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

(21) International Application Number:
PCT/US2015/012495

(22) International Filing Date:
22 January 2015 (22.01.2015)

(25) Filing Language:
English

(26) Publication Language:
English


(72) Inventors:
Mehta, Kunal [IN/US]; 2174 NW Thorton Drive, Apartment No. 1034, Hillsboro, Oregon 97124 (US).
Singh, Balbir [IN/IN]; Tower 3/301, Sushant Estate, Sector 52, Gurgaon 122002 (IN).
Bhattacharjee, Rajbir [IN/IN]; C-126, Sushant Apartments, Sushant Lok 1, Haryana, Gurgaon 122003 (IN).

(74) Agent: PEMBERTON, John D.; Patent Capital Group, c/o CPA Global, 900 Second Avenue South, Minneapolis, MN 55402 (US).


(54) Title: DETECTION OF MALICIOUS INVOCATION OF APPLICATION PROGRAM INTERFACE CALLS

(57) Abstract: Particular embodiments described herein provide for an electronic device that includes a binder kernel driver. The binder kernel driver can be configured to receive an application program interface (API) call, extract metadata from the API call, determine that the API call should be hooked based on the extracted metadata, and hook the API call.

(Continued on next page)
Declarations under Rule 4.17:

- of inventorship (Rule 4.17(iv))

Published:

- with international search report (Art. 21(3))
DETECTION OF MALICIOUS INVOCATION OF APPLICATION PROGRAM: INTERFACE CALLS

TECHNICAL FIELD

[0001] This disclosure relates in general to the field of information security, and more particularly, to detection of malicious invocation of application program interface calls.

BACKGROUND

[0002] The field of network security has become increasingly important in today's society. The Internet has enabled interconnection of different computer networks all over the world. In particular, the Internet provides a medium for exchanging data between different users connected to different computer networks via various types of client devices. While the use of the Internet has transformed business and personal communications, it has also been used as a vehicle for malicious operators to gain unauthorized access to computers and computer networks and for intentional or inadvertent disclosure of sensitive information.

[0003] Malicious software ("malware") that infects a host computer may be able to perform any number of malicious actions, such as stealing sensitive information from a business or individual associated with the host computer, propagating to other host computers, and/or assisting with distributed denial of service attacks, sending out spam or malicious emails from the host computer, etc. Hence, significant administrative challenges remain for protecting computers and computer networks from malicious and inadvertent exploitation by malicious software and devices.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] To provide a more complete understanding of the present disclosure and features and advantages thereof, reference is made to the following description, taken in conjunction with the accompanying figures, wherein like reference numerals represent like parts, in which;
FIGURE 1 is a simplified block diagram of a communication system for the detection of malicious invocation of application program interface calls in accordance with an embodiment of the present disclosure;

FIGURE 2 is a simplified diagram of a portion of a communication system for the detection of malicious invocation of application program interface calls in accordance with an embodiment of the present disclosure;

FIGURE 3 is a simplified flowchart illustrating potential operations that may be associated with the communication system in accordance with an embodiment;

FIGURE 4 is a simplified flowchart illustrating potential operations that may be associated with the communication system in accordance with an embodiment;

FIGURE 5A is a simplified flowchart illustrating potential operations that may be associated with the communication system in accordance with an embodiment;

FIGURE 5B is a simplified flowchart illustrating potential operations that may be associated with the communication system in accordance with an embodiment;

FIGURE 6 is a block diagram illustrating an example computing system that is arranged in a point-to-point configuration in accordance with an embodiment;

FIGURE 7 is a simplified block diagram associated with an example ARM ecosystem system on chip (SOC) of the present disclosure; and

FIGURE 8 is a block diagram illustrating an example processor core in accordance with an embodiment.

The FIGURES of the drawings are not necessarily drawn to scale, as their dimensions can be varied considerably without departing from the scope of the present disclosure.

DETAILED DESCRIPTION OF EXAMPLE EMBODIMENTS

EXAMPLE EMBODIMENTS

FIGURE 1 is a simplified block diagram of a communication system 100 for the detection of malicious invocation of application program interface calls in accordance with an embodiment of the present disclosure. As illustrated in FIGURE 1, an embodiment of communication system 100 can include electronic device 102, a server 104, and a cloud 106.
Electronic device 102a can include a processor 110a, a security module 112, memory 114a, and an application 116. Memory 114a can include an application space 118 and a kernel space 120. Application space 118 can include a transaction control library 128 and a middleware library 130. Kernel space 120 can include a kernel driver 132. Kernel driver 132 can include a whitelist 122, a blacklist 124, and security policies 126. Server 104 can include a processor 110b, memory 114b, and a network security module 134a. Memory 114b can include whitelist 122, blacklist 124, and security policies 126. Cloud 106 can include a processor 110c, memory 114c, and a network security module 134b. Memory 114c can include whitelist 122, blacklist 124, and security policies 126. Electronic device 102, server 104, and cloud 106 can be in communication using network 108.

[0016] In example embodiments, communication system 100 can be configured to hook or access control any Android’s API invocation by modifying the binder driver in the Android OS. Before reaching the actual implementation service, all the API calls flow through the binder kernel driver. Communication system 100 can be configured to hook and provide access control of Android APIs invocation for before the API call reaches its actual implementation service. The system includes a generic framework which provides a software development kit to hook or access control any given Android system service API or binder transactions. Other academic implementations just target one attack or a particular use case and do not include a generic way of hooking to build solutions against multiple attacks.

