(12) STANDARD PATENT (11) Application No. **AU 2014318585 B2**
(19) AUSTRALIAN PATENT OFFICE

(54) Title
Automated runtime detection of malware

(51) International Patent Classification(s)
G06F 21/56 (2013.01)

(21) Application No: **2014318585** (22) Date of Filing: **2014.09.12**

(87) WIPO No: **WO15/038944**

(30) Priority Data

(31) Number	(32) Date	(33) Country
61/960,209	**2013.09.12**	**US**

(43) Publication Date: **2015.03.19**
(44) Accepted Journal Date: **2018.01.04**

(71) Applicant(s)
Virsec Systems, Inc.

(72) Inventor(s)
Gupta, Satya Vrat;Demeo, Raymond F.

(74) Agent / Attorney
Davies Collison Cave Pty Ltd, Level 15 1 Nicholson Street, MELBOURNE, VIC, 3000, AU

(56) Related Art
US 2005/0108562 A1
US 8510596 B1
US 5983348 A

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 March 2015 (19.03.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/038944 A1

(51) International Patent Classification:
G06F 21/56 (2013.01)

(21) International Application Number:
PCT/US2014/055469

(22) International Filing Date:
12 September 2014 (12.09.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/960,209 12 September 2013 (12.09.2013) US

(71) Applicant: **VIRSEC SYSTEMS, INC.** [US/US]; 43 Nagog Park, Suite 215, Acton, MA 01720 (US).

(72) Inventors: **GUPTA, Satya, Vrat**; 3 Bramble Way, Acton, MA 01720 (US). **DEMEO, Raymond, F.**; 159 Fox Run Road, Bolton, MA 01740 (US).

(74) Agents: **MEAGHER, Timothy, J.** et al.; Hamilton, Brook, Smith & Reynolds, P.C., 530 Virginia Rd., P.O. Box 9133, Concord, MA 01742-9133 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

— *with international search report (Art. 21(3))*

WO 2015/038944 A1

(54) Title: AUTOMATED RUNTIME DETECTION OF MALWARE

(57) Abstract: One example method and correspond apparatus extracts a model of a computer application during load time and stores the model of the computer application in a database. This example method and corresponding apparatus also inserts instructions into the computer application to collect data at runtime. This example method and corresponding apparatus then analyzes the data collected at runtime against the stored model of the computer application to detect one or more security events and tracks the one or more security events using a state machine.

2014318585 22 Nov 2017

AUTOMATED RUNTIME DETECTION OF MALWARE

RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional Application No. 61/960,209, filed on September 12, 2013. The entire teachings of the above application are incorporated herein by reference.

BACKGROUND

[0002] With each passing day, cyber-attacks are becoming increasingly sophisticated. A large number of attacks are targeted to exploit specific vulnerabilities in specific applications. These attacks are not discernible at the network layer because they do not trigger network activity that appears to be overtly malicious. In order to deal with these targeted attacks, many vendors have deployed products, such as next generation firewalls which attempt to trace the behavior of the application or sandboxing technologies which attempt to run suspicious code in a sandbox and wait for the suspicious code to perform malicious activity. However, in these scenarios, malware simply adapts its behavior slightly or waits for a longer period of time to carry out its malicious intent. These changes in behavior hamper these products' ability to recognize the behavior of the attack and therefore their ability to detect the malware is greatly diminished.

[0002A] It is desired to provide a computer-implemented method, a computer-implemented method, a system, and an apparatus, that alleviate one or more difficulties of the prior art, or to at least provide a useful alternative.

SUMMARY

[0002B] In accordance with some embodiments of the present invention, there is provided a computer-implemented method comprising:

as a module of a computer application loads into memory:

disassembling machine code of the module;

examining the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for

2014318585 22 Nov 2017

determining a target address, wherein direct transition instructions determine a target address at load time, and indirect transition instructions have a runtime dependency preventing determining a target address until runtime,

for each identified transition instruction,

(i) storing the target address of the identified transition instruction, if the identified transition instruction is a direct transition instruction, storing the identified transition instruction, if the identified transition instruction is an indirect transition instruction; and

(ii) inserting one or more collection instructions into the module of the computer application to collect data at runtime; and

as the module of the computer application executes at runtime:

analyzing the data collected at runtime against the stored target addresses and stored indirect transition instructions to detect one or more security events; and

tracking the one or more security events using a state machine.

[0002C] In accordance with some embodiments of the present invention, there is provided a computer-implemented method comprising:

as a module of a computer application loads into memory:

disassembling machine code of the module;

examining the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for determining a target address, wherein a direct transition instruction determines a target address at load time, and an indirect transition instruction has a runtime dependency preventing determining a target address until runtime, for each identified transition instruction,

storing the target address of the identified transition instruction, if the identified transition instruction is a direct transition instruction, storing the identified transition instruction, if the identified transition instruction is an indirect transition instruction; and

2014318585 22 Nov 2017

inserting one or more collection instructions into the module of the computer application to collect data at runtime.

[0002D] In accordance with some embodiments of the present invention, there is provided a computer-implemented method comprising:

as a module of a computer application executes at runtime:

analyzing data collected at runtime for a computer application against stored target addresses and indirect transition instructions of the computer application to detect one or more security events; and

tracking the one or more security events using a state machine.

[0002E] In accordance with some embodiments of the present invention, there is provided a system comprising:

a client configured to:

as a module of a computer application loads into memory:

disassemble machine code of the module;

examine the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for determining a target address, wherein a direct transition instruction determines a target address at load time, and an indirect transition instruction has a runtime dependency preventing determining a target address until runtime,

for each identified transition instruction,

(i) store the target address of the identified transition instruction, if the identified transition instruction is a direction transition instruction, store the identified transition instruction, if the identified transition instruction is an indirect transition instruction;

2014318585 22 Nov 2017

(ii) insert one or more collection instructions into the module of the computer application to collect data at runtime; and

as the module of the computer application executes at runtime:

an analysis engine configured to:

analyze the data collected at runtime against the stored target addresses and indirect transition instructions to detect one or more security events; and

track the one or more security events using a state machine.

[0002F] In accordance with some embodiments of the present invention, there is provided an apparatus comprising:

a processor configured to execute a first process and a second process;

as a module of a computer application loads into memory:

the first process configured to

disassemble machine code of the module;

examine the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for determining a target address, wherein a direct transition instruction determines a target address at load time, and an indirect transition instruction has a runtime dependency preventing determining a target address until runtime,

for each identified transition instruction,

store the target address of the identified transition instruction, if the identified transition instruction is a direct transition instruction,

store the identified transition instruction, if the identified transition instruction is an indirect transition instruction;

the second process configured to insert one or more collection instructions into the module of the computer application to collect data at runtime.

2014318585 22 Nov 2017

[0002G] In accordance with some embodiments of the present invention, there is provided an apparatus comprising:

a processor configured to execute a first process and a second process;

the first process configured to analyze data collected at runtime for a computer application against stored target addresses and indirect transition instructions of the computer application to detect one or more security events, wherein the model includes transition mapping data from the computer application; and

the second process configured to track the one or more security events using a state machine.

[0003] Computer applications, including but not limited to single and multitier, closed and distributed, standalone, web-based, and cloud-based, are vulnerable to malware attacks. The largest number of malware attacks of computer applications today result from the ability of a malicious actor to inject and later execute malicious content in a running process of a computer application. The process of injecting such malicious content involves identifying and exploiting poorly designed code that performs inadequate input validation. The current cyber security technologies attempt to either observe malicious content in the application or trace the behavior of an

application or screen the behavior of suspicious code in a sandbox. These technologies do not have the capability to examine computer applications in real time at a low enough granularity to reliably detect events that indicate the injection of malicious content. In addition, these technologies do not have the capability to track and correlate such events over time in order to accurately identify these malware attacks before the malware successfully carries out its malicious intent.

[0004] One example method and corresponding apparatus described herein extracts and stores a model of a computer application at load time, wherein the model includes transition mapping data from the computer application. This example method and corresponding apparatus also inserts instructions into the computer application at load time in order to collect data at runtime. The data collected at runtime is analyzed against the stored model of the computer application to detect one or more security events. This example method and corresponding apparatus tracks the one or more security events triggered by an attacker using a state machine.

[0005] The method and corresponding apparatus may extract as part of the model of the computer application data one or more of the following: memory mapping data, soft spot data, and/or OS functions or system calls that affect access permissions and privileges referenced by the computer application. Such information may be saved in a model database. Furthermore, the method and corresponding apparatus may extract the model of the computer application at least in part using a code disassembler. The computer application being extracted may be in various formats including binary format or interpreted format.

[0006] The method and corresponding apparatus may check the computer application for integrity during load time. The method and corresponding apparatus may check the computer application for integrity by computing a checksum such as the MD5 hash of the code or using a trusted checksum verification service.

[0007] The model database may contain one or more tables for modeling the computer application. Furthermore, the model database may be on a local or remote system. If the model database is on a remote system, the method and corresponding apparatus may package the model of the computer application for transmission to the

2014318585 13 Apr 2017

remote system for storing in the database. The packaged model of the database may be transmitted using a standards based transport protocol such as TCP/IP or UDP.

[0008] The method and corresponding apparatus may insert instrumentation instructions at load time into the computer application using a dynamic binary analysis engine or a byte code instrumentation engine. When the instrumented application runs, the data collected at runtime may be packaged for transmission to another process for analyzing. This other process for analyzing may be on a local or a remote system. Furthermore, the data collected at runtime may comprise data for one or more threads of the computer application.

[0009] When analyzing the data collected at runtime against the stored model of the computer application, the method and correspond apparatus may analyze one or more of the following: transition data, Critical OS functions and system calls that affect access permissions and privileges, memory writes, heap allocation or de-allocation, and/or soft spot data.

[0010] Tracking the one or more security events using a state machine may comprise correlating the events based on a predefined sequence. Tracking the one or more security events may also include capturing forensic data for the events. The one or more security events may be tracked using severity levels. Furthermore, one or more actions may be taken in response to the occurrence of one or more security events. In one example embodiment, the one or more actions in response to an event are automatically taken by the system, and in another example embodiment, the one or more actions can be taken after manual intervention by the user. The one or more actions may include one or more of the following: terminating one or more threads of the computer application, closing a communication socket on one or more threads of the computer application, terminating the application, recording the event, and/or generating alerts in response to the one or more security events.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] Some embodiments of the invention are herein described , by way of example only, with reference to the accompanying drawings in which like reference characters refer to the same parts

throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating embodiments of the present invention.

[0012] Fig. 1 depicts an example configuration of an advanced persistent malware threat;

[0013] Fig. 2 shows an example chain of immutable events indicating a code execution style attack;

[0014] Fig. 3 shows an example flow chart of load time operations executed by a client;

[0015] Fig. 4 depicts an example protocol data unit (PDU) used to transmit data between the client and an analysis engine;

[0016] Fig. 5 shows an example flow chart of a process used by the analysis engine to generate events based on data collected at runtime;

[0017] Fig. 6 depicts an example Event Chain State Machine used to track generated events; and

[0018] Fig. 7 depicts an example block diagram of the client and analysis engine.

[0019] Fig. 8 illustrates a computer network or similar digital processing environment in which embodiments of the present invention may be implemented.

[0020] Fig. 9 is a diagram of an example internal structure of a computer (e.g., client processor/device or server computers) in the computer system of Fig. 8.

DETAILED DESCRIPTION

[0021] A description of example embodiments of the invention follows.

[0022] The teachings of all patents, published applications and references cited herein are incorporated by reference in their entirety.

[0023] Fig. 1 represents an example of an advance persistent malware threat. In this scenario, a malicious actor (or hacker) from the actor's own infrastructure 102 remotely scans the web facing the infrastructure of an enterprise 108 using scanning tools such as nmap 104. When the scan finds a server that has either a known vulnerability or zero-day vulnerability, the actor installs shell code 106 to gain control of the remote enterprise server 110 and access the enterprise network. Once inside the network, the malicious actor loads additional tools 104. These tools may include

nmap, port scanners, password cracking tools, ftp client, compression tools, hashing, and/or encryption and decryption tools.

