architecture of the major components of a system for malware and exploit campaign detection that may be known as BaitNET.

[0015] FIGS. **2A-2D** illustrates the process control and internal operations of the BaitNET Controller Process and its interoperability with the Capture, Replay, and Proxy processes.

[0016] FIGS. **3-6** are examples of the user interface of the BaitNET system of FIG. **1**.

DETAILED DESCRIPTION OF ONE OR MORE EMBODIMENTS AND IMPLEMENTATIONS OF THE SYSTEM AND METHOD

[0017] The system and method for malware and exploit campaign detection (known as BaitNET) is designed to seek out, detect, itemize, and retest active URLs serving drive-by exploits. BaitNET is a multi-leveled application operating within the kernel and user layers of the operating system that make it unique when compared to similar technologies utilized to detect and prevent malware.

[0018] Note that the distinction is important—malware is the payload that is delivered by an exploit. There are literally hundreds of thousands of malware samples in the wild, and it is a trivial matter to obfuscate these or morph them into something new. In contrast, there are only a few hundred active exploits in the wild at any given point in time—the exploit is the mechanism whereby the threat actor compromises the system in order to deliver and execute the malware. By identifying and blocking exploits, BaitNET moves further up the kill chain from traditional malware protection products and provides much more effective and far-reaching predictive capabilities.

[0019] BaitNET's core technology could be used to prevent exploitation as well as detect, but that is not its primary function. By operating out of band, BaitNET is freed from the usual restrictions of real-time or in-line protection technologies, allowing it to be much more accurate and thorough in its detection capabilities. BaitNET supports various types of operating systems as a threat forecast system. BaitNET's virtual machines (VMs) can simulate servers, workstations, even mobile computing devices such as smartphones and tablets.

[0020] As shown in FIG. **1**, a system **100** may utilize three arrays of servers and networking hardware known as "stacks." Each stack is any number of physical servers that host virtual machines ("guests".) The exact number of servers and guests is based on the scope and scale of the testing and research being performed. Typically, within "Live Testing" this will be many tens of thousands of guests. FIG. **1** illustrates the interoperation/communication of the

various stacks of servers and guests with the infrastructure support servers, as well as which components have Internet **102** connectivity.

[0021] Specifically, the system may be implemented using the computing resources shown in FIG. **1** including the stacks. As shown in FIG. **1**, the system may be implemented with a capture stack **104**, a replay stack **106**, a proxy stack **108**. The system may also have a master hypervisor controller **110** that controls each of the stacks as well as one or more data stores **112** (for storage of data and the like). As shown in FIG. **1**, the capture stack **104** and the proxy stack **108** have access to a computer network **102**, such as the Internet. The capture stack **104** implements the capture process **204** described below, the replay stack **106** implements the replay process **206** described below and the proxy stack **108** implements the proxy process **208** described below. Each of the stacks may be implemented using one or more computing resources, such as one or more cloud computing resources or one or more server computer resources. Each of the one or more computing resources may have a processor and memory and a plurality of lines of computer code that may be stored in the memory and executed by the processor to implement the capture, replay and proxy processes described below. Each of the stacks also may be implemented as one or more virtual machines that are controlled by the hypervisor controller **110**.

[0022] FIGS. **2**A-**2**D illustrates the process control and internal operations of the BaitNET Controller Process (implemented by the master hypervisor controller **110**) and its interoperability with the Capture, Replay, and Proxy processes. Each of the processes shown in FIGS. **2**A-**2**D may be implemented as a module/unit/device that is part of the respective stacks shown in FIG. **1** and each process may be implemented in software (a plurality of lines of instructions/computer code executed using a processor) or as a hardware device that is part of the respective stack shown in FIG. **1**. A controller process **210** and part of a replay process **206** are shown in FIG. **2**A with the replay process **206** also being shown in FIGS. **2**B and **2**C as shown by the reference designators (A and E) that show how FIGS. **2**A, **2**B and **2**C connect to each other to show the replay process **206**. FIGS. **2**B and **2**C also show the details of the capture process **204** as shown by the reference designator D that shows how FIGS. **2**B and **2**C connect to each other to show the capture process **204**. FIGS. **2**B and **2**C also show a ZeroDay process **220** as shown by the reference designator B that shows how FIGS. **2**B and **2**C connect to each other to show the ZeroDay process **220**. FIGS. **2**B and **2**C also show the proxy process **208** as shown by the reference designator C that shows how FIGS. **2**B and **2**C connect to each other to show the proxy process **208**. Finally, FIG. **2**D shows the details of an Obfuscation Engine **222** with references F and G showing the interchange between the capture process **204** and the Obfuscation Engine **222**. Also illustrated are the interchanges with the Obfuscation Engine, Exploit Feed, and ZeroDAY modules. FIGS. **2**A-**2**C show an enumeration process as shown by the reference designators (I and H) that show how FIGS. **2**A, **2**B and **2**C connect to each other to show the enumeration process.