[0017] Elements of FIGURE 1 may be coupled to one another through one or more interfaces employing any suitable connections (wired or wireless), which provide viable pathways for network (e.g., network 108) communications. Additionally, any one or more of these elements of FIGURE 1 may be combined or removed from the architecture based on particular configuration needs. Communication system 100 may include a configuration capable of transmission control protocol/Internet protocol (TCP/IP) communications for the transmission or reception of packets in a network. Communication system 100 may also operate in conjunction with a user datagram protocol/IP (UDP/IP) or any other suitable protocol where appropriate and based on particular needs.

[0018] For purposes of illustrating certain example techniques of communication system 100, it is important to understand the communications that may be traversing the
network environment. The following foundational information may be viewed as a basis from which the present disclosure may be properly explained.

[0019] Android is a mobile operating system (OS) based on a Linux kernel. Currently, the Android OS is the leading mobile operating system in the market with a wide presence among consumers. The popularity of the Android OS means that it is often a target of malicious operators. One common technique used by malicious operators is to exploit the legitimate exposed APIs of the Android framework. Currently, there is not a framework available which can allow a security system to hook and access control Android's system service APIs invocation or binder transactions generally. What is needed is a security solution that provides a generic framework by means of policies to dynamically hook into any given Android framework's API. In an Android OS, the binder is the official IPC communication mechanism of Android which is internally used by all the Android framework APIs and all (or almost all) of the Android's API invocation goes through the binder driver;

[0020] A communication system for detection of malicious invocation of application program interface calls, as outlined in Figure 1, can resolve these issues (and others). Communication system 100 may be configured to provide a generic policy driven framework to build security apps for hooking or access controlling Android system service APIs invocation or binder transactions to achieve access control or stopping a malicious attack. Android framework API's are implemented inside multiple services and these services are hosted inside a special daemon process called system server or context manager. An application calling an Android API gets a handle to remote services and calls exposed functions on the system server or context manager. The API calls are then marshalled to the actual service being hosted inside the system server via a binder inter-process communication which is implemented inside Android's binder kernel driver.

[0021] In a specific example, the framework is implemented by modifying Android's binder kernel driver. Inside the driver, the existing handling of BINDER_READ_WRITE ioctl's BC_TRANSACTION command can be modified. This is the IOCTL which is called from user space whenever an Android system service API is called and from here, binder driver further dispatches a call to actual destination service inside the system server in user space. Before the binder kernel driver passes the API calls to actual system services, the arguments of the API call can be unpacked and the name of the service and function being invoked can be
acquired. Also, the process details invoking the calls can be acquired and the details can be filtered through a security module (e.g., security module 112 or network security module 134a or 134b).

[0022] The security module can specify policies in terms of attributes for filtering the call. For example the attributes can include, but are not limited to a service name, function name, action to be taken, etc. Actions can be supported such as allow the API call, block or deny the API call, block the API call for few seconds, and in same context notify the security module with details of the API call like invoking process details, service name, function being invoked and argument, so the security module can further analyse the arguments or send the calls to some other framework for further scanning and return the verdict of allow or block to the kernel framework. The allow and block actions allow for access control whereas the ask action allows the system to simulate the API hooking. When an allow or block decision is finally made, the event can be raised to the security module about the decision with metadata details like, action performed, target API details and process details which invoked the API. The decision and metadata details can be used by a security system to create or suppress whitelist 122, blacklist 124, and generally improve the detection of malicious activities.

[0023] Communication system 100 can be configured to build policy driven applications to support various BYOD use cases (e.g., allowing and denying various operations). Communication system 100 can also be configured to help stop attacks like activity phishing (hooking into start ActivityO, pauseActivityl), resume Activityl) or all kinds of Activity lifecycle APIs), In addition, communication system 100 can be configured to help stop attacks like eavesdropping on messages and contacts. Further, communication system 100 can be configured to help stop attacks related to data loss like silent message uploads to CMC server or silent messages ending to CNC server (hooking into send Message() or send Data() APIs). The system can build a strong intent firewall to control the ways intents are getting broadcasted within the system (e.g., hooking into broadcastIntent() API). Communication system 100 can also be configured to hook into Android's packaging APIs to try to prevent installation of malicious applications or deletion of a security application by a rouge application (e.g., hooking into installPackage() or installPackageAsUserQ APIs). The system can attempt to prevent any rouge app to stop the service of a security application hooking into a startServiceO, stopService(), bindServiceO or all kinds of service lifecycle APIs, in
addition, communication system 100 can be configured to help prevent permission re-
delegation attacks. White list 122 and blacklist 124 can be used to restrict or access control various operations by a given applications. Communication system 100 can be configured to help prevent against user interface (UI) state inference kinds of attacks by stopping rouge application from doing targeted activity phishing. Further, communication system 100 can be configured to hook into activity Lifecycle APIs and when a privilege application is detected (e.g., a banking application is starting) log any inactivity and only allow start Activity!) from known legitimate applications.

[0024] Turning to the infrastructure of FIGURE 1, communication system 100 in accordance with an example embodiment is shown. Generally, communication system 100 can be implemented in any type or topology of networks. Network 108 represents a series of points or nodes of interconnected communication paths for receiving and transmitting packets of information that propagate through communication system 100, Network 108 offers a communicative interface between nodes, and may be configured as any local area network (LAN), virtual local area network (VLAN), wide area network (WAN), wireless local area network (WLAN), metropolitan area network (MAN), intranet. Extranet, virtual private network (VPN), and any other appropriate architecture or system that facilitates communications in a network environment, or any suitable combination thereof, including wired and/or wireless communication.

[0025] In communication system 100, network traffic, which is inclusive of packets, frames, signals, data, etc., can be sent and received according to any suitable communication messaging protocols. Suitable communication messaging protocols can include a multi-
layered scheme such as Open Systems interconnection (OSI) model, or any derivations or variants thereof (e.g., Transmission Control Protocol/Internet Protocol (TCP/IP), user datagram protocol/IP (UDP/IP)). Additionally, radio signal communications over a cellular network may also be provided in communication system 100. Suitable interfaces and infrastructure may be provided to enable communication with the cellular network.