[0024] The actor then accesses the enterprise infrastructure looking for a machine 114 or 116 from which elevated privileged users log into valuable targets such as databases and file repositories for the purposes of scraping the access credentials of the user and for finding a home for more hacking tools. Upon finding a machine with vulnerable applications 114 or 116, the malicious actor can scrape for credentials, infiltrate and subsequently access the target 118. Once the target is accessed, additional tools of the trade are loaded onto the target 104. Malware can also be deposited onto mobile devices such as smartphones and laptops of privileged users as they take their machines through unprotected networks such as coffee shops, airports, and hotels. In another scenario, an inside user may infect target machines.

[0025] By using sophisticated password cracking tools or snooping intranet traffic, the malware can obtain the credentials of the administrative user 116. After credentials have been obtained, the malicious actor can connect to the databases and file repositories 118 with impunity and extract valuable data such as real names, home addresses, social security, driver licenses, birth dates, medical records, financial information such as credit/debit cards, phone numbers, email addresses, user names and passwords, and insurance information. The malicious actor can optionally compress and encrypt this information and upload it to the Hacker Command Control Center 112 in small chunks so as to not draw attention of the enterprise security analysts. To achieve this objective, the malicious actor changes the IP addresses of the Hacker Command Control Center 112 on a daily basis or uses proxies so that intrusion detection systems in the enterprise cannot establish a pattern. An enterprise typically sends and receives over 10 GB of data every day; therefore uploading relatively small amounts of data in short bursts often goes unnoticed.

[0026] Being able to detect and block the chain of events for this type of scenario reliably is the key to thwarting such advanced persistent threats. Contemporary cyber security tools suffer from four major shortcomings. First, these tools do not examine the application at a low enough granularity. Without this capability, many signs of attacks cannot be discerned reliably. Next, these tools do not have the capability to track the attack over time. The typical security information and event management

(SIEM) systems only correlate events (that are of too high a granularity) for typically 24 hours. Malware can simply lie dormant for a long enough period of time to escape detection. Without the capability to track the attack reliably over time, the individual malicious events contributing to the attack appear as disjoint un-correlated events.

[0027] In addition, these cyber security tools depend on security analysts to set the threshold of events that signify an attack. For example, one security analyst may have internalized that some number of port scans and login attempts per hour will occur in the normal course of the day. The question becomes how many port scan attempts per source per hour are one too many before an alert is triggered. If an alert is generated too early, an analyst may be investigating port scans with no malicious intent or failed logins by legitimate users. If an alert is raised too late, then the malware attack may have already succeeded. Furthermore, these tools have an incomplete knowledge base to effectively detect the attack. The ability to generate truly genuine alerts requires the tools to distinguish between benign and malicious events reliably. A malicious actor would be aware of the same signatures or known suspicious network or application behaviors. Therefore, the malicious actor can tweak the behavior of the attack, such as using encryption to encode the data, changing IP and port combinations, or by slowing down the attack so as to avoid detection based on the behavior specified in the knowledge base.

[0028] Fig. 2 shows a chain of immutable events depicting a code execution attack. For various malware attacks, such as code execution attacks, a corresponding event chain of the malware attack can be represented by a series of immutable events. These events can be stored in the process memory of a computer for tracking specific malware attacks. The following scenario describes the behavior represented by the series of immutable events at 202-216 of a code execution attack.

[0029] In executing a code execution attack, once a malicious actor locates an unpatched application or a so-called 0-day vulnerability, the actor can compose a specially crafted payload of malformed content. This payload is then sent directly or through a network to a target process at 202 running on the central processing unit (CPU) of a computer system to divert the CPU from calling the instructions of a computer application and instead calls instructions at the behest of the malicious payload. This specially crafted payload can be injected into the application through

many mechanisms, such as over a network socket, through the keyboard, or even through a file, depending on the application being targeted.

[0030] To activate the malicious payload injected during the attack at 206, the malware can leverage one of many attack vectors. In the case of the code execution attack, the malware exploits a buffer error or user naivety at 204. Other examples of attacks vectors include the simplistic Stack Smashing approach, using format specifiers, finding the pseudo random stack canary, over running the exception handler tables, or Return Oriented Programming (ROP) gadgets as well as many additional vectors.

[0031] Before commencing its full blown malicious activity, the malware can hibernate for a sufficiently long period of time so that detection techniques can be subverted. For example sandbox techniques that inspect emails for suspicious activity must eventually give up and deliver the email to the recipient. At some later point in time, the malware can use either an existing application thread or spin one or more new threads at 212 to start executing its intent. In some cases, using an existing thread may attract user attention and spinning new ones ay go unnoticed. This is because most security solutions do not have the ability to determine if spinning a new thread is malicious or benign activity. Having now created a wedge in the application, the malware establishes connectivity with the malicious actor's Command Control Center (C&C) at 210. Once connectivity from inside the enterprise firewall is established, the thread on which malware can download more malware at 208 such as password cracking tools, port scanning tools, encryption tools etc. in bits and pieces so as to not attract attention of file blacklisting solution.

[0032] Once the tools are downloaded, the malware looks to extract useful content from the infected machine and all other machines reachable from the infected machine. In the code execution attack in Fig. 2, the malware may keep searching the enterprise until it finds a user with elevated privileges. Alternatively, it could sniff traffic, such as SMB/NETBIOS/CIFS traffic, in the enterprise to find privileged users' user names and login credentials. Alternatively, it could use password cracking tools or simply compare the hash of guess passwords with contents of password files. With today's processing power, an 8 character long password can be cracked in a few hours.

[0033] After the privileged user's credentials have been extracted, the malware is adequately armed and can go about its business of extracting useful content from the infected machine and all other machines reachable from the infected machines. The list of reachable machines may include database servers, code repositories, or CAD machines with valuable design documents. Once the valuable content has been extracted, the malware may encrypt data or upload data to the Command Control Center at 216 before the attack is culminated. If data is encrypted, the malicious actor may contact the target with a ransom request.

[0034] Fig. 3 shows the operations that an example client referred to herein as the Resolve Client performs at load time to prepare for detecting malware activity, in accordance with principles of the present disclosure. The Path Validation Engine is part of the Resolve Client that can reliably detect malware activity within microseconds from the point the malware starts to run. The Resolve Client first verifies the integrity and then analyzes each module of the application in order to extract a model of the application. The model of the application is stored in an Application Map Database that may contain the following tables: Code Table, Export Table, V Table, Other Table, Basic Block Table, Soft Spot Table, Memory Operand Table, Transition Table, Disassembly Table, and Critical OS Functions Table. In the embodiment in Fig. 3, the Application Map Database is located on a remote system from the Resolve Client. In other embodiments, the Application Map Database can be saved on the same hardware where the application is executing or on hardware external to both the Resolve Client and Analysis Engine. The Resolve Client uses a Streaming Engine to package the extracted model of the application into Resolve Protocol Data Units (PDUs) to dispatch the data to be stored in the Application Map Database on the analysis system. The PDU structure is shown in Fig. 4.

[0035] After the Resolve Client starts processing the application at load time at 302, the same operations are performed in a loop for each module of the computer application at 304 and 306. As each module of the application loads in memory, the Resolve Client examines all the executables and libraries of the given module using a disassembler such as a machine code or a byte code disassembler. The modules of the application file are in a standard file format, such as Executable and Linkable Format (ELF) or Common Object File Format (COFF). In this format, the modules of the

application are organized into sections that include a code section, exported data section, v-table section, and other additional sections. As each module of the application loads in memory, the Resolve Client extracts these data sections as part of the model of the application. The bounds and access attributes of the code section of the module are dispatched and saved to the Application Map Database in the Code Table at 314. Each record in this table is of the form {Start Address, End Address}. The bounds and number of instructions of each basic block in the code section of the module are dispatched and saved in the Application Map database in the Basic Block Table at 330. Each record in this table is of the form {Start Address, End Address, and Number of instructions}. The bounds and access attributes of the exported data section of the module are saved in the Application Map database in the Export Table at 318. Each record in this table is of the form {Start Address, End Address}. The bounds and access attributes of a v-table section (if any) of the module are dispatched and saved in the Application Map database in the V Table at 322. Each record in this table is of the form {Start Address, End Address}. The bounds and access attributes of all other sections of the module are dispatched and saved in the Application Map database in the Other Table at 326. Each record in this table is of the form {Start Address, End Address, and Protection Attributes}.

[0036] As each module loads into memory, the Resolve Client also extracts other memory mapping data and soft spot datafrom the modules of the application. Memory mapping data includes instructions for memory allocation, memory de-allocation, and memory writes to critical segments of memory. Soft spot data includes instructions for manipulating large memory buffers (spot spots) including instructions that execute loops (such as instructions with REP style opcodes). The address of soft spot instructions and the size of each memory write are dispatched and saved in the Application Map database in the Soft Spot Table at 334. Each record in this table is of the form {Address, Write size}. The address and the write size will be stored for memory write instructions where the destination is a memory operand. This data is stored in the Application Map Database in the Memory Operand Write Table at 340. Each record in this table is of the form {Source Address, Memory Write Size}.

[0037] As each module of the application loads into memory, the Resolve Client also extracts transition mapping data (branch transfer or transition data) from the module. The transition mapping data can be for a direct transition mapping where transition instructions for the target address can be presently determined or for an indirect memory mapping where transition instructions for the target address have run time dependency preventing these instructions from being fully determined until runtime. The full disassembly of instructions where indirect transitions occur are dispatched and saved in the Application Map Database in the Disassembly Table at 324. All the extracted transition mappings are also dispatched and saved in the Application Map Database in the Transition Table at 324 and 332. Each record in this table is of the form {Source Address, Destination Address}. In addition, an operator can manually add Transition Mapping Data into the Map Transition Table prior to runtime at 320. In order to add records manually into the Map Transition Table, an operator may be required to authenticate themselves using a 2-factor authentication process to eliminate possible tampering of the Transition Table by malware.

[0038] Transition mapping is central to the ability of the Path Validation Engine to reliably detect malware activity within microseconds from the point that the malware starts to run. The concept of transition mapping can be better understood by an examination of source code. In the following sample source code, the function main() makes a call to the function printf() which is defined in a library, but does not make a call to function notCalled(). After a compiler and linker have run through this code and the binary produced is examined, the inter-relationships or lack thereof between the functions main(), printf() and notCalled() is preserved. The function main() is said to have a "transition" to the function printf() which can be expressed as {Address $_{SRC}$ -> Address $_{DST}$} where Address $_{SRC}$ is the address of the instruction where the function printf() is called in the function main() and Address $_{DST}$ is the address of the function printf(). The source and target could be a system call or an exception handler. A record such as the above is a single record in the application's Transition Map Table.

```
//C hello world example
#include <stdio.h>
```

```
int main()
{
printf("Hello world\n");
return 0;
}

int notCalled()
{
printf("Feeling lonely !\n");
return 0;
}
```

[0039] While the above example is written in C/C++ which is a compiled language, this analogy between transitions in source code can be envisioned in code written in any other language, including interpreted or JIT compiled code. The same example is shown in an interpreted language like Java as shown below.

```
/*
Java Hello World example.
*/

public class HelloWorldExample {

  public static void main(String args[]){
    System.out.println("Hello World !");
  }

  public static void notCalled() {
    System.out.println("Feeling lonely !");
  }

}
```

[0040] As each module of the application loads into memory, the Resolve Client also checks the application for integrity at 308. In one embodiment, this is accomplished by computing a checksum such as the MD5 hash of the code as it is loading and comparing it against its corresponding known good checksum saved in a Checksum database. Alternatively, a trusted checksum verification service can also be leveraged. This ensures that the code of the currently loading module is not already corrupted with malware. The Resolve Client may be configured to throw an alarm if the integrity check fails at 310.

