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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 February 2024 (22.02.2024)

WIPO | PCT

(10) International Publication Number
WO 2024/038417 A1

(51) International Patent Classification:
G06F 21/52 (2013.01) *G06F 21/56* (2013.01)
G06F 21/55 (2013.01)

(21) International Application Number:
PCT/IB2023/058297

(22) International Filing Date:
18 August 2023 (18.08.2023)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
63/399,386 19 August 2022 (19.08.2022) US

(71) Applicant: **TELEFONAKTIEBOLAGET LM ERICSSON (PUBL)** [SE/SE]; SE-164 83 Stockholm (SE).

(72) Inventors: **KERMABON--BOBINNEC, Hugo**; 3400 Place Decelles, Montreal, Québec H3S 1X4 (CA). **POURZANDI, Makan**; 4087 Hampton Street, Montreal, Québec H4A 2L1 (CA). **WANG, Lingyu**; 1515 St. Catherine West, EV009.183, Montreal, Québec H3G 2W1 (CA). **MAJUMDAR, Suryadipta**; 451, 3700 Rue Saint-Antione West, Montreal, Québec H4C 0B1 (CA). **JARRAYA, Yosr**; 6566, boulevard St-Michel, Montreal, Québec H1Y 2G1 (CA).

(74) Agent: **WEISBERG, Alan M.**; Christopher & Weisberg, P.A., 1232 N. University Drive, Plantation, Florida 33322 (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, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, 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, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, 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, ME, 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).

(54) **Title:** DYNAMIC SYSTEM CALLS-LEVEL SECURITY DEFENSIVE SYSTEM FOR CONTAINERIZED APPLICATIONS

(57) **Abstract:** A host node configured to generate a state machine based on a first sequence of system calls identified as corresponding to a security attack and a plurality of parameters and perform, using the state machine, a first inspection of one or more system calls of the first sequence of system calls and one or more parameters corresponding the one or more system calls. The kernel process is updated to monitor a next system call in a second sequence of system calls associated with the software application based at least in part on the performed first inspection. A second inspection of the monitored next system call is performed based at least on the updated kernel process. An execution of the next system call in the second sequence of system calls is blocked or allowed based at least in part on the performed second inspection.

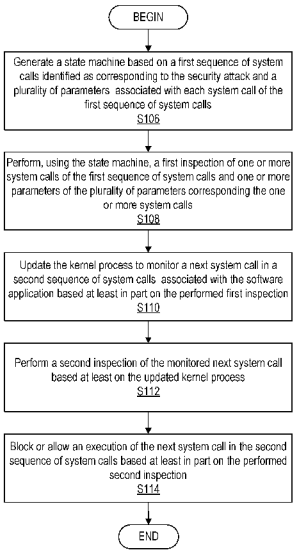

FIG. 4

WO 2024/038417 A1

[Continued on next page]

Published:

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

— *in black and white; the international application as filed contained color or greyscale and is available for download from PATENTSCOPE*

DYNAMIC SYSTEM CALLS-LEVEL SECURITY DEFENSIVE SYSTEM FOR CONTAINERIZED APPLICATIONS

TECHNICAL FIELD

The present disclosure relates to computer system security, and in particular, to a security defensive system for containerized applications.

BACKGROUND

Some computing systems use containerization, which may refer to a type of virtualization in which components of an application are bundled into a single container. Containerization provides strong isolation to a process or a set of processes. Further, the use of containers in computing environments such as cloud environments has increased rapidly as containers offers many advantages over traditionally used virtual machines (VM). Some advantages include faster deployment, increased portability, and less resource overhead. However, containers are quite different from virtual machines and bring new security concerns.

Containers may be run directly on top of an operating system. Further, containers may have access to system calls (also known as "syscalls") at the kernel level to execute one or more functionalities. However, containers can be exposed to attacks and be exploited to do something different from what they were intended to do (e.g., crypto mining). For example, containers may be exploited to compromise the security of containerized applications. An attacking entity or software (e.g., malware) can perform malicious activities by leveraging any system calls available to the exploited container. This can lead to security breaches ranging from data exfiltration to privilege escalation that can be used to gain access to a hosting node (i.e., a node hosting the container and/or containerized application). In addition, the underlying kernel (i.e., the kernel of the operating system where the containers are running), may be directly exposed to attacks from the containers.

In response to these concerns, an attack surface may be minimized, e.g., by reducing interactions between containers and the kernel to a minimum such that only required interactions are allowed. System calls may be employed by the operating system to allow user-spaced programs to use kernel-space features (e.g., open a network connection, load a file in its memory, etc.). Further, in computer system architecture,

protecting rings may be used as a protection mechanism that limits processes operations to their own address space.

One mechanism that may be used to restrict syscalls available to container is Seccomp. Seccomp helps reduce the attack surface of a given process by allowing only the usage of a subset of system calls, i.e., blocking (e.g., filtering out) the rest of possible system calls. Such filtering may be performed by providing Seccomp with a profile composed of Berkeley Packet Filters (BPF) which may be transparent to the filtered process (i.e., the blocked process does not know which "system calls" it can/cannot use until it actually tries them). Further, Seccomp is not limited to only allowing/blocking operations but can also log actions, kill the process, send signals, and/or return a specific error number. Typically, a Seccomp profile is defined (in advance and statically) and used to run the container that is to be protected, e.g., so that access is restricted to only the system calls that the container needs for its execution.