[0026] The term "packet" as used herein, refers to a unit of data that can be routed between a source node and a destination node on a packet switched network. A packet includes a source network address and a destination network address. These network addresses can be Internet Protocol (IP) addresses in a TCP/IP messaging protocol. The term
"data" as used herein, refers to any type of binary, numeric, voice, video, textual, or script data, or any type of source or object code, or any other suitable information in any appropriate format that may be communicated from one point to another in electronic devices and/or networks. Additionally, messages, requests, responses, and queries are forms of network traffic, and therefore, may comprise packets, frames, signals, data, etc.

[0027] In an example implementation, electronic device 102, server 104, and cloud 106 are network elements, which are meant to encompass network appliances, servers, routers, switches, gateways, bridges, load balancers, processors, modules, or any other suitable device, component, element, or object operable to exchange information in a network environment. Network elements may include any suitable hardware, software, components, modules, or objects that facilitate the operations thereof, as well as suitable interfaces for receiving, transmitting, and/or otherwise communicating data or information in a network environment. This may be inclusive of appropriate algorithms and communication protocols that allow for the effective exchange of data or information.

[0028] In regards to the internal structure associated with communication system 100, each of electronic device 102, server 104, and cloud 106 can include memory elements for storing information to be used in the operations outlined herein. Each of electronic device 102, server 104, and cloud 106 may keep information in any suitable memory element (e.g., random access memory (RAM), read-only memory (ROM), erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM), application specific integrated circuit (ASIC), etc.), software, hardware, firmware, or in any other suitable component, device, element, or object where appropriate and based on particular needs. Any of the memory items discussed herein should be construed as being encompassed within the broad term 'memory element'. Moreover, the information being used, tracked, sent, or received in communication system 100 could be provided in any database, register, queue, table, cache, control list, or other storage structure, all of which can be referenced at any suitable timeframe. Any such storage options may also be included within the broad term 'memory element' as used herein.

[0029] In certain example implementations, the functions outlined herein may be implemented by logic encoded in one or more tangible media (e.g., embedded logic provided in an ASIC, digital signal processor (DSP) instructions, software (potentially inclusive of object
code and source code) to be executed by a processor, or other similar machine, etc.), which may be inclusive of non-transitory computer-readable media. In some of these instances, memory elements can store data used for the operations described herein. This includes the memory elements being able to store software, logic, code, or processor instructions that are executed to carry out the activities described herein.

[0030] In an example implementation, network elements of communication system 100, such as electronic device 102, server 104, and cloud 106 may include software modules (e.g., security module 112 and network security modules 134a and 134b) to achieve, or to foster, operations as outlined herein. These modules may be suitably combined in any appropriate manner, which may be based on particular configuration and/or provisioning needs. In example embodiments, such operations may be carried out by hardware, implemented externally to these elements, or included in some other network device to achieve the intended functionality. Furthermore, the modules can be implemented as software, hardware, firmware, or any suitable combination thereof. These elements may also include software (or reciprocating software) that can coordinate with other network elements in order to achieve the operations, as outlined herein.

[0031] Additionally, each of electronic device 102, server 104, and cloud 106 may include a processor that can execute software or an algorithm to perform activities as discussed herein. A processor can execute any type of instructions associated with the data to achieve the operations detailed herein. In one example, the processors could transform an element or an article (e.g., data) from one state or thing to another state or thing. In another example, the activities outlined herein may be implemented with fixed logic or programmable logic (e.g., software/computer instructions executed by a processor) and the elements identified herein could be some type of a programmable processor, programmable digital logic (e.g., a field programmable gate array (FPGA), an EPROM, an EEPROM) or an ASIC that includes digital logic, software, code, electronic instructions, or any suitable combination thereof. Any of the potential processing elements, modules, and machines described herein should be construed as being encompassed within the broad term 'processor'.

[0032] Electronic device 102 can each be a network element that runs an Android OS and includes, for example, desktop computers, laptop computers, mobile devices, personal digital assistants, smartphones, tablets, or other similar devices. Server 104 can be a network
element such as a server or virtual server and can be associated with clients, customers, endpoints, or end users wishing to initiate a communication in communication system 100 via some network (e.g., network 108). The term 'server' is inclusive of devices used to serve the requests of clients and/or perform some computational task on behalf of clients within communication system 100. Although security module 112 is represented in FIGURE 1 as being located in electronic device 102, this is for illustrative purposes only. Security module 112 could be combined or separated in any suitable configuration. Furthermore, security module 112 could be integrated with or distributed in another network accessible by electronic device 102 such as server 104 or cloud 106. Cloud 106 is configured to provide cloud services to electronic device 102. Cloud services may generally be defined as the use of computing resources that are delivered as a service over a network, such as the Internet. Typically, compute, storage, and network resources are offered in a cloud infrastructure, effectively shifting the workload from a local network to the cloud network.

[0033] Turning to FIGURE 2, FIGURE 2 is a simplified block diagram of a portion of a communication system 100 for the detection of malicious invocation of application program interface calls in accordance with an embodiment of the present disclosure. As illustrated in FIGURE 2, electronic device 102 can include application space 118 and kernel space 120. Application space 118 can include transaction control library 128 and middleware library 130. Transaction control library 128 can include a policy handler 136, a notification handler 138, a kernel events handler 140, and a transaction handler 142. Middleware library 130 can include policies 150, notifications 152, events 154, an input/output control (ioctl) dispatcher 156, and a netlink socket 158. Kernel space 120 can include kernel driver 132. Kernel driver 132 can include whitelist 122, blacklist 124, security policies 126, an ioctl handler 160, a netlink socket layer 162, a policy manager 166, a kernel events manager 168, and a read/write ioctl interceptor 170.