[0023] The system and method for malware and exploit campaign detection shown in FIGS. **1**-**2**D and described above may be typically operated by an entity, such as NSS Labs ("NSS"), as a cloud-based system that is used by the entity to perform the malware and exploit campaign detection as shown in FIG. **1**. However, the system and method for malware and exploit campaign detection may also be installed on a premises of a customer and perform the same malware and exploit campaign detection. Using sources from around the globe, BaitNET's process begins with the correlation and normalization of multiple threat feeds for information regarding potentially malicious websites. This normalized data is presented to BaitNET's Capture Process **204** and is queued as targets for each of the configured operating system variations that are assigned to series of testing.

[0024] BaitNET, using the Capture Process **204**, issues the URL to each of the thousands of Victims, utilizing thousands of variations in configurations, and each Victim in turn visits the URL using disparate VPN tunnels, upstream HTTP proxies, and even physical data centers, located around the world, to obfuscate their true geographical location as well as to explore the geolocation filtering that may be employed by the malicious URL.

[0025] If successful, a visit to the URL will result in exploitation of the Victim (the "exploit") sometimes (but not always) followed by a "drop" of malicious code (the "malware") to the target workstation. BaitNET monitors the progress of the exploit and records the network traffic, creates a copy of the malicious code, and catalogs all changes to the operating system made by the malicious code during the capture process **204**. Any malware dropped on the Victim is also captured, as are the effects of executing that malware on the Victim.

[0026] Note that one of the unique features of BaitNET is that, unlike traditional anti-malware security solutions, even if no malware is dropped, the exploit is still detected and captured. Additionally, the Capture Process **204** may record any and all outbound communications from the now infected/compromised Victim. This outbound traffic will include any command and control (C&C) communications, often identifying the true threat actor, as well as any data being exfiltrated from the now infected Victim.

[0027] Validation of the recorded data occurs by analyzing the events that were generated on the Victim. Note that this process occurs in seconds, not minutes, hours, or even days. This is possible because BaitNET analyzes the contextual relationship between the events that were generated to confirm infection. Furthermore, as a threat forecast system, the longer BaitNET is online and the more data it gathers, the more efficient its analysis becomes.

[0028] When infection is confirmed, BaitNET provides information such as the URL where the attack originated, the type of URI/attack used, the IP address of the server that hosted the malicious URL, and the country of origin of the IP address (aka "geolocation."). For example, FIG. **6** shows detailed information on the URI and network behavior of the malicious website when accessed by the guest systems inside of BaitNET Further detail is presented on the operable target platform(s) that were successfully infected with the malicious content. The hashes (MD5) of the malicious executables (files) along with the exact size of each file are also made available. The network packet capture (PCAP) data as well as the decoded HTTP traffic that is relevant to understanding the attack vector is also available. Provided are the full URI, protocol of the attack (i.e. HTTP/1.1) the specific web browser used (i.e. Internet Explorer 11), the actual URL of the drop (i.e. .DE, Germany, domain), as well as information about the server such as the web server

operating platform (i.e. Apache 2.0.59 running on a UNIX operating system with Open SSL). This information can be used by end-users to write firewall rules as well as other rules within in-line systems such as IPS, IDS, WAF, and NGFW. The information can also be used to update endpoint products such as anti-virus to now identify the hash values of the now known malicious content and block it from either being downloaded or executed. The exact vector of the attack being provided; which includes hosting, transmission, and target configuration are vital pieces of information that are uniquely provided by BaitNET.

[0029] The Victim that was successfully infected is now reset to its virgin state, thus preparing it to be reused for the next URL in the queue. All the data collected is stored in a data store used for logging and intelligence. BaitNET is modular enough to support any number of different storage technologies, including everything from traditional relational databases, NoSQL databases, and even graph-based databases.

[0030] The infected URL is now queued up for the Replay Process **206** using data from the data store. During the Replay Process **206**, Victims matching the configuration of the Victim that successfully was infected during the Capture Process **204** are prepared for testing of the malicious code. To prepare the Victims, all recent versions of products being tested (in-line security devices to endpoint protection products) are automatically configured. Copies of the workstation used during the Capture Process **204** are configured with the latest versions of any and all endpoint protection products being tested. In-line security products such as intrusion prevention systems and next generation firewalls stand in wait on the network between the Victims and the Replay Servers. The Victims visit an internal (LAN-based) URL that has been created by BaitNET as a perfect copy of the malicious URL that was validated during the Capture Process. As each copy of the Victim is presented the internal URL, BaitNET once again monitors the Victim capturing all metadata related to the malicious code. If the code is successful in reaching the Victim and then executing properly, the endpoint protection product being tested has failed to identify and/or stop the malicious code. If, during the visit to the internal URL, the drop is prevented, thus malicious code is prevented from ever reaching the Victim, then the in-line security product was successful in identifying the exploit and worked as designed.

[0031] During the Replay Process **206**, the effectiveness of the malicious code is tested in a live environment. For example, all major makes and versions of web browsers are tested to determine which are susceptible to the exploitation during a drive-by-exploitation attack (i.e. an attack that executes within the browser and does not require the end-user to manually execute the malicious code). Different versions of application systems, language packs (localization data), base operating system revision, and even different architectures can be checked against the copy of the malicious URL and the malicious code itself.