[0041] At load time, particular OS functions and system calls that affect access permissions and privileges are also identified and their addresses are dispatched and saved in the Critical OS Functions Table at 312 and 316. The particular OS functions and system calls dispatched by the Resolve client have long reaching effects on the execution path of the executable. These administrative and critical OS functions and system calls change access permissions of memory segments, bump up access privileges, changes the no-execute policy, changes the Structured Exception Handler protection, shuts down the Address Space Layout Randomization policy, allocated and de-allocates memory, creates a new process, creates a new thread, or are involved in encrypting and decrypting data.

[0042] As each module of the application loads into memory, the Resolve Client additionally instruments instructions that are inserted into the module of the application to collect data at runtime. The instrumented code is inserted into the modules of the application using a dynamic binary analysis engine and/or a byte code instrumentation engine. Soft spot instructions are instrumented in areas within the modules that malware tend to attack, such as instructions that execute loops, to collect data to track activities in these areas at runtime at 338. Direct and indirect transition mapping instructions are instrumented in the modules to collect data to track activities involving transition mappings at runtime at 328. Memory Operand Write instructions are instrumented in the modules to collect data on memory write activities at runtime at 336. In the presence of self-modifying code, the basic blocks may change at run time. Additionally, instructions are instrumented in the application to collect data for activities involving OS functions and systems calls stored in the Critical OS Function Table at 312 and 316.

[0043] As a result of the instrumentation inserted at load time, critical information is generated at run time and collected for analysis. As the transition mapping data related instrumentation is accessed, the Resolve Client collects the thread ID, current instruction address, destination instruction address and optionally data contained in each general purpose register. As the Soft Spot instrumentation is accessed before the instruction is executed, the Resolve Client captures the thread ID and the bounds of the stack through appropriate registers. As the soft spot instrumentation is completed, the Resolve Client captures the thread ID and a few general purpose registers that

allow it to estimate the region of memory updated by this write operation. As the critical API or OS call instrumentation is accessed before the call is executed, the Resolve Client captures the thread ID, API name or System Call number and input parameters. As the critical API or OS call instrumentation is accessed after the call is executed, the Resolve Client captures the thread ID, API name or System Call number and return value. Instrumentation in the OS functions or system calls that allocate or de-allocate memory helps to track the regions of memory that are currently involved in the various heaps the application may have created. This memory envelop is leveraged to track the target of indirect memory writes run time in order to find if the malware wants to overrun control structures in the heap. In addition, by tracking the bounds of basic blocks using a cache, the Analysis Engine can determine if the basic block has changed. When the determination is positive, the Basic Block Table in the model database can be updated.

[0044] In this example embodiment, the Resolve Client then dispatches the captured information to a Streaming Engine to be packaged into a PDU for transmission to the Analysis Engine. The Streaming Engine uses a very low overhead OS artifact such as pipes or local procedure calls to move the data generated by the various instrumentation to another process so that the instrumented process can continue its normal course of operation. As in this example embodiment, the Streaming Engine can also package the information collected from the instrumentation into a Resolve PDU for further transmission to the Analysis Engine using an appropriate standards based transport protocol. In one embodiment, the transport protocol may be TCP/IP. In another embodiment, it may be UDP. In yet another embodiment the transport protocol may involve using shared memory technologies such as pipes or local procedure calls.

[0045] Fig. 4 depicts the Resolve PDU. In order for the Resolve Client and the Analysis Engine to work effectively with each other, they communicate with each other using the Resolve PDU. The Resolve PDU can specifically be used by the Resolve Client to package the extracted model of the application and/or collected runtime data for transmission to the Analysis Engine. The Resolve PDU contains fields for each type of information to be transmitted between the Resolve Client and

the Analysis Engine. The Resolve PDU is divided into the Application Provided Data Section, the HW/CAE Generated, and Content Analysis Engine or Raw Data sections.

[0046] The Application Provided Data Section contains data from various registers as well as source and target addresses that are placed in the various fields of this section. The Protocol Version contains the version number of the Resolve PDU 402. As the Resolve protocol version changes over time, the source and destination must be capable of continuing to communicate with each other. This 8 bit field describes the version number of the Resolve packet as generated by the source entity. A presently unused reserved field 404 follows the Protocol Version field.

[0047] The next field of the Application Provided Data Section is the Message Source/Destination Identifiers 406, 408, and 410 are used to exchange traffic within the Analysis Engine infrastructure as shown in Fig. 7. From time to time, the various entities shown in Fig. 7, exchange traffic between themselves. Not all these devices have or need IP addresses and therefore, the two (hardware and host) Query Router Engines uses the Message Source and Destination fields to route traffic internally. Some messages need to go across the network to entities in the Analysis Engine. For this purpose, the entities are assigned the following IDs. A given Analysis Engine appliance may have more than one accelerator card. Each card will have a unique IP address; therefore, the various entities will have a unique ID. The aforementioned infrastructure may also be running more than one application. Since each application server will have a unique IP address, the corresponding Resolve client side entity will also have a unique ID.

[0048]

Resolve Client Side Entities

1. Resolve GUI
2. Instrumentation and Analysis Engine
3. Client Message Router
4. Streaming Engine
5. Client Side Daemon
6. CLI Engine
7. Client Watchdog
8. Client Compression Block
9. Client iWarp Ethernet Driver (100 Mb/1Gb/10Gb)

Per PCI Card Entities (starting address = 20 + n*20)

20. Securalyzer TOE block

21. Securalyzer PCI Bridge
22. Decompression Block
23. Message Verification Block
24. Packet Hashing Block
25. Time-Stamping Block
26. Message Timeout Timer Block
27. Statistics Counter Block
28. Securalyzer Query Router Engine
29. Securalyzer Assist

Securalyzer Host Entities

200. Securalyzer PCIe Driver
201. Host Routing Engine
202. Content Analysis Engine
203. Log Manager
204. Daemon
205. Web Service Engine
206. Watchdog
207. IPC Messaging Bus
208. Configuration Database
209. Log Database

SIEM Connectors

220. SIEM Connector 1 – Virsec Dashboard
221. SIEM Connector 2 – HP ArcSight
222. SIEM Connector 3 – IBM QRadar
223. SIEM Connector 4 – Alien Vault USM

Securalyzer Infrastructure Entities

230. Virsec dashboard
231. SMTP Server
232. LDAP Server
233. SMS Server
234. Entitlement Server
235. Database Backup Server
236. OTP Client
237. OTP Server
238. Checksum Server
239. Ticketing Server
240. Virsec Rules Server
241. Virsec Update Server

All user applications

255. User Applications – Application PID is used to identify the application issuing a query

[0049] Another field of the Application Provided Data section is the Message Type field which indicates the type of data being transmitted 412. At the highest level, there are three distinct types of messages that flow between the various local Resolve client side entities, between the Analysis Engine appliance side entities and between Client side and appliance side entities. Furthermore, messages that need to travel over a network must conform to the OSI model and other protocols.

[0050] The following field of the Application Provided Data section is the Packet Sequence Number field containing the sequence identifier for the packet 414. The Streaming Engine will perform error recovery on lost packets. For this purpose it needs to identify the packet uniquely. An incrementing signed 64 bit packet sequence number is inserted by the Streaming Engine and simply passes through the remaining Analysis Engine infrastructure. If the sequence number wraps at the 64 bit boundary, it may restart at 0. In the case of non-application packets such as heartbeat or log message etc., the packet sequence number may be -1.

[0051] The Application Provided Data section also contains the Resolve Canary Message field contains a canary used for encryption purposes 422. The Resolve Client and the Analysis Engine know how to compute the Canary from some common information but of a fresh nature such as the Application Launch time, PID, the license string, and an authorized user name.

[0052] The Application Provided Data section additionally contains generic fields that are used in all messages. The Application Source Instruction Address 458, Application Destination Instruction Address 416, Memory Start Address Pointer 418, Memory End Address Pointer 420, Application PID 424, Thread ID 426, Analysis Engine Arrival Timestamp 428, and Analysis Engine Departure Timestamp 430 fields which hold general application data.

[0053] The Resolve PDU also contains the HW/CAE Generated section. In order to facilitate analysis and to maintain a fixed time budget, the Analysis Engine hashes the source and destination address fields and updates the Resolve PDU prior to processing. The HW/ CAE Generated section of the Resolve PDU is where the hashed

data is placed for later use. This section includes the Hashed Application Source Instruction Address 432, Hash Application Destination Instruction Address 434, Hashed Memory Start Address 436, and Hashed Memory End Address 438 fields. The HW/CAW Generated section additionally contains other fields related to the Resolve Canary 442 including the Hardcoded Content Start Magic header, API Name Magic Header, Call Context Magic Header and Call Raw Data Magic Header are present in all Resolve PDU packets.

[0054] The HW/CAW Generated section also includes a field 440 to identify other configuration and error data which includes Result, Configuration Bits, Operating Mode, Error Code, and Operating Modes data. The Result part of the field is segmented to return Boolean results for the different Analysis Engine queries – the transition playbook, the code layout, the Memory (Stack or Heap) Overrun, and the Deep Inspection queries. The Configuration Bits part of the field indicates when a Compression Flag, Demo Flag, or Co-located Flag is set. The presence of the flag in this field indicates to the Analysis Engine whether the packet should be returned in compression mode. The Demo Flag indicates that system is in demo mode because there is no valid license for the system. In this mode, logs and events will not be available in their entirety. The Co-located Flag indicates that the application is being run in the Analysis Engine so that Host Query Router Engine can determine where to send packets that need to return to the Application. If this flag is set, the packets are sent via the PCI Bridge, otherwise they are sent over the Ethernet interface on the PCI card. The Operating Mode part of the field indicates whether the system is in Paranoid, Monitor, or Learn mode. These modes will be discussed in more details later in this section. Lastly, the Error Code part of the field indicates an error in the system. The first eight bits of the error code will correspond to the message source. The remaining 12 bits will correspond to the actual error reported by each subsystem.

[0055] The Resolve PDU also contains the Content Analysis Engine or Raw Data. All variable data such as arguments and return value of the OS library calls and System Calls is placed in this section of the Resolve PDU. The data in this section contains the content of the data collected from the application and is primarily targeted at the Content Analysis Engine. This section contains the Variable Sized API Name or Number 444, the Call Content Magic Header 446, the Variable Sized Call

Content 450, the Call Raw Data Magic Header 452, Variable Sized Raw Data Contents 456, and two reserved 448 and 454 fields. Furthermore, these fields can be overloaded for management messages.

[0056] Fig. 5 shows the process used by Analysis Engine to detect security events at runtime. In this example embodiment, the Analysis Engine is a separate process running on the same or different hardware relative to the Resolve Client. By separating the data collection from the data analysis part, the processing overheads associated with the analysis of information generated by the instrumentation process is substantially reduced. This allows the Analysis Engine to perform runtime analysis for multiple applications at the same time.

[0057] During run time, instrumentation on the code provides an opportunity to examine the appropriate context, such as thread ID, stack start and base addresses, general purpose registers, source addresses, and destination addresses. The sequence of operations for examining the content has no specific order, is interchangeable, and can even be performed in parallel by multiple threads of the Analysis Engine. Using the process described in Fig. 5, the Secure Engine will generate individual events for each thread in the application that can be collected on a specific and individual Event Chain State Machine to track the progression of the attack in real time. The events generated by the Analysis Engine follow a standards based event format, such as SYSLOG. This enables a standards based event management system to use the events generated by the Resolve Client either directly or through a connector that translates SYSLOG to a standard target format, such as Common Event Format (CEF). Since the Analysis Engine has access to the previously saved Application Map with all its tables, when a performed transition instruction reports its context, the Analysis Engine has access to the extracted model of the application, including data indicating whether or not a target address is a valid destination.

[0058] After the process starts at 502, new runtime information is received in a Resolve PDU packet from the Resolve Client at 504. Before saving the packet in a buffer at 508, the Analysis Engine verifies the canary and timestamp contained in the packet at 506. While the Resolve PDU is still buffered, the Analysis Engine hashes the address fields and puts the hashed data in the HW/CAE section of the Resolve PDU at 510. The packet can then be pulled from the packet buffering for processing

at 512. When processing of the packet is complete, the process then waits for the next packet at 554.