[0034] Transaction control library 128 is the main Java library that exposes all the functionality of the system and interacts with the middleware library 130 whenever any exposed APIs are called. Transaction control library 128 can provides API for policy management, listening to framework kernel events, etc. Policy handler 136 can have functionality related to policy configuration or policy management. Policy handler 136 can be configured to exposes the APIs for policy management and redirect the calls to policies 150
which in-turn can call policy manager 164. Notification handler 13S can pull allow or block verdict events from notification manager 166 using notifications 152. Kernel events handler 140 can receive sync and async callbacks from netiink socket 158. The sync and async callbacks can be generated by kernel events manager 168.

[0035] Because the APIs are provide in Java, for Java to call kernel mode code middleware library 130 can be configured to act as a bridge or middleware between transaction control library 128 and kernel driver 132. Policies 150 indicate the code and functionality for preprocessing before sending a call to policy manager 164. Notifications 152 can be configured to interact with notification manager 166. Events 154 can call and receive callbacks from kernel event manager 168 using netiink socket 158. IOCTL dispatcher 156 can use exposed IOCTL code for the exposed API and other related functionality of the framework. Netiink socket 158 can be configured to listen over the socket to for the sync and async callback from netiink socket layer 162.

[0036] Whitelist 122 can include entries of known clean or trusted applications, code, strings, etc. Blacklist 124 can include entries of known malicious or untrusted applications, code, strings, etc. Security policies 126 can includes policies on how to handle identified or suspect malware. In a specific example, the format of the policy can include a source and a target. The source can include a list of processes that are invoking the system APIs or process that should be monitored. The target can include the APIs being invoked (e.g., API interface name and functions) or APIs that are being monitored for source applications. All the APIs for the system are exposed by means of IOCTL and IOCTL handler 160 can provide an entry point for transaction control library to call into kernel driver 132. Netiink socket layer 162 can be used by kernel events manager 168 to send sync and async callbacks to transaction control library 128. Netiink socket layer 162 can communicate over Linux sockets between the kernel and the user mode. Policy manager 164 can be configured to provide functionality to maintain policies inside memory in lookup-able data structures, configured for API access controlling and hooking. Policy manager 164 can also takes care of matching given APIs and process metadata with configured policies and provide the decision on how to handle the APIs (e.g., allow-block-hook). Notification manager 166 can be configured to provide functionality to add allow or block notifications when a decision to allow or block an API call is made by policy manager 164. Notification manager 166 can maintain the notification inside
A queue in memory so that security module 112 can pull all these notifications using transaction control library. Kernel events manager 168 can be configured to provide functionality to send callbacks from kernel driver 132 to user mode (e.g., transaction control library 128) and allow kernel driver 132 to send synchronous and asynchronous callbacks to transaction control library. Read/write IOCTL interceptor 170 can be configured to intercept Binder, READ, WRITE IOCTL of the binder driver (where the API call comes from) and extract the API and caller app and other related metadata for further processing.

[0037] Along with read/write IOCTL interceptor 170 inside kernel driver 132, the system can also include components for keeping policies in memory, managing the decision events in memory and components to call back user mode (e.g., an ASK callback). Transaction control library 128, can specifying policies in terms of service name and function name, and keep some metadata about all the Android system service APIs and their arguments type, so that while getting ASK call back, the security module 112 can read the invoked APIs arguments and also specify a policy in terms of a service name and a function name.

[0038] Turning to FIGURE 3, FIGURE 3 is an example flowchart illustrating possible operations of a flow 300 that may be associated with the detection of malicious invocation of application program interface calls, in accordance with an embodiment. At 302, an application calls an API. At 304, the API call is communicated to a kernel driver. At 306, metadata related to the API call is extracted. At 308, the system determines if, based on the extracted metadata, the API call should be allowed. If the API call should not be allowed, then the API call is not allowed, as in 310, if the API call should be allowed, then the system determines, based on the extracted metadata, if the API call should be hooked, as in 312. If the API call should not be hooked, then the API call is allowed without hooking, as in 314. If the API call should be hooked, then the API call is hooked, as in 316.

[0039] Turning to FIGURE 4, FIGURE 4 is an example flowchart illustrating possible operations of a flow 400 that may be associated with the detection of malicious invocation of application program interface calls, in accordance with an embodiment. At 402, an application calls an API. At 404, the API call is communicated to a kernel driver. At 406, an interceptor in the kernel driver intercepts the API call and metadata related to the API call is extracted. At 408, the system determines if the API or the metadata or both match a security policy. If neither the API or the metadata match a security policy, then the API call is allowed,
as in 410. If the API or the metadata or both match a security policy, then the API is processed according to the security policy, as in 412.

[0040] Turning to FIGURES 5A and 5B, FIGURES 5A and 5B are an example flowchart illustrating possible operations of a flow 500 that may be associated with the detection of malicious invocation of application program interface calls, in accordance with an embodiment. At 502, an application calls or invokes an API. At 504, the API call is communicated to a kernel driver. At 506, an interceptor located in the kernel driver intercepts the API call and extracts API and invoking application related data. At 508, the API and invoking application related metadata are compared to and matched with security policies. At 510, the system determines if a security policy match as found. If a security policy match was not found, then the application call is allowed, as in 512. If a security policy match was found, then the security policy that matched the API and invoking application related metadata is invoked, as in 514.