[0032] During the Replay Process **206**, an emulated HTTP proxy is generated and utilized. This HTTP proxy facilitates the ability of BaitNET to perform continual testing against the malicious URL that was collected during the Capture Process without the need of the original/actual malicious URL. This is important due to the short lifespan of most malware campaigns, security features within the malware campaign to identify and prevent drops of malicious code to systems on the same network, and to obfuscate/protect the research and investigation into the malware campaign. The HTTP proxy uses the original source code of the malicious website as recorded by the Capture Process **204**. The HTTP proxy emulates the remote server, source code of the website, and will serve (hand-out) the malware in the same way that the original website did. Note that an HTTP proxy, as a technology, is not in itself unique. The use of an HTTP proxy that is generated dynamically to emulate recorded network traffic in order to test the effectiveness of security products without depending on a live malware campaign and mimicking real live network traffic, as opposed to a dummy network traffic generated by network penetration tools, is the unique advantage provided by BaitNET.

[0033] The Capture **204** and Replay **206** Processes can be used to continue to check localization configurations and geolocation exit points on the Internet to determine the full scope of the attack vector, provide intelligence on the threat actor(s), and harvest as much viable metadata as possible. These processes are key in enumerating the various configurations of operating system, browser, applications, security products, etc. that the malware can use to successfully execute itself. The collation of this intelligence allows modeling to be performed, as well as direct risk assessments, so that consumers understand if their systems, networks, and tools are at risk—and what to do, if anything, to protect them against active exploit/malware campaigns.

[0034] All data are retained in the data stores and can be reused by BaitNET at any time. New Victim configurations can be presented to the captured malicious URL for future testing. All tested products can be retested to confirm that patches/updates supplied by the vendor are working as designed, to outline exactly which systems provided by 3rd parties are susceptible to the attack ("Gold Images"), and to validate attack data captured during the Capture Process.

[0035] The system and method shown in FIGS. **1**-**2**D may define a messaging protocol, dubbed as Horus, that is an application level protocol and consists of a set of rules for messages that are exchanged between a Controller **110**, a set of Victims, and a set of Notification Sinks. Each message has a special sematic meaning and is meant to provoke a certain behavior. Horus consists of two sub-protocols: Horus/Victim which defines how a Controller communicates with Victims and Horus/Notification which defines how a Controller and Victims communicate with Notification Sinks.

[0036] Horus/Victim is a synchronous and stateful request-reply protocol, more commonly referred to as an RPC protocol, for two endpoints, a client and a service, to communicate with one another. The client sends a request. The service reads the request and sends a response. The client reads the response. Both the client and the service are responsible for maintaining state for the duration of a session. In BaitNET, the client is a Controller and the services are the Victims.

[0037] Horus/Notification is an asynchronous and stateless publisher-subscriber protocol for one endpoint, a publisher, to communicate with a set of endpoints that is of an undefined size, the subscribers. Subscribers register with the publisher their interest in receiving messages. The publisher broadcasts messages to the subscribers. Subscribers receive all messages broadcast by the publisher that they are interested in. The publisher expects no response from the subscribers. In Horus/Notification, the subscribers are the Notification Sinks. Both a Controller and Victims in different parts of Horus' topology fulfill the role of the publisher.

[0038] A Controller is a producer, or ventilator, that MUST produce URLs to be distributed to interested Victims. A Controller is also a message pump that MUST collect

results that are produced by Victims and MUST publish them to interested Notification Sinks. We refrain from using the term "broker" to describe a Controller in its role as a message pump even though its purpose seems analogous to one. Traditional middleware brokers are too complex, too stateful, complicate an application's deployment model, and are usually meant to serve as a shared entity between many different disparate systems. The Controller as a message pump thus acts more like an intermediary to push data downstream to interested Notification Sinks, like a switch, to avoid a mesh topology between them and Victims.

[0039] A Victim is a consumer, or worker, that MUST consume URLs and process them. A Victim MUST either publish its results to a Controller or it MUST maintain its results as state until a Controller explicitly polls it for them, depending on the topology deployed. In either case, the Controller MUST always forward the results downstream to interested Notification Sinks. A Notification Sink is a collector, or subscriber, that SHOULD collect results produced by Victims. A Notification Sink SHOULD register with a Controller its interest in receiving results produced by Victims. In Horus, Notification Sink interest is referred to as a Subscription.

[0040] Victims are discovered using a method we refer to as hybrid discovery. Hybrid discovery is a mix between traditional static and dynamic discovery methods.

[0041] Static discovery refers to Victims being known beforehand. This is analogous to a having a configuration data store of some sort, such as a configuration file, a configuration database, or even a hard coded in-memory collection, which contains relevant information about available Victims within a network. This form of discovery is relatively easy to implement but obviously requires Victims to be deployed manually beforehand.