[0059] The process used by the Analysis Engineer looks at transition mapping data. If the runtime information reported by a transition type instruction is from a direct transition at 520 and the target address is not found in the Transition Map Table at 528, the Analysis Engine generates an event with the details of the thread in which the event occurred at 544. If the runtime information reported by a transition type instruction is from an indirect transition at 522 and the target address is in the Code Table at 530 and in the middle of a basic block at 548, the Analysis Engine generates an event at 550 with the details of the thread in which the event occurred. If the runtime information reported by a transition type instruction is from an indirect transition at 522 and the target address is in a region associated with the heap memory at 530 and 548, the Analysis Engine generates an event at 550 with the details of the thread in which the event occurred. If the runtime information reported by a transition type instruction is from an indirect transition at 522 and the target address is in a non-code, non-import-table region of memory at 552, the Analysis Engine generates an event at 556 with the details of the thread in which the event occurred.

[0060] The process used by the Analysis Engine looks at Memory Write data at 516. If the runtime information being reported is from a memory write instruction and if the write target address is in a memory region of the V Table at 524, the Analysis Engine generates an event at 536 with the details of the thread in which the event occurred. If the runtime information being reported is from a memory write instruction and if the write target address is in a memory region of the Export Table at 524, the Analysis Engine generates an event at 536 with the details of the thread in which the event occurred. If the runtime information being reported is from a memory write instruction and if the write target address is in the target control section of a heap memory region at 538, the Analysis Engine generates an event at 536 with the details of the thread in which the event occurred. If the runtime information being reported is from a memory write instruction and if the write target address is in the target control section of the stack memory region at 532, the Analysis Engine generates an event 536 with the details of the thread in which the event occurred.

[0061] The process used by the Analysis Engine looks at soft spot data at 514. If the instruction is a soft spot instruction and the write target address is in a memory region of the V Table at 524, the Analysis Engine generates an event at 536 with the details of the thread in which the event occurred. If the instruction is a soft spot instruction and the write target address is in the memory region of the Export Table at 524, the Analysis Engine generates an event at 536 with the details of the thread in which the event occurred. If the instruction is a soft spot instruction and the write target address is in the target control section of the heap memory region at 538, the Analysis Engine generates an event at 536 with the details of the thread in which the event occurred. If the instruction is a soft spot instruction and the write target address overwrites the instruction pointer saved in the control region of the stack at a memory location below the base pointer at 532, the Analysis Engine generate an event 536 with the details of the thread in which such an event occurred.

[0062] The process used by the Analysis Engine looks at OS functions and system calls at 518. If the instrumentation associated with the entry point of a critical OS function or system call is called at 526 because of an invalid OS function or system call operation at 542, the Analysis Engine generates an event at 544 with the details of the thread on which the event occurred when activities in the Critical OS Function Table occur. This causes an event being generated for calls to code that changes memory access permissions is declared invalid, calls to code that changes privilege levels is declared as potentially invalid, a privilege escalation occurs on a thread servicing a WAN based user is elevated, changes to the No-Execute policy is declared invalid, changes the Address Space Layout Randomization (ASLR) policy is declared invalid, changes the Safe Exception Handling (SEH) policy is declared invalid, calls to admin specified functions are not handled in accordance with the event associated with that particular function call, or calls to admin specified system calls are not handled in accordance with the event associated with that particular system call. In addition, critical system call operations at 526 to heap memory operations at 534 are used to update the heap memory envelop at 540. Those regions of memory that are related to allocation increase the memory envelop size whereas those that de-allocate free up the targeted region.

[0063] Fig. 6 depicts an Event Chain State Machine used to track detected security events. The Path Validation Engine generates events and tracks the state of the events in accordance with the event chain state machine shown. The Event Chain State Machine uses records stored in a Transition Playbook database to track these events. When a new thread starts, the Event Chain state machine is initialized to Start State at 602. The Event Management Engine serializes events so that the set of events on one thread are updated on the state machine for the appropriate thread. As transitions occur on the thread, the state machine records the target address but stays in the Start State. When the Path Validation Engine generates a medium or high severity event, the Event Chain state machine advances to the next state, the E_i,state at 604. This new state continues to record the target address of transitions as they occur. This process of the state machine updating to a new state on medium and high severity events and recording transition events continues at 606, 608, and 610 until the thread reaches the Terminate State at 612. The Security Analyst is notified when a highest severity level event is triggered and the Event Chain State Machine captures the forensics when an attack occurs.

[0064] Depending on the modes of the system, there are various remedial actions that can be taken in response to events. The remedial actions taken are performed in real-time. One remedial action may consist of shutting down the application. Other remedial actions may be to unbind or release the socket(s) associated with the threads on which the threat has appeared, terminate the thread on which a threat has appeared, and/or blacklisting the user that caused the threat. Another remedial action would be to unblock all sockets associated with the application server's socket(s). This will block all users connected at the time. Yet another remedial action may be to ignore the attack. This may be due to the analyst not considering the attack to be of major consequence. The most appropriate suggested remedial actions for a given event are pre-programmed so that if the security analyst has chosen to operate the Analysis Engine in automatic mode, the remedial action occurs automatically.

[0065] The Path Validation Engine of the Analysis Engine can be run in three modes: Monitor, Paranoid or Learning mode. The difference between these modes is how and when to enforce the remedial actions associated with the event(s) received on a given thread in the application. In the Monitor mode, as runtime information

arrives and is analyzed by the Analysis Engine, it generates notifications that are directed to the designated security analysts. It is then the security analyst's job to intervene and choose the most appropriate remedial action. The security analyst may decide to change an "ignore" type pre-programmed remedial action to a higher impact remedial action even before the thread has reached the Terminate State. On finalizing the remedial action, the Analysis Engine implements the remedial action when the appropriate authority in the enterprise clears the proposed remedial action for a given thread.

[0066] In the paranoid mode, the programmed (default or user configured) remedial action is executed automatically without any manual intervention from the security analyst. In either mode, once the remedial action is ready to be carried out, the Analysis Engine lets the Resolve Client know which remedial action to carry out. The Resolve Client then performs the action associated with the remedial action on the application. Once the remedial action is completed, the Resolve Client sends a confirmation message back to the Analysis Engine. On receiving the acknowledgement, the Analysis Engine performs housekeeping including updating the security analyst.

[0067] In learning mode, the Analysis Engine ignores all events and remedial actions. In this mode, the application runs in a pristine environment and records all events and event chains. The security analyst uses this information to build criteria for when an event should be raised and what remedial action should be associated with the said event.

[0068] Fig. 7 depicts a high level block diagram of an example Resolve Client and Analysis Engine infrastructure. This infrastructure may be configured on a various hardware including computing devices ranging from smartphones, tablets, laptops, desktops to high end servers. As shown in this figure, data collection performed by the Resolve Client may be segregated from analysis performed by the Analysis Engine to improve application performance. The infrastructure provides high availability to prevent hackers from subverting its protection against malware attacks. The Resolve Client interacts with an application to gather load time and runtime data. The infrastructure of the application 701 includes process memory 703, third-party libraries 704, kernel services 706, and an instruction pipeline 707. The infrastructure

of the Resolve Client 702 includes the Instrumentation & Analysis Engine (IAE) 705, graphical user interface (GUI) 711, Client Daemon 708, Configuration database 709, and Streaming and Compression Engine 710, and central processing unit (CPU) 736. Local or remote users 702 of the application 701 interact with the application either through devices like keyboards, mice or similar I/O devices or over a network through a communication channel that may be established by means of pipes, shared memory or sockets. In response the application process 703 dispatches appropriate sets of instructions into the instruction pipeline 707 for execution. The application may also leverage its own or third party libraries 704 such as libc.so (Linux) or msvcrtxx.dll (Windows). As functionality from these libraries is invoked, appropriate instructions from these libraries are also inserted into the instruction pipeline for execution 707. In addition the application may leverage system resources such as memory, file I/O etc. from the kernel 706. These sequences of instructions from the application, libraries and the kernel put together in a time ordered sequence deliver the application functionality desired by a given user.

[0069] As the application's code begins to load into memory, the IAE 705 performs several different load time actions. Once all the modules have loaded up, the instrumented instructions of the application generate runtime data. The Client Daemon 708 initializes the Instrumentation and Analysis Engine 705, the Streaming Engine 710 and the GUI 711 processes in the CPU at 736 by reading one or more configuration files from the Configuration database 709. It also initializes intercommunication pipes between the IAE, Streaming Engine, GUI, Analysis Engine and itself. The Client Daemon also ensures that if any Resolve Client process, including itself, becomes unresponsive or dies, it will be regenerated. This ensures that the Resolve Client is a high availability enterprise grade product.

[0070] The Instrumentation and Analysis Engine pushes load and runtime data collected from the application into the Streaming Engine. The Streaming Engine packages the raw data from the Resolve Client into the Resolve PDU. Then it pushes the Resolve PDU over a high bandwidth, low latency communication channel 712 to the Analysis Engine 711. If the Client and the analyzer are located on the same machine this channel can be a memory bus. If these entities are located on different hardware but in the same physical vicinity, the channel can be an Ethernet or Fiber

based transport, which allows remote connections to be established between the entities to transport the load and runtime data across the Internet.

[0071] The infrastructure of the Analysis Engine includes the Network Interface Card (NIC) 713, the Packet Pool 714, the Time Stamp Engine 715, the Processor Fabric 716, the Hashing Engine 717, the TCAM Engine 718, the Application Map database 719, and the Thread Context database 720. The infrastructure of the Analysis Engine further includes the Content Analysis Engine 721, the Events and Event Chains 722, the Event Management Engine 723, the Event Log 724, the Application Daemon 725, the Analysis Engine Configuration database 726, the Network Interface 727, the Dashboard 728, the SMS/SMTP Server 729, the OTP Server 730, the Upgrade Client 731, the Software Upgrade Server 732, Software Images 733, the Event Update Client 734, and the Event Upgrade Server 735.

[0072] The Resolve PDU together with the protocol headers is intercepted at the Network Interface Card 713 from where the Resolve PDU is pulled and put into the Packet Pool 714. The timestamp fields in the Resolve PDU are filled up by the Time Stamp Engine 715. This helps to make sure that no packet is stuck in the packet Pool buffer for an inordinately long time.

[0073] The Processor Fabric 716 pulls packets from the packet buffer and the address fields are hashed and replaced in the appropriate location in the packet. This operation is performed by the Hashing Engine 717. Then the Processor Fabric starts removing packets from the packet buffer in the order they arrived. Packets with information from the load time phase are processed such that the relevant data is extracted and stored in the Application Map database 719. Packets with information from the runtime phase are processed in accordance with Figure 5. The efficiency of the Analysis Engine can be increased or decreased based on the number of processors in the Processor Fabric.

[0074] The transition target data is saved in the Thread Context database 720 which has a table for each thread. The Processor fabric also leverages the TCAM Engine 718 to perform transition and memory region searches. Since the processor fabric performing lookups using hashes, the actual time used is predictable and very short. By choosing the number of processors in the fabric carefully, per packet throughput can be suitable altered.

[0075] When the Analysis Engine performs searches, it may, from time to time find an invalid transition, invalid operation of critical/admin functions or system calls, or find a memory write on undesirable locations. In each of these cases, the Analysis Engine dispatches an event of the programmed severity as described by the policy stored in the Event and Event Chain database 722 to the Event Management Engine 723. The raw event log is stored in the Event Log Database 724. The Dashboard can also access the Event Log and display application status.

[0076] A remedial action is also associated with every event in the Event and Event Chain database 722. A user can set the remedial action from a range of actions from ignoring the event in one extreme to terminating the thread in the other extreme. A recommended remedial action can be recommended to the analyst using the Event Update Client 734 and Event Upgrade Server 735. In order to change the aforementioned recommended action, an analyst can use the Dashboard 728 accordingly. The Dashboard provides a GUI interface that displays the state of each monitored application and allows a security analyst to have certain control over the application, such as starting and stopping the application. When an event is generated, the Event Chain advances from the normal state to a subsequent state. The remedial action associated with the new state can be taken. If the remedial action involves a non-ignore action, a notification is sent to the Security Analyst using and SMS or SMTP Server 729. The SMS/ SMTP address of the security analyst can be determined using an LDAP or other directory protocol. The process of starting or stopping an application from the Dashboard requires elevated privileges so the security analyst must authenticate using an OTP Server 730.