[0041] At 516, the system determines if the API call should be blocked. For example, the security policy that matched the API and invoking application related metadata may indicate that the API should be blocked. If the API call should be blocked, then the API call is denied or blocked, as in 518 (see FIGURE 5B). At 520, an access denied code is returned to the application that called or invoked the API. At 522, a blocking notification along with the API and invoking application related metadata is added to the security system. For example, the API and invoking application related metadata may be sent to server 104 or cloud 106 to be used by the security system.

[0042] Going back to 516 (see FIGURE 5A), if the API call should not be blocked, then the system determines if the API call should be allowed, as in 524. If the API call should be allowed, then the API call is allowed, as in 526 (see FIGURE 5B). At 528, an allowed notification along with the API and invoking application related metadata is communicated to the security system. For example, the API and invoking application related metadata may be sent to server 104 or cloud 106 to be used by the security system.

[0043] Going back to 524 (see FIGURE 5A), if the API call should not be allowed, then the system determines if the API call should be hooked, as in 530. If the API call should not be hooked, then an error messages is generated, as in 530. If an API call should be hooked, then the API is blocked for a pre-determined amount of time, as in 532 (see FIGURE 5B). At
an in-band notification about the API call and the API and invoking application related metadata is communicated to the security system along with a request to allow the call. For example, notification about the API call and the API and invoking application related metadata can be communicated to security module 112 or network security module 134a or 134b, in an embodiment, an in-band notification about the API call, the API, and metadata related to the application invoking the API call is communicated to the security system and the kernel framework. At 536, the system determines if the pre-determined amount of time has expired. For example, the system may wait a predetermined number of cycles for a response from the security system, if the amount of time has expired, then an error message is generated as in 530. At 526, the API call is allowed, if the amount of time has not expired, then the system determines if a response to allow the API call was received, as in 538. If a response to allow the API call was received, then the API call is allowed, as in 526. If a response to allow the API call was not received, then the system determines if a response to block the API call was received, as in 540, if a response to block the API call was received, then the API call is denied or blocked, as in 518. If a response to block the API call was not received, then the system determines if the pre-determined amount of time has expired, as in 536.

[00441] Figure 6 illustrates a computing system 600 that is arranged in a point-to-point (PtP) configuration according to an embodiment, in particular, Figure 6 shows a system where processors, memory, and input/output devices are interconnected by a number of point-to-point interfaces. Generally, one or more of the network elements of communication system 600 may be configured in the same or similar manner as computing system 600.

100451 As illustrated in Figure 6, system 600 may include several processors, of which only two, processors 670 and 680, are shown for clarity. While two processors 670 and 680 are shown, it is to be understood that an embodiment of system 600 may also include only one such processor. Processors 670 and 680 may each include a set of cores (i.e., processor cores 674A and 674B and processor cores 684A and 684B) to execute multiple threads of a program. The cores may be configured to execute instruction code in a manner similar to that discussed above with reference to Figures 1-5. Each processor 670, 680 may include at least one shared cache 671, 681. Shared caches 671, 681 may store data (e.g., instructions)
that are utilized by one or more components of processors 670, 680, such as processor cores 674 and 684.

[0046] Processors 670 and 680 may also each include integrated memory controller logic (MC) 672 and 682 to communicate with memory elements 632 and 634. Memory elements 632 and/or 634 may store various data used by processors 670 and 680. In alternative embodiments, memory controller logic 672 and 682 may be discreet logic separate from processors 670 and 680.

[0047] Processors 670 and 680 may be any type of processor and may exchange data via a point-to-point (PtP) interface 650 using point-to-point interface circuits 678 and 688, respectively. Processors 670 and 680 may each exchange data with a chipset 690 via individual point-to-point interfaces 652 and 654 using point-to-point interface circuits 676, 686, 694, and 698. Chipset 690 may also exchange data with a high-performance graphics circuit 638 via a high-performance graphics interface 639, using an interface circuit 692, which could be a PtP interface circuit. In alternative embodiments, any or all of the PtP links illustrated in FIGURE 6 could be implemented as a multi-drop bus rather than a PtP link.

[0048] Chipset 690 may be in communication with a bus 620 via an interface circuit 696. Bus 620 may have one or more devices that communicate over it, such as a bus bridge 618 and I/O devices 616. Via a bus 610, bus bridge 618 may be in communication with other devices such as a keyboard/mouse 612 (or other input devices such as a touch screen, trackball, etc.), communication devices 626 (such as modems, network interface devices, or other types of communication devices that may communicate through a computer network 660), audio I/O devices 614, and/or a data storage device 628. Data storage device 628 may store code 630, which may be executed by processors 670 and/or 680. In alternative embodiments, any portions of the bus architectures could be implemented with one or more PtP links.

[0049] The computer system depicted in FIGURE 6 is a schematic illustration of an embodiment of a computing system that may be utilized to implement various embodiments discussed herein. It will be appreciated that various components of the system depicted in FIGURE 6 may be combined in a system-on-a-chip (SoC) architecture or in any other suitable configuration. For example, embodiments disclosed herein can be incorporated into systems including mobile devices such as smart cellular telephones, tablet computers, personal digital
assistants, portable gaming devices, etc. It will be appreciated that these mobile devices may be provided with SoC architectures in at least some embodiments.

[0050] Turning to FIGURE 7, FIGURE 7 is a simplified block diagram associated with an example ARM ecosystem SOC 700 of the present disclosure. At least one example implementation of the present disclosure can include the detection of malicious strings features discussed herein and an ARM component. For example, the example of FIGURE 7 can be associated with any ARM core (e.g., A-7, A-35, etc.). Further, the architecture can be part of any type of tablet, smartphone (inclusive of Android™ phones, iPhones™, iPad™, Google Nexus™, Microsoft Surface™, personal computer, server, video processing components, laptop computer (inclusive of any type of notebook), UStrabook™ system, any type of touch-enabled input device, etc.