[0042] Dynamic discovery refers to Victims dynamically being deployed and providing a notification to a beacon that they are available. This form of discovery is incredibly difficult to implement but offers the most flexibility for certain use cases. In BaitNET however, Victims are usually deployed in a virtualized environment so that they can be reset to a clean state after they process a URL. BaitNET recognizes the importance of running in a virtualized environment for that very reason and thus has first class support for it. In a virtualized environment, virtualized Victims can be deployed, and in some advanced uses cases even provisioned, dynamically. Because complete control over the environment is possible, Victims do not need to notify a beacon on when they are available. Thus, we refer to this mode of discovery as hybrid discovery: its static in the sense that the virtualized environment is known before hand and dynamic in the sense that Victims can be deployed or provisioned dynamically in that environment.

[0043] Victims are implemented as finite state machines, with each state representing a step in its progress in processing a URL. This allows the Controller, with high level of accuracy and without the dependency on third party middleware products, to track the Victims and to distribute URLs to them as they become available. The different states are:

[0044] Available—Indicates the Victim is available and ready to process a URL

[0045] Booting—Indicates the Victim has received a URL from a Controller and is in the process of booting

[0046] Acquired—Indicates the Victim has successfully launched a browser and navigated to the URL

[0047] Completed—Indicates the Victim has successfully completed monitoring the system and collected relevant data

[0048] Error—Indicates the Victim has encountered an error and might need to be reset

[0049] FIG. **3** is an output from the system, which illustrates validated exploits that have been discovered by the BaitNET system. The capture date, e.g. the date and time the malware or exploit was downloaded, is shown along with the corresponding source URL (Universal Record Locator) which shows the full path to the file on the infected/malicious website, the exact operating system that was used on the guest (virtual) workstation that the malware/exploit executed upon, and the exact application that the malware/exploit targeted (needed to be successful.) In this example, the first exploit in the list uses Java version 6 update 27 on Microsoft Windows 7 and was downloaded from a URL which was redirected (linked) on a google.com website. A user of the system can click any of these fields to drill-down into more detailed information. E.g., The "Source" section provides IP addresses, packet capture data, geo-location information, etc.

[0050] FIG. **4** is output from the system which illustrates detailed information on the "drop" or malicious file that was downloaded and has been validated to be malware/exploit code. Again the pertinent date and time is displayed, a unique filename is presented which was generated by the system when the malware/exploit was captured. This file contains the malicious content and can been downloaded in its archived (safe) version for inspection and reverse engineering. The hash value (MD5) of the archived file is presented so that the end-user can validate the file from the repository has not been altered. The system will indicate, as presented in this example, that the malware/exploit has been validated. Validation occurs when the BaitNET system utilizes the Proxy and Replay Processes to confirm infection and execution of the captured malware/exploit. The center section of the page reflects the URI where the file was collected from (This matches the data on FIG. **3**) the type of URI/attack used, the IP address of the server that hosted the malicious file, and the country of origin of the IP address (aka "geo-location.") Further detail is presented on the operable target platform(s) that were successfully infected with the malicious content.

[0051] FIG. **5** is more detailed information from the system with regard to malicious content (malware/exploit code) that was captured. Here the end-user can find the hash (MD5) of the malicious executables (files) along with the exact size of each file. This information can be used to update in-line security systems such as an IPS, NGFW, or even endpoint products such as anti virus to now identify the hash values of the now known malicious content and block it from either being downloaded (in-line devices) or executed (end point products.)

[0052] For Threat Forecasting, the Enumeration Process/scout algorithm can be used to continue to check localization configurations and geolocation exit points on the Internet to determine the full scope of the attack vector, provide intelligence on the threat actor(s), and harvest as much viable metadata as possible. This process is key in enumerating the various configurations of operating system, browser, applications, etc. that the malware can use to successfully execute itself. The collation of this intelligence allows modeling to

be performed, as well as direct risk assessments, so that consumers understand if their systems, networks, and tools are at risk—and what to do, if anything, to protect them against active malware/exploit campaigns.

[0053] In one implementation, the entire BaitNET suite of processes may take place in parallel, currently utilizing four parallel threads that are responsible for managing each of the aforementioned processes (Capture, Replay, and Proxy) along with their sub-processes such as the Obfuscation Engine shown in FIG. **2**D and modules covered within this document such as ZeroDAY and are collectively controlled from the Control Process. Additionally monitoring of the virtual machines (VMs) and the setup and tear down (establishment and reverting) of the VMs along with their guest operating system and application configurations take place from within the Controller Process.

[0054] Full control of the VM architecture is done through BaitNET's Controller Process (implemented by the Master Hypervisor Controller in FIG. **1**), which is modified to operate natively. This control is automated and functions as a separate thread during the Control Process and works in parallel with the Capture, Replay, and Proxy processes. BaitNET can procure, configure, and operate VMs on demand, autonomously, and scale resources during testing.

[0055] Additional cloaking technologies that prevent detection of BaitNET are covered herein within the model overviews.