[0077] New events can also be created and linked into the Event and Event Chain database 722 with a severity and remedial action recommended to the analyst. This allows unique events and event chains for a new attack at one installation to be dispatched to other installations. For this purpose, all new events and event chains are loaded into the Event Upgrade Server 735. The Event Update Client 734 periodically connects and authenticates to the Event Upgrade Server to retrieve new events and event chains. The Event Update Client then loads these new events and event chains into the Events and Events Chain database 722. The Content Analysis Engine 721

can start tracking the application for the new attacks encapsulated into the new event chains.

[0078] Just as with the Client Daemon, the Appliance Daemon 725 is responsible for starting the various processes that run on the Analysis Engine. For this purpose, it must read configuration information from the Analysis Engine Configuration database 726. The daemon is also responsible for running a heartbeat poll for all processes in the Analysis Engine. This ensures that all the devices in the Analysis Engine ecosystem are in top working condition at all times. Loss of three consecutive heartbeats suggests that the targeted process is not responding. If any process has exited prematurely, the daemon will revive that process including itself.

[0079] From time to time, the software may be upgraded in the Appliance host, or of the Analysis Engine or of the Client for purposes such as fixing errors in the software. For this purpose, the Upgrade Client 731 constantly checks with the Software Upgrade Server 732 where the latest software is available. If the client finds that the entities in the Analysis Engine or the Client are running an older image, it will allow the analysts to upgrade the old image with a new image from the Software Upgrade Server 732. New images are bundled together as a system image 733. This makes it possible to provision the appliance or the host with tested compatible images. If one of the images of a subsystem in the Analysis Engine or the Resolve Client does not match the image for the same component in the System image, then all images will be rolled to a previous known good system image.

[0080] Fig. 8 illustrates a computer network or similar digital processing environment in which embodiments of the present invention may be implemented.

[0081] Client computer(s)/devices 50 and server computer(s) 60 provide processing, storage, and input/output devices executing application programs and the like. The client computer(s)/devices 50 can also be linked through communications network 70 to other computing devices, including other client devices/processes 50 and server computer(s) 60. The communications network 70 can be part of a remote access network, a global network (e.g., the Internet), a worldwide collection of computers, local area or wide area networks, and gateways that currently use respective protocols (TCP/IP, Bluetooth®, etc.) to communicate with one another. Other electronic device/computer network architectures are suitable.

2014318585 13 Apr 2017

[0082] Fig. 9 is a diagram of an example internal structure of a computer (e.g., client processor/device 50 or server computers 60) in the computer system of Fig. 8. Each computer 50, 60 contains a system bus 79, where a bus is a set of hardware lines used for data transfer among the components of a computer or processing system. The system bus 79 is essentially a shared conduit that connects different elements of a computer system (e.g., processor, disk storage, memory, input/output ports, network ports, etc.) that enables the transfer of information between the elements. Attached to the system bus 79 is an I/O device interface 82 for connecting various input and output devices (e.g., keyboard, mouse, displays, printers, speakers, etc.) to the computer 50, 60. A network interface 86 allows the computer to connect to various other devices attached to a network (e.g., network 70 of Fig. 8). Memory 90 provides volatile storage for computer software instructions 92 and data 94 used to implement an embodiment of the present invention (e.g., resolve client and analysis engine elements described herein). Disk storage 95 provides non-volatile storage for computer software instructions 92 and data 94 used to implement an embodiment of the present invention. A central processor unit 84 is also attached to the system bus 79 and provides for the execution of computer instructions.

[0083] While this invention has been particularly shown and described with references to example embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the scope of the invention encompassed by the appended claims.

[0084] Throughout this specification and claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" and "comprising", will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

[0085] The reference in this specification to any prior publication (or information derived from it), or to any matter which is known, is not, and should not be taken as an acknowledgment or admission or any form of suggestion that that prior publication (or information derived from it) or known matter forms part of the common general knowledge in the field of endeavour to which this specification relates.

2014318585 22 Nov 2017

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A computer-implemented method comprising:

as a module of a computer application loads into memory:

disassembling machine code of the module;

examining the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for determining a target address, wherein direct transition instructions determine a target address at load time, and indirect transition instructions have a runtime dependency preventing determining a target address until runtime,

for each identified transition instruction,

(i) storing the target address of the identified transition instruction, if the identified transition instruction is a direct transition instruction, storing the identified transition instruction, if the identified transition instruction is an indirect transition instruction; and

(ii) inserting one or more collection instructions into the module of the computer application to collect data at runtime; and

as the module of the computer application executes at runtime:

analyzing the data collected at runtime against the stored target addresses and stored indirect transition instructions to detect one or more security events; and

tracking the one or more security events using a state machine.

2. A computer-implemented method comprising:

as a module of a computer application loads into memory:

disassembling machine code of the module;

examining the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for

2014318585 22 Nov 2017

determining a target address, wherein a direct transition instruction determines a target address at load time, and an indirect transition instruction has a runtime dependency preventing determining a target address until runtime, for each identified transition instruction,

storing the target address of the identified transition instruction, if the identified transition instruction is a direct transition instruction, storing the identified transition instruction, if the identified transition instruction is an indirect transition instruction; and

inserting one or more collection instructions into the module of the computer application to collect data at runtime.

3. A computer-implemented method comprising:

as a module of a computer application executes at runtime:

analyzing data collected at runtime for a computer application against stored target addresses and indirect transition instructions of the computer application to detect one or more security events; and

tracking the one or more security events using a state machine.

4. The method of Claim 1 or 2 wherein disassembling machine code of the module includes: extracting direct and indirect transition mapping data from the computer application, extracting memory mapping data from the computer application, extracting instructions that manipulate memory buffers from the computer application, extracting instructions that execute program loops from the computer application, and extracting OS functions and system calls referenced by the computer application.

5. The method of Claim 1 or 2 wherein disassembling machine code of the module is accomplished at least in part using a code disassembler.

2014318585 22 Nov 2017

6. The method of Claim 1 or 2 wherein the computer application is in binary or interpreted format.

7. The method of Claim 1 or 2 further comprising:

checking the computer application for integrity during load time,

optionally wherein checking the computer application for integrity comprises computing a checksum.

8. The method of Claim 1 or 2 wherein storing the target address or target instruction includes storing the target address or target instruction in a database that contains one or more tables for modeling the computer application.

9. The method of Claim 1 or 2 wherein storing the target address or target instruction includes storing the target address or target instruction in a database on a remote system, optionally wherein the method further comprises:

packaging the target address or target instruction for transmission to the remote system for storing in the database; and

further optionally, wherein the method comprises:

placing a canary into the transmission to secure the transmission.

10. The method of Claim 1 or 2 wherein inserting one or more collection instructions into the computer application is accomplished at least in part using a dynamic binary analysis engine or a byte code instrumentation engine.

2014318585 22 Nov 2017

11. The method of Claim 1 or 2 further comprising:

packaging the data collected at runtime for transmission to one or more processes, optionally wherein the one or more processes are on a remote system or the method further comprises:

placing a canary into the transmission to secure the transmission.

12. The method of Claim 1 or 2 wherein the data collected at runtime comprises data for one or more threads of the computer application.

13. The method of Claim 1 or 3 wherein analyzing the data collected at runtime against the stored target addresses and transition instructions comprises one or more of analyzing transition data, analyzing OS functions, analyzing system calls, analyzing memory writes, analyzing instructions that manipulate memory buffers, analyzing instructions that execute program loops.

14. The method of Claim 1 or 3 wherein tracking the one or more security events comprises correlating the one or more security events based on a predefined sequence, optionally wherein the predefined sequence is based on an immutable chain.

15. The method of Claim 1 or 3 wherein tracking the one or more security events includes capturing forensic data for the events or wherein tracking the one or more security events comprises the use of severity levels.

16. The method of Claim 1 further comprising taking one or more actions in response to tracking the one or more security events, optionally wherein the one or more actions are automatically taken by a system or wherein the one or more actions are taken by a user or wherein the one or more actions include

2014318585 22 Nov 2017

any of terminating the computer application, include terminating one or more threads of the computer application, include closing a socket on one or more threads of the computer application, and generating alerts in response to the one or more security events or wherein the one or more actions taken in response to tracking the one or more security events are adjustable.

17. The method of Claim 1 or 3, further comprising: receiving the data collected at runtime from a process on a remote system; and/or further comprising receiving the stored target addresses and transition instructions from a process on a remote system.

18. A system comprising:

a client configured to:

as a module of a computer application loads into memory:

disassemble machine code of the module;

examine the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for determining a target address, wherein a direct transition instruction determines a target address at load time, and an indirect transition instruction has a runtime dependency preventing determining a target address until runtime,

for each identified transition instruction,

(i) store the target address of the identified transition instruction, if the identified transition instruction is a direction transition instruction, store the identified transition instruction, if the identified transition instruction is an indirect transition instruction;

(ii) insert one or more collection instructions into the module of the computer application to collect data at runtime; and

as the module of the computer application executes at runtime:

2014318585 22 Nov 2017

an analysis engine configured to:

analyze the data collected at runtime against the stored target addresses and indirect transition instructions to detect one or more security events; and

track the one or more security events using a state machine.

19. An apparatus comprising:

a processor configured to execute a first process and a second process;

as a module of a computer application loads into memory:

the first process configured to

disassemble machine code of the module;

examine the disassembled machine code to identify transition instructions, including direct and indirect transition instructions for determining a target address, wherein a direct transition instruction determines a target address at load time, and an indirect transition instruction has a runtime dependency preventing determining a target address until runtime,

for each identified transition instruction,

store the target address of the identified transition instruction, if the identified transition instruction is a direct transition instruction, store the identified transition instruction, if the identified transition instruction is an indirect transition instruction;

the second process configured to insert one or more collection instructions into the module of the computer application to collect data at runtime.

20. An apparatus comprising:

a processor configured to execute a first process and a second process;

the first process configured to analyze data collected at runtime for a computer application against stored target addresses and indirect transition

2014318585 22 Nov 2017

instructions of the computer application to detect one or more security events, wherein the model includes transition mapping data from the computer application; and

the second process configured to track the one or more security events using a state machine.