[0051] In this example of FIGURE 7, ARM ecosystem SOC 700 may include multiple cores 706-707, an L2 cache control 708, a bus interface unit 709, an L2 cache 710, a graphics processing unit (GPU) 715, an interconnect 702, a video codec 720, and a liquid crystal display (LCD) 725, which may be associated with mobile industry processor interface (MIPI) high-definition multimedia interface (HDMI) links that couple to an LCD.

[0052] ARM ecosystem SOC 700 may also include a subscriber identity module (SIM) 730, a boot read-only memory (ROM) 735, a synchronous dynamic random access memory (SDRAM) controller 740, a flash controller 745, a serial peripheral interface (SPI) master 750, a suitable power control 755, a dynamic RAM (DRAM) 760, and flash 765. In addition, one or more example embodiments include one or more communication capabilities, interfaces, and features such as instances of Bluetooth™ 770, a 3G modem 775, a global positioning system (GPS) 780, and an 802.11 Wi-Fi 785.

[0053] In operation, the example of FIGURE 7 can offer processing capabilities, along with relatively low power consumption to enable computing of various types (e.g., mobile computing, high-end digital home, servers, wireless infrastructure, etc.). In addition, such an architecture can enable any number of software applications (e.g., Adobe® flash® Player, Java Platform Standard Edition (Java SE), JavaFX, Linux, Microsoft Windows Embedded, Symbian and Ubuntu, etc.). In at least one example embodiment, the core processor may implement an out-of-order superscalar pipeline with a coupled low-latency level-2 cache.
FIGURE S illustrates a processor core 800 according to an embodiment. Processor core 800 may be the core for any type of processor, such as a micro-processor, an embedded processor, a digital signal processor (DSP), a network processor, or other device to execute code. Although only one processor core 800 is illustrated in Figure 8, a processor may alternatively include more than one of the processor core 800 illustrated in Figure 8. For example, processor core 800 represents one example embodiment of processors cores 674a, 674b, 684a, and 684b shown and described with reference to processors 670 and 680 of FIGURE 6. Processor core 800 may be a single-threaded core or, for at least one embodiment, processor core 800 may be multithreaded in that it may include more than one hardware thread context (or 'logical processor") per core.

FIGURE 8 also illustrates a memory 802 coupled to processor core 800 in accordance with an embodiment. Memory 802 may be any of a wide variety of memories (including various layers of memory hierarchy) as are known or otherwise available to those of skill in the art. Memory 802 may include code 804, which may be one or more instructions, to be executed by processor core 800. Processor core 800 can follow a program sequence of instructions indicated by code 804. Each instruction enters a front-end logic 806 and is processed by one or more decoders 808. The decoder may generate, as its output, a micro operation such as a fixed width micro operation in a predefined format, or may generate other instructions, microinstructions, or control signals that reflect the original code instruction. Front-end logic 806 also includes register renaming logic 810 and scheduling logic 812, which generally allocate resources and queue the operation corresponding to the instruction for execution.

Processor core 800 can also include execution logic 814 having a set of execution units 816-1 through 816-N. Some embodiments may include a number of execution units dedicated to specific functions or sets of functions. Other embodiments may include only one execution unit or one execution unit that can perform a particular function. Execution logic 814 performs the operations specified by code instructions.

After completion of execution of the operations specified by the code instructions, back-end logic 818 can retire the instructions of code 804. In one embodiment, processor core 800 allows out of order execution but requires in order retirement of instructions. Retirement logic 820 may take a variety of known forms (e.g., re-order buffers
or the like). In this manner, processor core 800 is transformed during execution of code 804, at least in terms of the output generated by the decoder, hardware registers and tables utilized by register renaming logic 810, and any registers (not shown) modified by execution logic 814.

[0058] Although not illustrated in FIGURE 8, a processor may include other elements on a chip with processor core 800, at least some of which were shown and described herein with reference to FIGURE 6. For example, as shown in FIGURE 6, a processor may include memory control logic along with processor core 800. The processor may include i/O control logic and/or may include i/O control logic integrated with memory control logic.

[0059] Note that with the examples provided herein, interaction may be described in terms of two, three, or more network elements. However, this has been done for purposes of clarity and example only, in certain cases, it may be easier to describe one or more of the functionalities of a given set of flows by only referencing a limited number of network elements. It should be appreciated that communication system 100 and its teachings are readily scalable and can accommodate a large number of components, as well as more complicated/sophisticated arrangements and configurations. Accordingly, the examples provided should not limit the scope or inhibit the broad teachings of communication system 100 as potentially applied to a myriad of other architectures.

[0060] It is also important to note that the operations in the preceding flow diagrams (i.e., FIGURES 3-5B) illustrate only some of the possible correlating scenarios and patterns that may be executed by, or within, communication system 100. Some of these operations may be deleted or removed where appropriate, or these operations may be modified or changed considerably without departing from the scope of the present disclosure, in addition, a number of these operations have been described as being executed concurrently with, or in parallel to, one or more additional operations. However, the timing of these operations may be altered considerably. The preceding operational flows have been offered for purposes of example and discussion. Substantial flexibility is provided by communication system 100 in that any suitable arrangements, chronologies, configurations, and timing mechanisms may be provided without departing from the teachings of the present disclosure.

[0061] Although the present disclosure has been described in detail with reference to particular arrangements and configurations, these example configurations and arrangements
may be changed significantly without departing from the scope of the present disclosure. Moreover, certain components may be combined, separated, eliminated, or added based on particular needs and implementations. Additionally, although communication system 100 has been illustrated with reference to particular elements and operations that facilitate the communication process, these elements and operations may be replaced by any suitable architecture, protocols, and/or processes that achieve the intended functionality of communication system 100.