[0056] As outlined herein, BaitNET is a system of custom developed applications, application program interfaces (APIs), and kernel-level modification, such as the AT Module of the system, the Obfuscation Module of the system, the ZeroDAY module of the system and the capture process, for example which are applications. The applications are for both the hypervisor host and the guest functionality as well as the operating systems. BaitNET currently supports all versions of Microsoft Windows operating system, all Intel-based versions of OS X, iOS, and Android. One key feature for BaitNET is its ability to render the "VM Detection System" (e.g. ability to discern a virtual machine from a physical/real machine by malware) found in modern malware/exploits useless. This thwarts the ability of malware to detect a VM, which would normally prevent it from deploying its payload as VMs are often used in anti virus systems to incubate suspected executable files.

[0057] BaitNET's functions are expanded and complimented through the use of modular components (Modules) described below in more detail, each of which provides functionality used in threat forecasting and the evaluation of 3^{rd} party security product effectiveness as shown in FIG. **2**.

[0058] Capture Process **204**

[0059] BaitNET's Capture Process **204** may be implemented using a scout process (that may be implemented as an algorithm when this process is implemented in software.) The scout process may also be referred to as an enumeration process. Like classical battle strategies in which a small scout party is detached from the main fighting force and sent out to gather intelligence about the enemy fighting force and the intelligence gathered helps in formulating a strategy for winning the battle, the scout process is designed to seek out, by testing, as many URLs as possible to determine if they are malicious. The intelligence gathered is thus which URLs are worth spending precious computing resources against to determine the capabilities of the threat actors.

[0060] URLs are gathered from a variety of different sources from around the globe. The system correlates and normalizes URLs from multiple threat feeds for information regarding potentially malicious websites. These URLs are then queued up for testing.

[0061] The Scout process may define what may be referred to as Tiers. Tiers are sets of victims that are configured to test URLs using an operating system, browser, and application(s) combinations. Each victim may be a virtual machine having an operating system, browser, and one or more application(s) configuration with different strategies against which an exploit may be tested to determine if the URL is malicious with respect to each particular configuration of each victim. The Scout process may define three Tiers, each representing a different strategy.

[0062] Tier 1 defines a set of victims that have a combination of operating systems, browsers, and applications that are highly targeted by exploit kits. More than one application can be installed on the victim but only one operating system and one browser can be installed. The configured combination is reinforced through ongoing research of the threat landscape and will change when the thread landscape changes.

[0063] The purpose of Tier 1 is to test as many URLs as quickly as possible and determine if they are malicious. Identifying the full capabilities of the threat actor on Tier 1 is not important. Malicious URLs are normally live, that is to say they are either infected or publicly accessible, for a short amount of time. And they usually represent a small percentage of the overall number of URLs that will be tested. It is thus imperative that they are tested as quickly as possible. Multiple applications are usually installed on Tier 1 Victims, though that is not absolutely necessary, to maximize the possibility of a drive-by-exploitation attack taking place. The URL distribution algorithm on Tier 1 is a simple load balancing or round-robin algorithm. Essentially the URLs that are queued up for testing are distributed to each available Tier 1 Victim, in parallel. As each Tier 1 Victim completes testing one URL (by accessing the URL to determine if the URL is malicious), it is assigned the next URL in the queue until the queue is exhausted. Once the queue is exhausted, the Tier 1 Victims remain idle until more URLs are queued up. The more Tier 1 Victims that are available the more URLs that can be tested. When a URL is found to be malicious by a Tier 1 Victim, it is queued up for further testing on Tier 2.

[0064] Similar to Tier 1, Tier 2 defines a set of victims that have a combination of operating systems, browsers, and applications that are also highly targeted by exploit kits. Unlike Tier 1 however, only one application can be installed on the Victim. The configured combination is reinforced through ongoing research of the threat landscape and will change when the thread landscape changes. The Tier 2 combination of operating systems, browsers, and applications is a superset of the Tier 1 combination.

[0065] Drive-by-exploitation exploit kits usually fingerprint the Victim when a malicious URL is tested. Based on the configuration of the Victim, a different exploit might be served. Consider, for example, a Tier 1 Victim that has both Microsoft Silverlight and Adobe Flash Player installed. When such a Victim tests a malicious URL, the exploit kit might fingerprint the Victim and determined that both Microsoft Silverlight and Adobe Flash Player are installed. It's entirely possible that the exploit kit supports both

Microsoft Silverlight and Adobe Flash Player. However, randomly at runtime, the malicious URL might decide that it will only serve an exploit targeting one of the applications.

[0066] This is where the benefit of Tier 2 comes in, and the primarily difference between it and Tier 1. On Tier 1, the URL was identified as malicious but there is no strong indication on the full extent of the capabilities of the threat actor. When the URL is queued up on Tier 2, two Victims, one with Microsoft Silverlight only and one Adobe Flash Player only, will be instructed to test the URL. Now, if the exploit kit supports both applications, testing the URL again on Tier 2 will derive a possible addition of two more exploits, bringing the total number of exploits potentially being served by a single URL to three. The total number of exploits is three because the operating system and the browser are also considered as attack vectors in addition to the two installed applications. This is also why it is imperative that, similar to Tier 1, the URL must be live when it is tested on Tier 2.