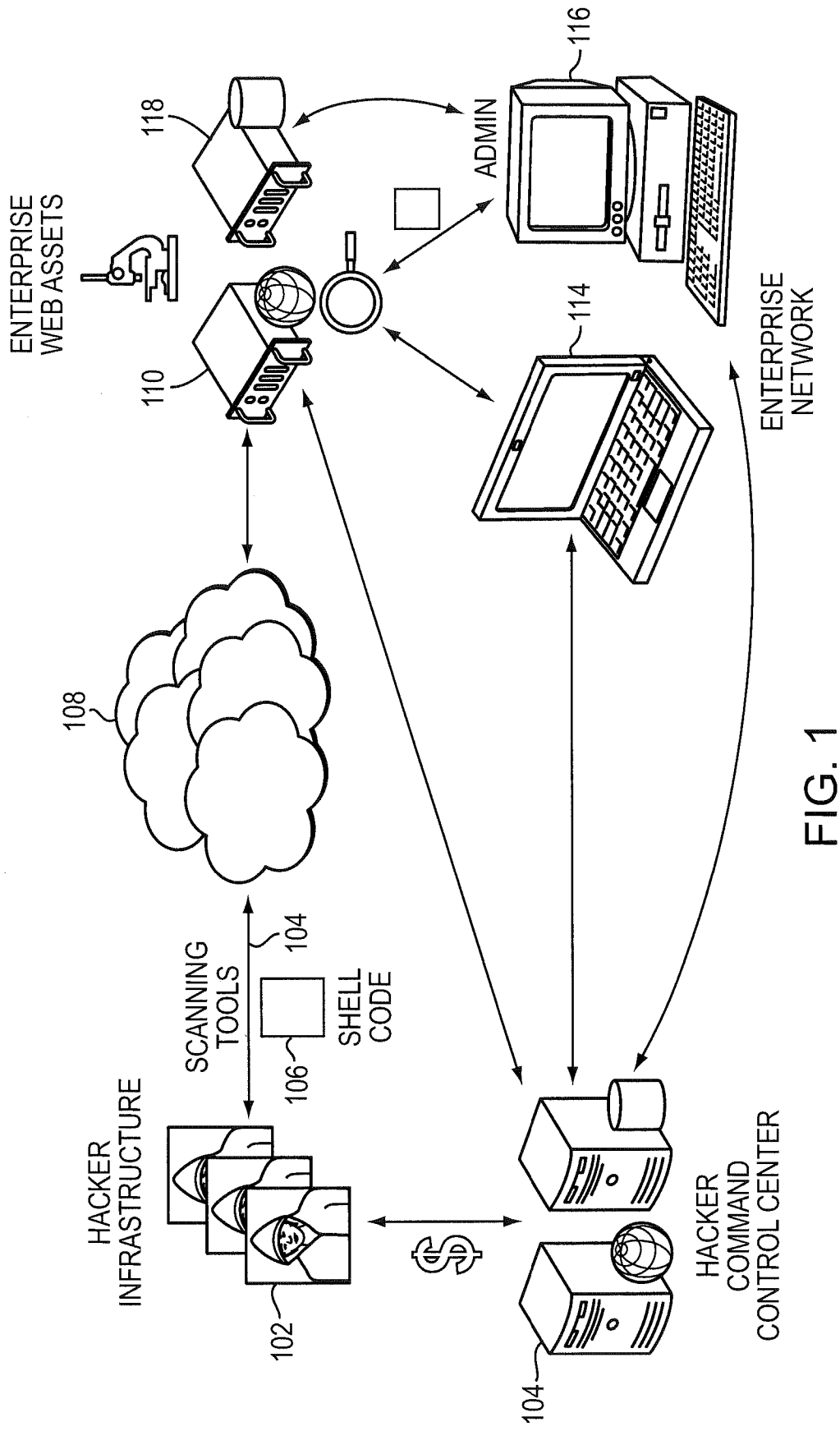
ENTERPRISE WEB ASSETS
118
110
ADMIN
116
114
ENTERPRISE NETWORK
108
SCANNING TOOLS
104
SHELL CODE
106
HACKER INFRASTRUCTURE
102
HACKER COMMAND CONTROL CENTER
104

FIG. 1

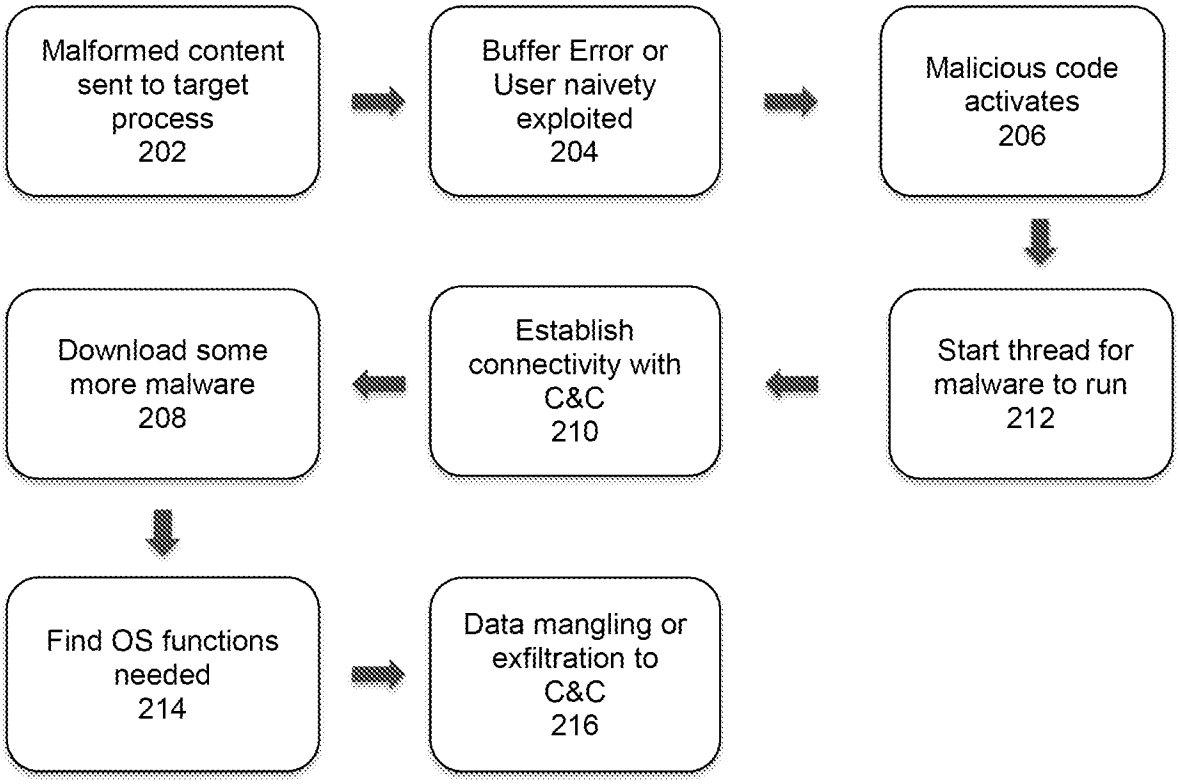
Malformed content sent to target process 202
Buffer Error or User naivety exploited 204
Malicious code activates 206
Start thread for malware to run 212
Establish connectivity with C&C 210
Download some more malware 208
Find OS functions needed 214
Data mangling or exfiltration to C&C 216

FIG. 2

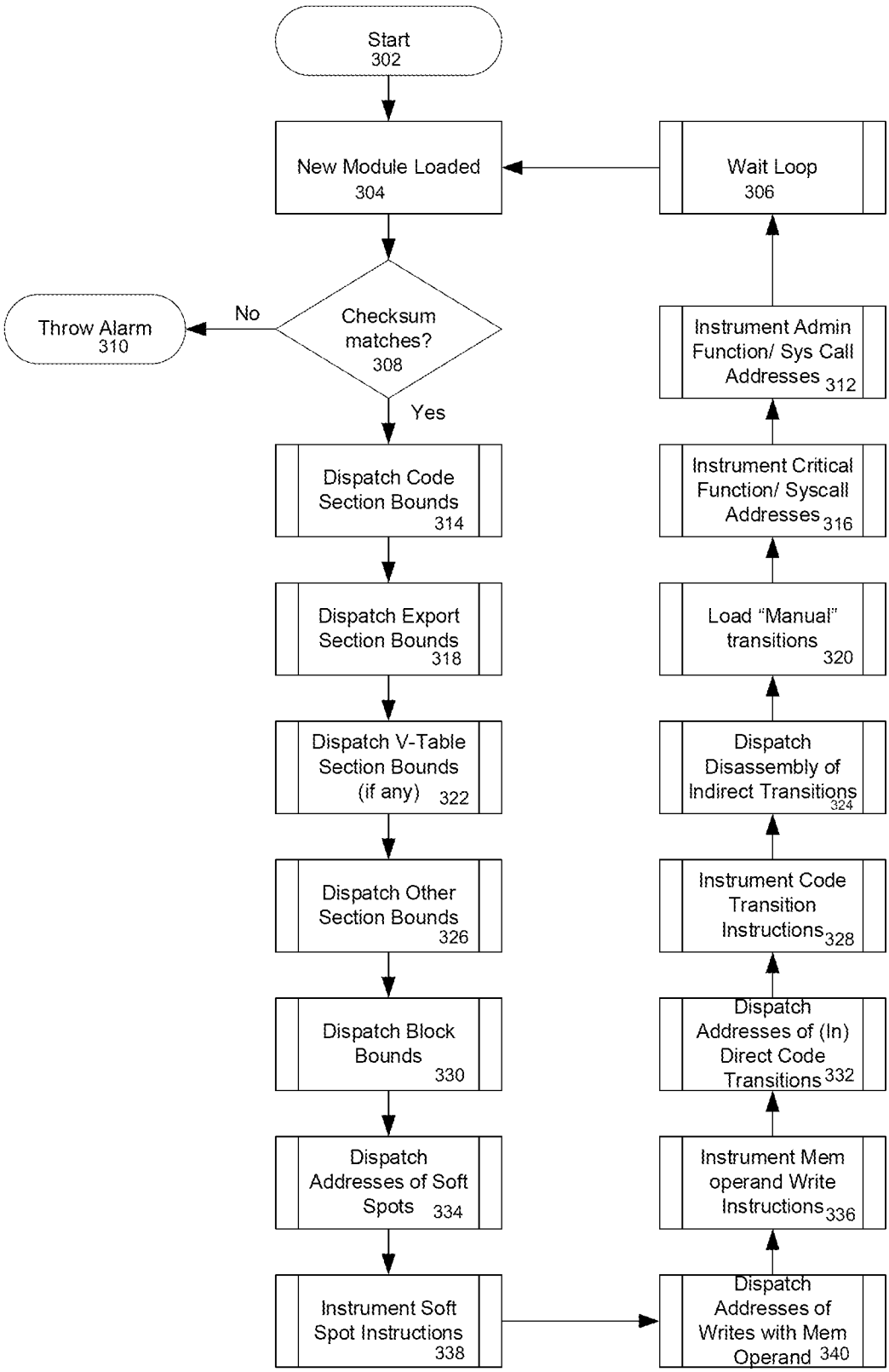
Start
302
New Module Loaded
304
Wait Loop
306
Checksum matches?
308
No
Throw Alarm
310
Yes
Instrument Admin Function/ Sys Call Addresses
312
Dispatch Code Section Bounds
314
Instrument Critical Function/ Syscall Addresses
316
Dispatch Export Section Bounds
318
Load "Manual" transitions
320
Dispatch V-Table Section Bounds (if any)
322
Dispatch Disassembly of Indirect Transitions
324
Dispatch Other Section Bounds
326
Instrument Code Transition Instructions
328
Dispatch Block Bounds
330
Dispatch Addresses of (In) Direct Code Transitions
332
Dispatch Addresses of Soft Spots
334
Instrument Mem operand Write Instructions
336
Instrument Soft Spot Instructions
338
Dispatch Addresses of Writes with Mem Operand
340