Numerous other changes, substitutions, variations, alterations, and modifications may be ascertained to one skilled in the art and it is intended that the present disclosure encompass all such changes, substitutions, variations, alterations, and modifications as falling within the scope of the appended claims, in order to assist the United States Patent and Trademark Office (USPTO) and, additionally, any readers of any patent issued on this application in interpreting the claims appended hereto, Applicant wishes to note that the Applicant: (a) does not intend any of the appended claims to invoke paragraph six (6) of 35 U.S.C section 112 as it exists on the date of the filing hereof unless the words "means for" or "step for" are specifically used in the particular claims; and (b) does not intend, by any statement in the specification, to limit this disclosure in any way that is not otherwise reflected in the appended claims.
CLAIMS:

1. At least one computer-readable medium comprising one or more instructions that when executed by at least one processor, cause the process to:
   - receive an application program interface (API) call at a kernel driver;
   - extract metadata from the API call;
   - determine that the API call should be hooked based on the extracted metadata; and
   - hook the API call.

2. The at least one computer-readable medium of Claim 1, wherein the kernel driver is a binder kernel driver.

3. The at least one computer-readable medium of Claim 2, wherein an interceptor in the kernel driver intercepts the API call.

4. The at least one computer-readable medium of Claim 2, wherein a policy manager inside the kernel driver determines that the API call should be hooked based on the extracted metadata.

5. The at least one computer-readable medium of Claim 2, wherein the extracted metadata is compared to a security policy to determine that the API call should be hooked.

6. The at least one computer-readable medium of Claim 5, wherein the security policy is located in the kernel driver.

7. The at least one computer-readable medium of Claim 2, further comprising one or more instructions that when executed by the at least one processor, further cause the processor to:
   - communicate the API call and the extracted metadata to a security module, wherein the security module determines if the API call should be allowed or denied,
8. The at least one computer-readable medium of Claim 7, further comprising one or more instructions that when executed by the at least one processor, further cause the processor to:

   allow the API call if a response from the security module is not received after a predetermined amount of time has passed from when the API call and the extracted metadata was communicated to the security module,

9. An apparatus comprising:

   a kernel driver, wherein the kernel driver is configured to:
   
   receive an application program interface (API) call;
   
   extract metadata from the API call;
   
   determine that the API call should be hooked based on the extracted metadata; and
   
   hook the API call.

10. The apparatus of Claim 9, wherein the kernel driver is a binder kernel driver.

11. The apparatus of Claim 10, wherein an interceptor in the kernel driver intercepts the API call.

12. The apparatus of Claim 10, wherein a policy manager inside the kernel driver determines that the API call should be hooked based on the extracted metadata.

13. The apparatus of Claim 10, wherein the extracted metadata is compared to a security policy to determine that the API call should be hooked.

14. The apparatus of Claim 13, wherein the security policy is located in the kernel driver.

15. The apparatus of Claim 10, wherein the kernel driver is further configured to,
communicate the API call and the extracted metadata to a security module, wherein the security module determines if the API call should be allowed or denied.

16. The apparatus of Claim 15, wherein the kernel driver is further configured to; allow the API call if a response from the security module is not received after a predetermined amount of time has passed from when the API call and the extracted metadata was communicated to the security module.

17. A method comprising:
receiving an application program interface (API) call at a kernel driver;
extracting metadata from the API call;
determining that the API call should be hooked based on the extracted metadata; and
hooking the API call,

18. The method of Claim 17, wherein the kernel driver is a binder kernel driver.

19. The method of Claim 18, wherein an interceptor in the kernel driver intercepts the API call.

20. The method of Claim 18, wherein a policy manager inside the kernel driver determines that the API call should be hooked based on the extracted metadata.

21. The method of Claim 18, wherein the extracted metadata is compared to a security policy in the kernel driver to determine that the API call should be hooked.

22. The method of Claim 18, further comprising:
communicating the API call and the extracted metadata to a security module, wherein the security module determines if the API call should be allowed or denied.

23. The method of Claim 22, further comprising:
allowing the API call if a response from the security module is not received after a predetermined amount of time has passed from when the API call and the extracted metadata was communicated to the security module.

24. A system for detecting and mitigating malicious invocation of sensitive code, the system comprising:
   a binder kernel driver, wherein the binder kernel driver is configured to:
   receive an application program interface (API) call;
   extract metadata from the API call;
   determine that the API call should be hooked based on the extracted metadata; and
   hook the API call.

25. The system of Claim 24, wherein a policy manager inside the binder kernel driver compares the extracted metadata to a security policy in the binder kernel driver and determines that the API call should be hooked based on the extracted metadata.
FIG. 1
START

302 AN APPLICATION CALLS AN API

304 THE API CALL IS COMMUNICATED TO A KERNEL DRIVER

306 METADATA RELATED TO THE API CALL IS EXTRACTED

308 BASED ON THE EXTRACTED METADATA, SHOULD THE API CALL BE ALLOWED?

310 NO THE API CALL IS NOT ALLOWED

312 BASED ON THE EXTRACTED METADATA, SHOULD THE API CALL BE HOOKED?

314 YES THE API CALL IS ALLOWED WITHOUT HOOKING

312 NO THE API CALL IS NOT ALLOWED

316 THE API CALL IS HOOKED

END

FIG. 3
400

START

402
AN APPLICATION CALLS AN API

404
THE API CALL IS COMMUNICATED TO A KERNEL DRIVER

406
AN INTERCEPTOR IN THE KERNEL DRIVER INTERCEPTS THE API CALL AND METADATA RELATED TO THE API CALL IS EXTRACTED

408
DOES THE API OR THE METADATA OR BOTH MATCH A SECURITY POLICY?

NO

END

YES

410
THE API CALL IS ALLOWED

THE API CALL IS PROCESSED ACCORDING TO THE SECURITY POLICY

FIG. 4
START

502
AN APPLICATION CALLS OR INVOKES AN API

504
THE API CALL IS COMMUNICATED TO A KERNEL DRIVER

506
AN INTERCEPTOR LOCATED IN THE KERNEL DRIVER INTERCEPTS THE API CALL AND EXTRACTS API AND INVOKING APPLICATION RELATED METADATA