[0067] Of course, it is not only applications that are tested on Tier 2 but also different operating systems and browsers. The below matrix is an example of the possible different combinations a URL can be tested against if it is found to be malicious on Tier 1 and queued up on Tier 2:

[0068] This matrix is just a small example of the many combinations that can be tested and does not even include different mainstream browsers like Google Chrome and Mozilla Firefox. The large number of possible combinations that need to be tested is the primary reason for having different Tiers in the scout process.

[0069] Ongoing research has suggested for quite some time now that only about 10% of URLs are malicious at any given point in time. BaitNET's design takes into consideration that computing resources are precious and thus does not attempt to test a URL using every possible combination simply to determine if it is malicious. Instead, on Tier 1 it simply identifies the 10% that are relevant and throws away the remaining 90%. It is only those 10% that are tested against the remaining combinations. This means not only massive savings in time but also in the costs associated in running BaitNET. It is also a precursor for Tier 3.

[0070] Tier 3 defines a set of Victims that have the same combination of operating systems and browsers as Tier 2 but with different versions of the applications. For example, if Tier 2 is configured to test Microsoft Silverlight 1, and there are ten versions of Microsoft Silverlight released, Tier 3 will have Victims with the remaining Microsoft Silverlight 2 through Microsoft Silverlight 10.

Tier 1			Tier 2		
Operating System	Browser	Applications	Operating System	Browser	Application
Windows 7	Internet Explorer 9	Adobe Flash Player, Microsoft Silverlight	Windows XP	Internet Explorer 8	N/A
			Windows 7	Internet Explorer 9	N/A
			Windows 7	Internet Explorer 9	Adobe Flash Player
			Windows 7	Internet Explorer 9	Microsoft Silverlight
			Windows 7	Internet Explorer 10	N/A
			Windows 7	Internet Explorer 10	Adobe Flash Player
			Windows 7	Internet Explorer 10	Microsoft Silverlight
			Windows 7	Internet Explorer 11	N/A
			Windows 7	Internet Explorer 11	Adobe Flash Player
			Windows 7	Internet Explorer 11	Microsoft Silverlight
			Windows 8	Internet Explorer 10	N/A
			Windows 8	Internet Explorer 10	Adobe Flash Player
			Windows 8	Internet Explorer 10	Microsoft Silverlight
			Windows 8.1	Internet Explorer 11	N/A
			Windows 8.1	Internet Explorer 11	Adobe Flash Player
			Windows 8.1	Internet Explorer 11	Microsoft Silverlight
			Windows 10	Microsoft Edge	N/A
			Windows 10	Internet Explorer 11	N/A
			Windows 10	Internet Explorer 11	Adobe Flash Player
			Windows 10	Internet Explorer 11	Microsoft Silverlight

[0071] Normally, a single exploit will take advantage of a vulnerability that impacts multiple different versions of an application. It's important to note however that for Tier 3, the Victim does not need to test the URL while it is live. Once a session of the attack has been recorded on either Tier 1 or Tier 2, it can be tested against multiple different version of the same application. Since the attack has been recorded and it impacts multiple versions, Tier 3 does not need as much computing resources as Tiers 1 and 2 since there is no longer a requirement to test the URL as quickly as possible.

[0072] The enormous size of BaitNET's cloud infrastructure requires that its design take into consideration the fact that computing resources are both precious and costly. With the massive number of combinations that a single URL needs to be tested against across all three Tiers, it is simply not realistic to require a single Victim per combination. Even if that was the case, as the number of URLs that need to be tested grows, a single Victim will not be able to test them all fast enough. Recall, that on Tier and Tier 2, a URL must be tested as quickly as possible while it is live. For this reason, Victims can be provisioned dynamically, based on the target number of URLs that need to be tested in a time period. Similarly, browsers and applications can be installed dynamically on the victims such that there doesn't need to be one dedicated Victim for each combination.

[0073] The success of BaitNET's Scout process is evident by the large number of URLs that can be processed in a 24-hour period using relatively little computing resources. With a mere 400 Victims, BaitNET can successfully process in excess of 250,000 URLs in a 24-hour period which is absolutely phenomenal in comparison to other threat forecasting systems.

[0074] ZeroDAY Module/Process **220**

[0075] This module **220** is a state of the art plugin for the BaitNET system allowing it to detect any type of exploitation attack, and was developed to identify 0day attacks, e.g., exploits and malware that have yet to be categorized or identified within the security community, often meaning there is no currently known defense to these attacks as the maintainers of the commercial or open source products being targeted are themselves unaware of the flaw being exploited.

[0076] This module **220** is capable of dissecting the attack and recording the smallest components, uncovering how every intricate step and security mitigation tactic was used to achieve the attack. This module is based on unique knowledge that the owner of BaitNET has developed through various research projects.

[0077] The ZeroDAY module **220** is effective when presented with the most complex and customized/never before seen malware as used in advanced persistent threat (APT) attacks. The module can be set to detect and catalog the attack, or detect and block the attack. Unlike EMET, Microsoft's current security mitigation technology, the ZeroDAY module utilizes the combined filtration of KERNEL32, KERNELBASE and NTDLL.