FIG. 3

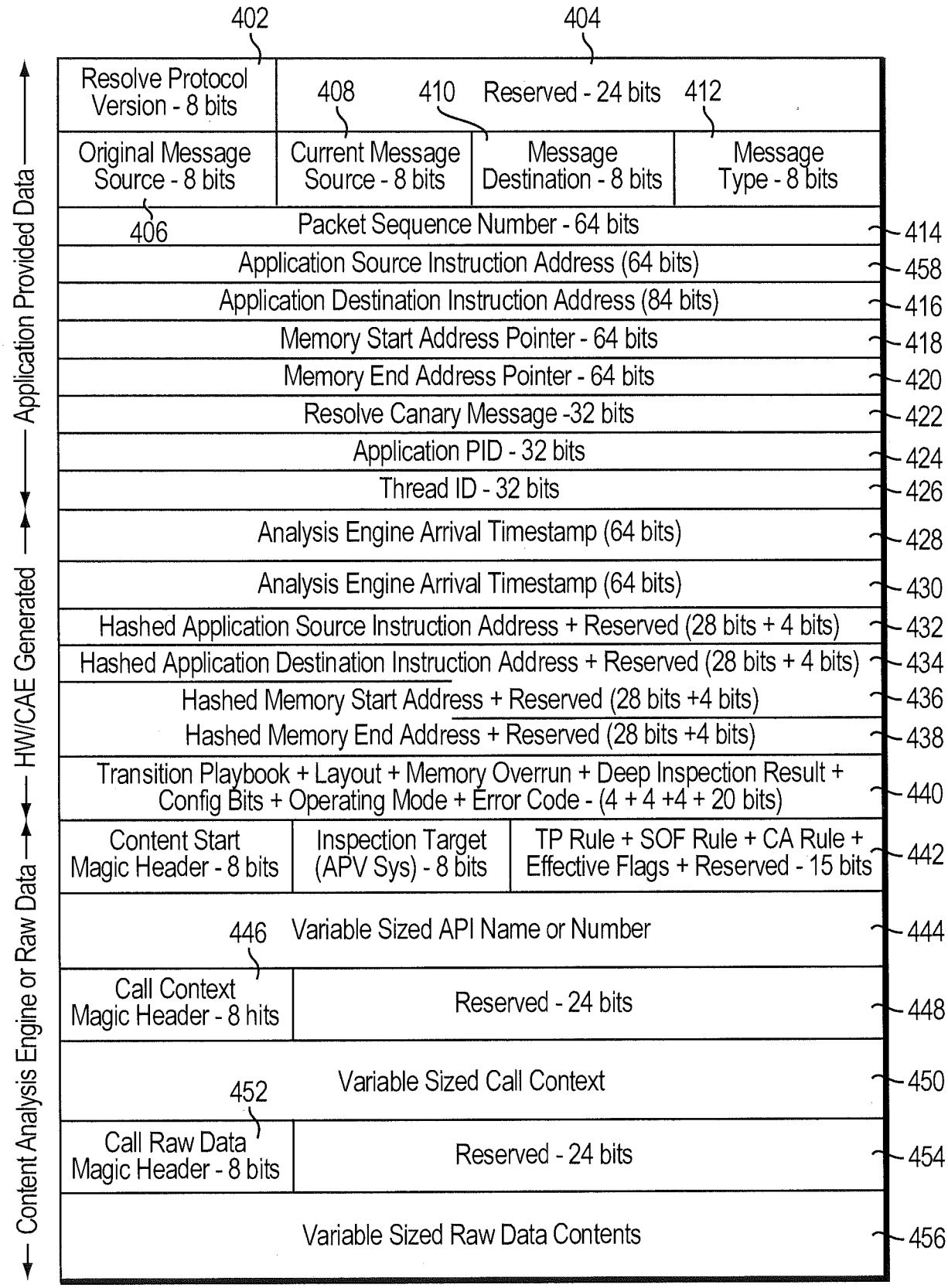
402
404
Resolve Protocol Version - 8 bits
408
410
Reserved - 24 bits
412
Original Message Source - 8 bits
Current Message Source - 8 bits
Message Destination - 8 bits
Message Type - 8 bits
406
Packet Sequence Number - 64 bits
414
Application Source Instruction Address (64 bits)
458
Application Destination Instruction Address (84 bits)
416
Memory Start Address Pointer - 64 bits
418
Memory End Address Pointer - 64 bits
420
Resolve Canary Message -32 bits
422
Application PID - 32 bits
424
Thread ID - 32 bits
426
Analysis Engine Arrival Timestamp (64 bits)
428
Analysis Engine Arrival Timestamp (64 bits)
430
Hashed Application Source Instruction Address + Reserved (28 bits + 4 bits)
432
Hashed Application Destination Instruction Address + Reserved (28 bits + 4 bits)
434
Hashed Memory Start Address + Reserved (28 bits +4 bits)
436
Hashed Memory End Address + Reserved (28 bits +4 bits)
438
Transition Playbook + Layout + Memory Overrun + Deep Inspection Result + Config Bits + Operating Mode + Error Code - (4 + 4 +4 + 20 bits)
440
Content Start Magic Header - 8 bits
Inspection Target (APV Sys) - 8 bits
TP Rule + SOF Rule + CA Rule + Effective Flags + Reserved - 15 bits
442
446
Variable Sized API Name or Number
444
Call Context Magic Header - 8 hits
Reserved - 24 bits
448
452
Variable Sized Call Context
450
Call Raw Data Magic Header - 8 bits
Reserved - 24 bits
454
Variable Sized Raw Data Contents
456
Application Provided Data
HW/CAE Generated
Content Analysis Engine or Raw Data

FIG. 4

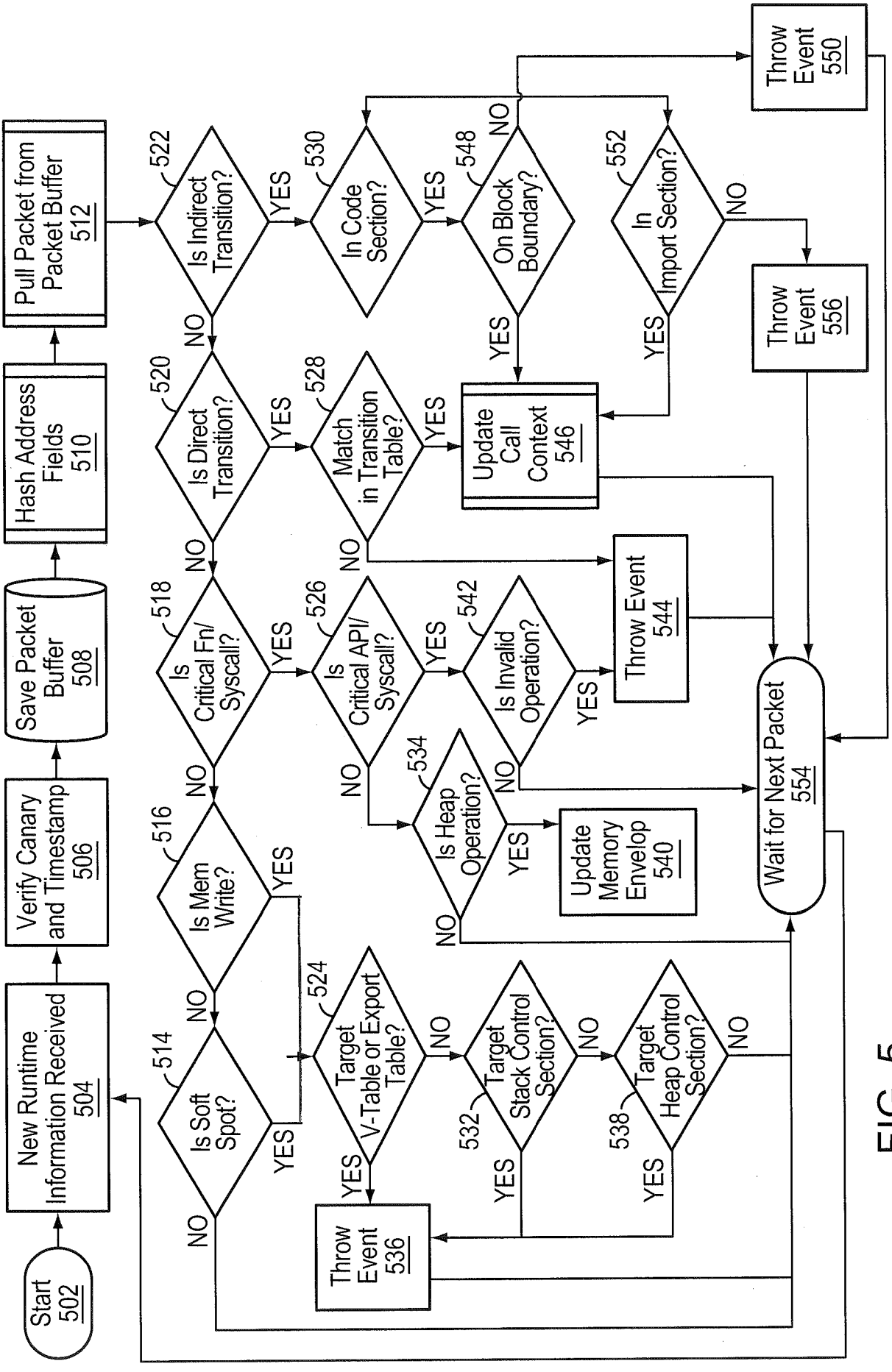
Start 502
New Runtime Information Received 504
Verify Canary and Timestamp 506
Save Packet Buffer 508
Hash Address Fields 510
Pull Packet from Packet Buffer 512
Is Soft Spot? 514
Is Mem Write? 516
Is Critical Fn/ Syscall? 518
Is Direct Transition? 520
Is Indirect Transition? 522
Target V-Table or Export Table? 524
Is Critical API/ Syscall? 526
Match in Transition Table? 528
In Code Section? 530
Target Stack Control Section? 532
Is Heap Operation? 534
Target Heap Control Section? 538
Update Memory Envelop 540
Is Invalid Operation? 542
Throw Event 544
Update Call Context 546
On Block Boundary? 548
Throw Event 550
In Import Section? 552
Throw Event 556
Wait for Next Packet 554
Throw Event 536
YES
NO

FIG. 5

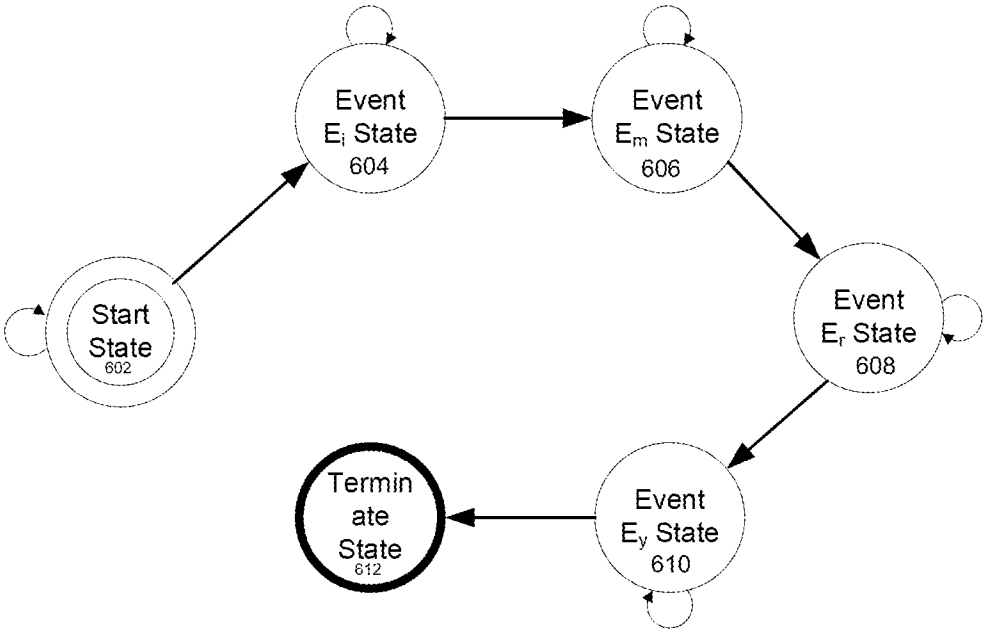
Start State 602
Event E_i State 604
Event E_m State 606
Event E_r State 608
Event E_y State 610
Termin ate State 612