508
THE API AND INVOKING APPLICATION RELATED METADATA ARE COMPARED TO AND MATCHED WITH SECURITY POLICIES

510
WAS A SECURITY POLICY MATCH FOUND?

YES

512
THE APPLICATION CALL IS ALLOWED

NO

500

514
THE SECURITY POLICY THAT MATCHED THE API AND INVOKING APPLICATION RELATED METADATA IS INVOKED

516
SHOULD THE API CALL BE BLOCKED?

YES

A
TO FIG. 5B

NO

524
SHOULD THE API CALL BE ALLOWED?

YES

B
TO FIG. 5B

NO

530
AN ERROR MESSAGE IS GENERATED

532
SHOULD THE API CALL BE HOOKED?

YES

C
TO FIG. 5B

NO

END

FIG. 5A
THE API IS BLOCKED FOR A PRE-DETERMINED AMOUNT OF TIME

AN IN-BAND NOTIFICATION ABOUT THE API CALL AND THE API AND INVOKING APPLICATION RELATED METADATA IS COMMUNICATED TO THE SECURITY SYSTEM ALONG WITH A REQUEST TO ALLOW THE API CALL

HAS THE PRE-DETERMINED AMOUNT OF TIME EXPIRED?

WAS A RESPONSE TO THE REQUEST TO ALLOW THE API CALL RECEIVED?

AN EXPIRED TIME ERROR MESSAGE IS GENERATED

THE API CALL IS ALLOWED

AN ALLOWED NOTIFICATION ALONG WITH THE API AND INVOKING APPLICATION RELATED METADATA IS COMMUNICATED TO THE SECURITY SYSTEM

THE API CALL IS DENIED

AN ACCESS DENIED CODE IS RETURNED TO THE APPLICATION THAT CALLED OR INVOKED THE API

A BLOCKING NOTIFICATION ALONG WITH THE API AND INVOKING APPLICATION RELATED METADATA IS ADDED TO THE SECURITY SYSTEM

FIG. 5B
FIG. 8
INTERNATIONAL SEARCH REPORT

PCT/US2015/012495

A. CLASSIFICATION OF SUBJECT MATTER

G06F 21/56(2013.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 21/56; G06F 15/16; G06F 21/52; H04L 29/06; G06F 12/14; G06F 21/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic database consulted during the international search (name of database and, where practicable, search terms used)

eKOMPASS(KIPO internal) & Keywords: binder kernel driver, hook, API call, security policy, malicious

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>US 2014-0137184 AI (AUCKLAND UNISERVICES LTD.) 15 May 2014 See paragraphs [0043], [0237], [0255H0256], [0393] and figure 4.</td>
<td>1-7, 9-15, 17-22, 24-25, 8, 16, 23</td>
</tr>
<tr>
<td>Y</td>
<td>US 2009-0049550 AI (SERGEI SHEVCHENKO) 19 February 2009 See paragraphs [0076], [0082], [0092], [0095] and figures 4, 6-7.</td>
<td>8, 16, 23</td>
</tr>
<tr>
<td>A</td>
<td>US 2014-0242945 AI (YANG ZENG et al.) 28 August 2014 See paragraphs [0029], [0031] and figure 2.</td>
<td>1-25</td>
</tr>
<tr>
<td>A</td>
<td>WO 2013-103989 AI (OPTIO LABS, LLC) 11 July 2013 See paragraphs [0121], [0122] and figure 4.</td>
<td>1-25</td>
</tr>
<tr>
<td>A</td>
<td>US 2006-0174078 AI (ANDREW ROBISON) 03 August 2006 See paragraphs [0032], [0034] and figure 3.</td>
<td>1-25</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C.

* Special categories of cited documents:
  "A" document defining the general state of the art which is not considered to be of particular relevance
  "E" earlier application or patent but published on or after the international filing date
  "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
  "O" document referring to an oral disclosure, use, exhibition or other means
  "P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

08 July 2015 (08.07.2015)

Date of mailing of the international search report

09 July 2015 (09.07.2015)

Name and mailing address of the ISA/KR

International Application Division
Korean Intellectual Property Office
189 Cheongna-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer

AHN, Jeong Hwan
Telephone No. +82-42-481-8440

Form PCT/ISA/210 (second sheet) (January 2015)
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date</th>
<th>Patent family member(s)</th>
<th>Publication date</th>
</tr>
</thead>
<tbody>
<tr>
<td>US 2014-0137184 Al</td>
<td>15/05/2014</td>
<td>US 2014-0137183 Al</td>
<td>15/05/2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 8893222 B2</td>
<td>18/11/2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2014-077702 Al</td>
<td>22/05/2014</td>
</tr>
<tr>
<td>US 2014-0242945 Al</td>
<td>28/08/2014</td>
<td>CN 103108320 A</td>
<td>15/05/2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2013-071841 Al</td>
<td>23/05/2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP 2015-508540 A</td>
<td>19/03/2015</td>
</tr>
<tr>
<td>US 2006-0174078 Al</td>
<td>03/08/2006</td>
<td>CN 100492300 C</td>
<td>27/05/2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 1818876 A</td>
<td>16/08/2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1693752 Bl</td>
<td>09/04/2008</td>
</tr>
</tbody>
</table>