[0078] The ZeroDAY module may perform any or all of the following industry recognized tasks for recognition and cataloging of exploits:

[0079] In-memory shellcode detection

[0080] Raw shellcode dumping (raw output of shell code to file)

[0081] Raw shellcode disassembly (post analysis)

[0082] Shellcode emulation

[0083] Identify APIs used in the shellcode

[0084] Log API parameter information:

- **[0085]** Network
- **[0086]** Memory
- **[0087]** File
- **[0088]** Process

[0089] ROP detection

[0090] ROP gadgets detection

[0091] ROP gadgets dumping with backward disassembly (module+function)

[0092] Heap spray detection

[0093] NOP sled detection

[0094] NULL page allocation detection

[0095] In general, the ZeroDAY module can monitor and protect any application in user-land (ring-three e.g., "r3"), but can only monitor and not protect against kernel-land (ring-zero e.g., "r0") exploits affecting the operating system services directly. It will still, however, be able to protect against kernel-based exploits being served through user-land and any other application that utilizes this attack vector.

[0096] The ZeroDAY module provides stack validation and monitoring that includes the protection from direct access to KERNEL32, KERNELBASE and NTDLL APIs. The module may also have a CODE/TEXT section permission change monitor. This monitor is a novel process/mechanism. This mechanism allows the detection and monitoring of privilege escalation through a process whereby the system monitors for CODE/TEXT changes. This is possible due to the way that the ZeroDAY module integrates into the kernel and ties directly into primary system sub processes.

[0097] A semi control-flow-transfer (CFT) check is part of the system and all system calls (r3) will still tunnel back to the original one in the kernel (r0). Therefore, calls will be filtered through KiFastSystemCall [SystemCallStub] (triggered by interrupt vector int 0x2E).

[0098] The ZeroDAY module was designed to not only to detect and stop the attack, but also to gather information post the attack. This information may include communication with a command and control (C&C) server and the downloading of malware. It serves well to automate the detection, post-automated analysis of the attack and gathering in-depth information for data analysis (i.e. briefs and blog posts) that other individuals or companies do not have.

VM+SandBox Detection Avoidance and Circumvention Module

[0099] Almost all malware detects the presence of/if it is hosted by an operating system managed as a virtual machine (VM.) aka "SandBox" and will avoid execution and revealing their control-flow (CF) to be dynamically analyzed. This was developed to circumvent this anti-detection capability in modern malware; it will detect whether the dropped malware is a result of an exploit or was simply the result of typical drive-by's that attempt to avoid execution within a VM or a Sandbox. There are multiple options for circumvention of the anti-detection technology within malware:

[0100] 1) Direct in-memory patching based on signatures developed in the lab using advanced regular expressions and Boolean algebra

[0101] 2) Hijacking the system calls made by the malware through a proxy stub, trampolining the original code with the new one and feeding the malware the wrong results tricking it to run as expected on a bare-metal machine.

AI Module

[0102] This module is responsible in generating artificial human activity in the VM. As some malware will check for the lack of mouse activity or keyboard activity or even processes being spawned. The absence of activity from these human interface devices, along with the absence of ancillary processes and applications are indicative of a automated machine, and therefore a trap. The AT Module corrects this oversight in other incubation systems by injecting randomized mouse movements and usage, keyboard input to include realistic typing patterns, mistakes variations in speed, etc. All of this produces a very realistic usage of the machine.

[0103] The AT module and Sandbox modules may be part of the system (like the ZeroDay module) in FIGS. **1** and **2**, but is not shown in these figures.

VM Templates

[0104] All the VM images being used across the stacks are created from custom made templates, which use the underpinning of the Control Process, which integrates the virtual machine controls into the base operating system, thus hiding the Guest OSes and appearing like a normal bare-metal machine. This includes options such as:

[0105] Getting the PTR location

[0106] Setting the PTR location

[0107] Direct Exec

[0108] NT Reloc

[0109] Self Modification

[0110] Reloc

[0111] BT Segment

[0112] BT Privilege

[0113] BT Mem Space

[0114] BT IN Port

[0115] BT Out Port

[0116] The system and method for malware and exploit campaign detection is a technical solution to a technical problem that did not exist prior to the Internet and computer networks. Specifically, the technical problem is trying to detect malware and exploit campaigns on a computer and computer networks. This technical problem did not exist prior to computer networks and the Internet. The system and method for malware and exploit campaign detection addresses this problem as disclosed using the capture stack that is configured to issue a uniform resource locator to each computer system to download a piece of malicious code, the replay stack that is configured to test the piece of malicious code in a live environment and generate data about the replay of the piece of malicious code, the proxy stack that is configured to perform testing of the piece of malicious code without accessing the uniform resource locator and the master hypervisor controller that controls the capture stack, the replay stack and the proxy stack. Furthermore, the system and method for malware and exploit campaign detection overcomes the limitations of the conventional systems and methods as described above.