FIG. 6

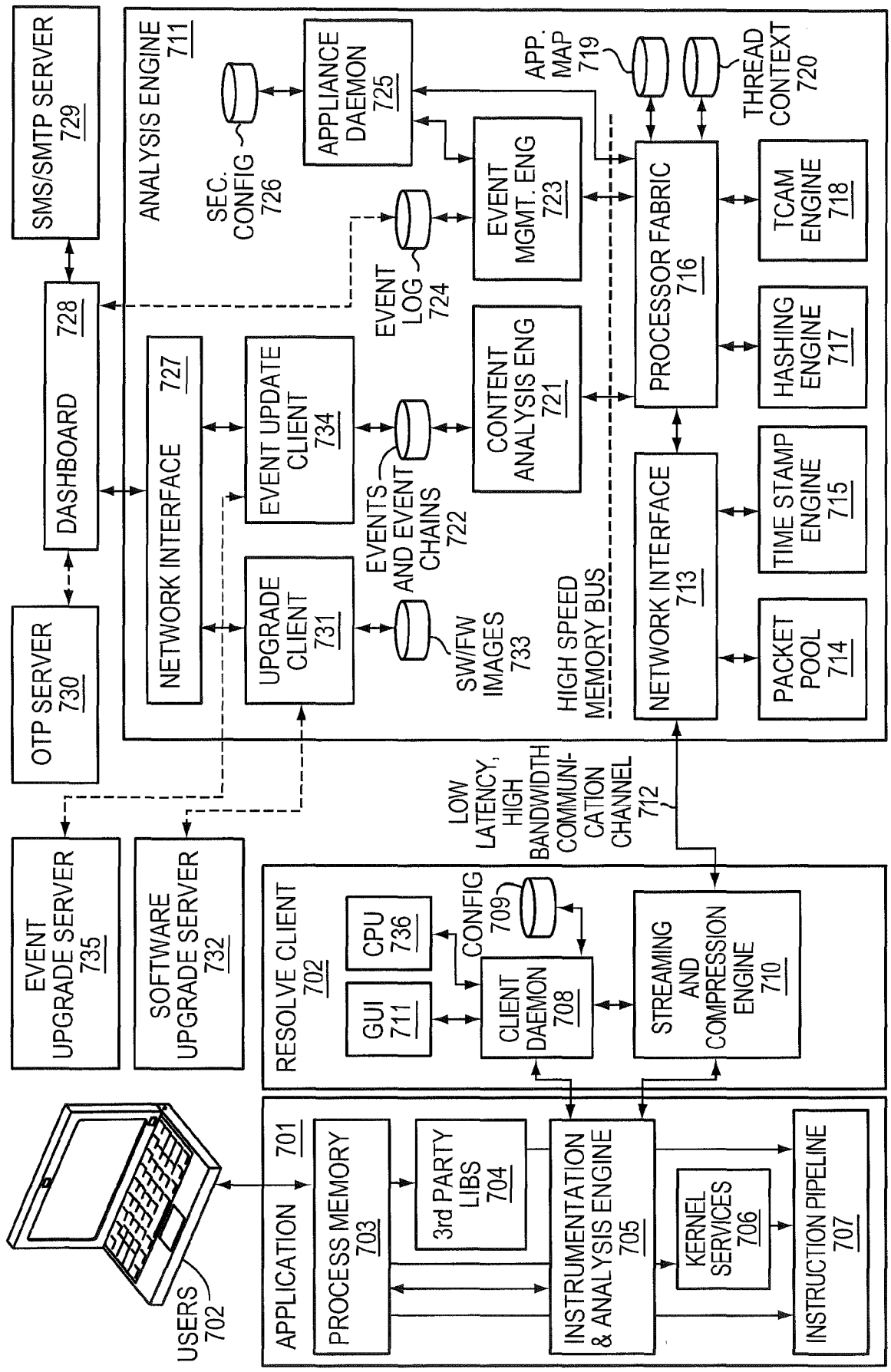
SMS/SMTP SERVER 729
ANALYSIS ENGINE 711
APPLIANCE DAEMON 725
APP. MAP 719
THREAD CONTEXT 720
SEC. CONFIG 726
EVENT MGMT. ENG 723
EVENT LOG 724
PROCESSOR FABRIC 716
TCAM ENGINE 718
DASHBOARD 728
NETWORK INTERFACE 727
EVENT UPDATE CLIENT 734
CONTENT ANALYSIS ENG 721
HASHING ENGINE 717
EVENTS AND EVENT CHAINS 722
OTP SERVER 730
UPGRADE CLIENT 731
SW/FW IMAGES 733
HIGH SPEED MEMORY BUS
NETWORK INTERFACE 713
TIME STAMP ENGINE 715
PACKET POOL 714
LOW LATENCY, HIGH BANDWIDTH COMMUNI-CATION CHANNEL 712
EVENT UPGRADE SERVER 735
SOFTWARE UPGRADE SERVER 732
RESOLVE CLIENT 702
CPU 736
CONFIG 709
GUI 711
CLIENT DAEMON 708
STREAMING AND COMPRESSION ENGINE 710
USERS 702
APPLICATION 701
PROCESS MEMORY 703
3rd PARTY LIBS 704
INSTRUMENTATION & ANALYSIS ENGINE 705
KERNEL SERVICES 706
INSTRUCTION PIPELINE 707

FIG. 7

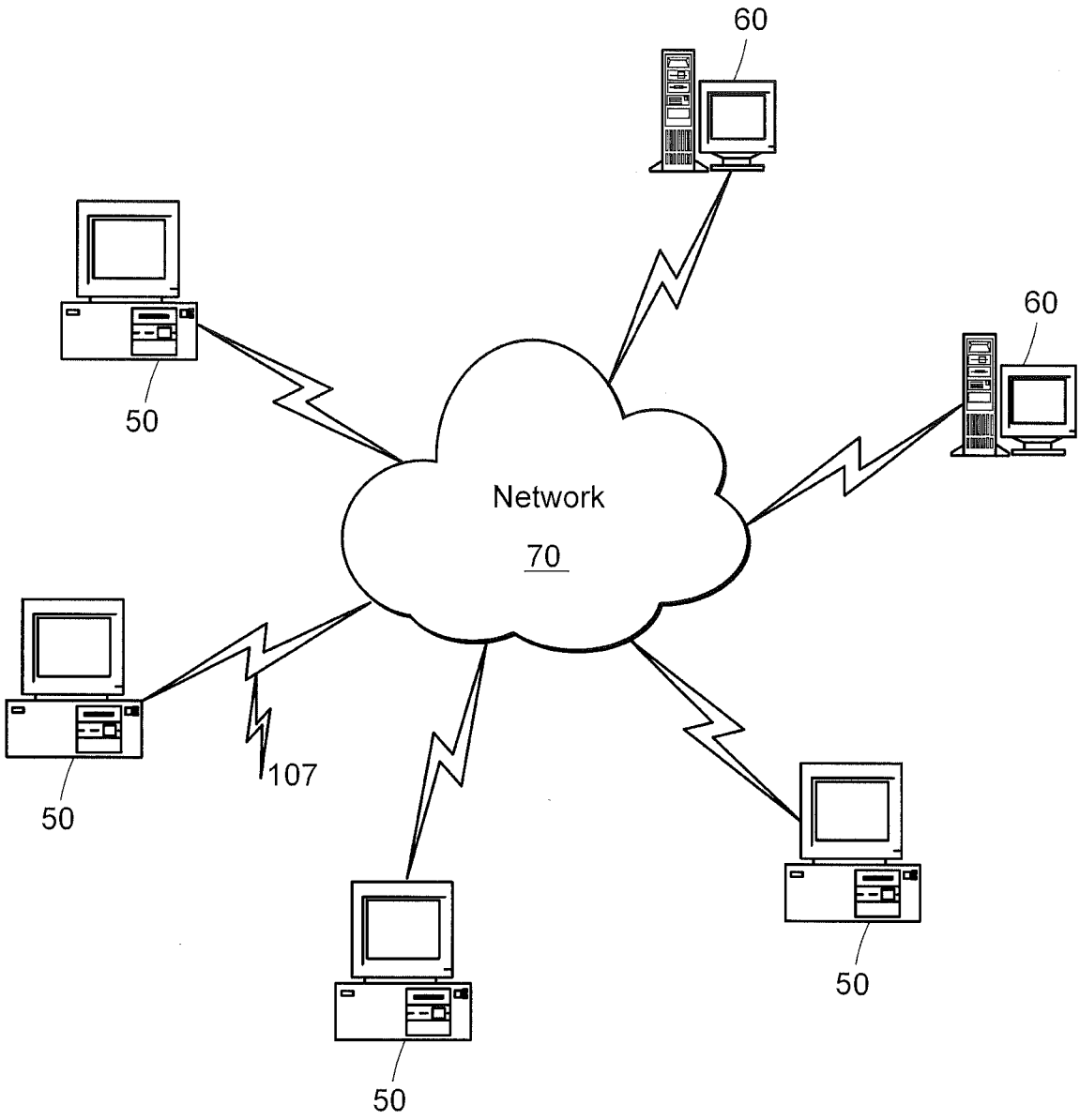

FIG. 8

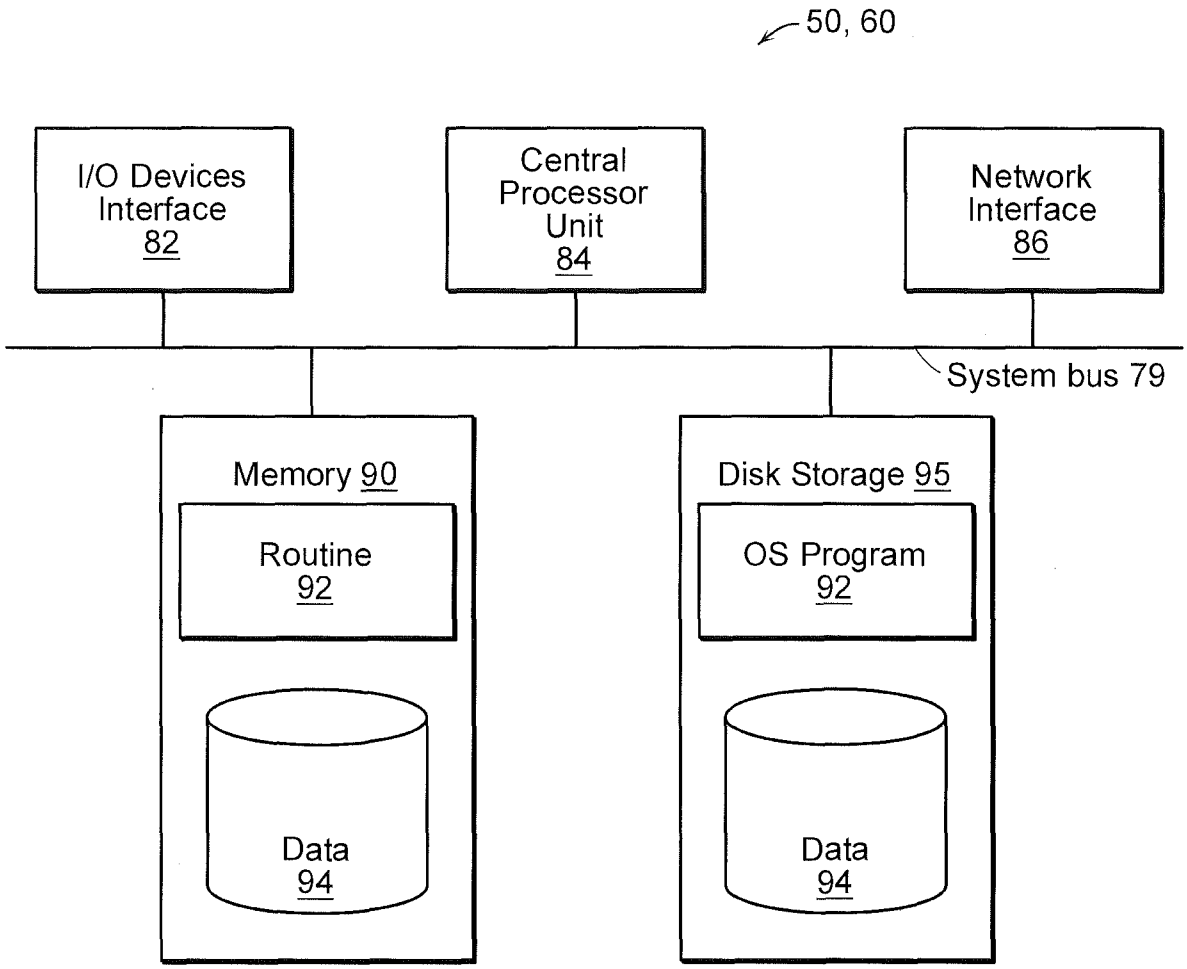
50, 60
I/O Devices Interface 82
Central Processor Unit 84
Network Interface 86
System bus 79
Memory 90
Routine 92
Data 94
Disk Storage 95
OS Program 92
Data 94

FIG. 9