[0117] The system and method for malware and exploit campaign detection use rules and the capture stack, the replay stack and the proxy stack (and their corresponding processes) to perform the malware and exploit campaign detection. Furthermore, the system and method provide an improved technical result for malware and exploit campaign detection using the capture stack, the replay stack and the proxy stack (and their corresponding processes) which are an advance and are an inventive concept over the conventional system as described above.

1. A malware and exploit campaign detection system, comprising:

a plurality of computer systems;

a capture stack, implemented on the computer system, that is configured to identify a plurality of malicious uniform resource locators that each have a piece of malicious code;

a replay stack, implemented on the computer systems, that is configured to test each piece of malicious code from the capture stack in a live environment using a victim by accessing the malicious uniform resource locator and to generate data about the replay of each piece of malicious code, each victim having a configuration of an operating system, a browser and at least one application that is exploitable by an exploit; and

wherein the capture stack has a scout process that gathers the plurality of malicious uniform resource locators and that sends each malicious uniform resource locator to a particular victim of the replay stack.

2. The system of claim **1**, wherein the scout process defines a first tier comprising a set of victims of the replay stack with each victim having a combination of an operating system, a browser and one or more applications that are targeted by an exploit, each first tier victim testing the uniform resource locators assigned to that first tier victim to identify a plurality of first level malicious uniform resource locators, wherein each first level malicious uniform resource locator exploits the combination of the operating system, the browser and the one or more applications on the first tier victims.

3. The system of claim **2**, wherein the scout process defines a second tier comprising a set of victims of the replay stack with each victim having a combination of an operating system, a browser and one application that are targeted by an exploit, each second tier victim testing a first level malicious uniform resource locator identified by the first tier to identify a plurality of second level malicious uniform resource locators from the first level malicious uniform resource locators, wherein each second level malicious uniform resource locator exploits the one application.

4. The system of claim **3**, wherein the scout process defines a third tier comprising a set of victims of the replay stack with each victim having a combination of the same operating system and browser as the second tier victim that identified the second level malicious uniform resource locator and a different version of the application of the second tier victim, each third tier victim testing a second level malicious uniform resource locator identified by the second tier to identify a plurality of third level malicious uniform resource locators from the second level malicious uniform resource locators wherein each third level malicious uniform resource locator exploits the different version of the application.

5. The system of claim **1** further comprising a proxy stack that is configured to perform testing of the piece of malicious code without accessing the uniform resource locator and a master hypervisor controller that controls the capture stack, the replay stack and the proxy stack.

6. The system of claim **5**, wherein the capture stack, the replay stack and the proxy stack run in parallel.

7. The system of claim **5** further comprising a zero day module that identifies zero day attacks.

8. The system of claim **1**, wherein the capture stack is configured to create a copy of the piece of malicious code and catalogs operating system changes caused by the piece of malicious code.

9. The system of claim **1**, wherein the capture stack is configured to capture communications with the plurality of computer systems.

10. The system of claim **5**, wherein each stack is one or more server computers.

11. The system of claim **10**, wherein each stack has a virtual machine.

12. A method for malware and exploit campaign detection, comprising:

identifying a plurality of malicious uniform resource locators wherein each malicious uniform resource locator contains a piece of malicious code;

sending each of the plurality of malicious uniform resource locator to each of a plurality of victims, each victim having a configuration with an operating system, a browser and at least one application that are exploitable by an exploit of a malicious uniform resource locator;

testing, at each victim, each of the plurality of malicious uniform resource locators in a live environment; and

generating data about the replay of the malicious uniform resource locator and the piece of malicious code.

13. The method of claim **12**, wherein testing the plurality of uniform resource locators further comprises accessing each uniform resource locator using a first tier victim that has a combination of an operating system, a browser and one or more applications that are targeted by an exploit and identifying a plurality of first level malicious uniform resource locators, wherein each first level malicious uniform resource locator exploits the combination of the operating system, the browser and the one or more applications on the first tier victims.

14. The method of claim **13**, wherein testing the plurality of uniform resource locators further comprises accessing each uniform resource locator using a second tier victim that has a combination of an operating system, a browser and one application that are targeted by an exploit and identifying a plurality of second level malicious uniform resource locators from the first level malicious uniform resource locators, wherein each second level malicious uniform resource locator exploits the one application of the second tier victim.

15. The method of claim **14**, wherein testing the plurality of uniform resource locators further comprises accessing each uniform resource locator using a third tier victim that has a combination of the same operating system and browser as the second tier victims and a different version of the application of the second tier victim and identifying a plurality of third level malicious uniform resource locators from the second level malicious uniform resource locators wherein each third level malicious uniform resource locator exploits the different version of the application of the third tier victim.

16. The method of claim **12** further comprising performing testing of the piece of malicious code without accessing the uniform resource locator.

17. The method of claim **16** further comprising identifying a zero day attack.

18. The method of claim **12** further comprising creating a copy of the piece of malicious code and cataloging operating system changes caused by the piece of malicious code.

19. The method of claim **12** further comprising capturing communications with the plurality of computer systems.

* * * * *