A list of system calls (i.e., whitelist of syscalls) that are needed by containers may be generated based on inspecting the binary code or monitoring the execution of the container. However, the list does not address some requirements such as some syscalls are required by the container to run its normal functionality and thus cannot be blindly blocked. Further, the generated profile is applied based on the regular Seccomp mechanism, which is only used at the starting of the container and cannot be changed at runtime. More specifically, Seccomp profiles may be automatically generated by performing a dynamic analysis of a container and/or automatically generated for containers by tracing all system calls used during a unit and integration testing phase, provided that integration testing exists. Otherwise, the application may be fuzzed, e.g., to attempt to cover all its potential functionalities and the system calls it uses.

Further, Seccomp profiles may be automatically generated by performing static binary analysis of the container application. In particular, compiled application code (in assembly code) may be analyzed, and both system calls directly called by the application or called through Libc (i.e., C programming language library) or other libraries may be identified. Then, a minimal Seccomp profile required by the container may be generated according to the static analysis.

With respect to splitting container runtime into two phases, each split phase may be configured to use a different list of system calls to be whitelisted. In particular, containerized applications use a wide set of system calls during their startup phase, but then only require a reduced set of system calls afterward. The container application

runtime may be split in two phases, and different sets of system calls during each phase may be restricted to reduce the attack surface. The container may be started with its startup Seccomp profile, and the Seccomp profile applied using the second set of syscalls to the container processes may be updated whenever the container has left the startup phase. However, the whitelist-based approach is also used in this case, which is statically generated in advance, and is not able to block syscalls based on a known malicious sequence of syscalls.

In addition, a security observability and runtime enforcement tool based on an extended BFP (eBPF) may be used, e.g., to monitor system calls invocation and follow processes execution in containers. Further, system call filters may be enforced at runtime in containers. However, this tool lacks visualization aspects, makes it difficult for users to write rules to filter sequences of system calls, and relies solely on user knowledge to build security policies.

In sum, existing technologies are limited to reducing attack surface by restricting the container system calls to only what is perceived as necessary. However, an attacker can still use this minimal set of system calls to perform attacks on the container or on the kernel itself. Further, typical Seccomp profiles consist of a static list of system calls that allow (whitelist) some calls, while implicitly blocking the rest. In addition, one goal of existing technologies is to provide a container with the most restrictive Seccomp profile, i.e., the one allowing only the system calls needed for the application to function correctly. However, these approaches may cause several issues:

- For instance, as the Linux kernel (v5.6) contains more than 300 system calls, building the most restrictive profile for a containerized application (manually) is troublesome, as profiles should be generated depending on the libraries used at runtime, the operating system, the build version, etc. Further, profiles should be updated at each update of library, application or operating system. "Default" profiles are offered but are not tailor-made and often end up being inadequate. In addition, although the existing technologies have been proposed to remedy these issues (i.e., claiming to assist users in automatically generating restrictive Seccomp profiles), existing technologies do not address the problems where any system call required by the application is available to use by an attacker.

- Even where split-phase dynamic Seccomp update is used, the Seccomp filter is updated only once (when the container enters the second phase). Afterwards, a regular static Seccomp filter is used until the end of the container lifecycle, thereby bringing back the aforementioned problems.
- Seccomp profiles are generated beforehand and do not address runtime threats that solely rely on system calls required by the container.

SUMMARY

Some embodiments advantageously provide methods, systems, and apparatuses for securing containers and/or containerized applications/systems. In some embodiments, a system (e.g., a defensive system) is described. The system may be configured to provide dynamic security to system calls associated with containerized applications and/or containers.

In some embodiments, a system (also referred to as Phoenix or Phoenix system), e.g., an efficient monitoring and defensive system, enables the protection of containerized applications from attacks originating in the container. The attacks may include zero-day attacks (i.e., attacks with no known patches yet). In some other embodiments, a root cause is learned (i.e., determined) by performing a security analysis (e.g., a security analysis that is received from another host node). In one embodiment, a sequence of system calls (e.g., operating system calls) are learned and/or determined to be related to an execution of a security attack and/or a vulnerability exploit by a set of (at least one) processes of a containerized application. The system (i.e., Phoenix) may be configured to monitor the execution of each system call in the sequence of systems call. The monitoring may be performed (e.g., separately performed) such as by dynamically updating a kernel process (e.g., an in-kernel mechanism such as a seccomp profile) at runtime based on the previously monitored/observed system call in the sequence.

In one embodiment, the system (i.e., Phoenix) enables stopping an attack (i.e., interrupt an execution of the attack by preventing it from succeeding). In another embodiment, the attack is stopped by blocking a subsequent system call in the sequence based on the security criticality of at least one system call in the sequence. For example, some syscalls can be blocked if they are part of a known malicious sequence. If not part of a known malicious sequence, these syscalls may be allowed. This is not supported by existing technology. The sequence input to Phoenix can be learned using an anomaly detection system and/or an analysis of a provenance graph that captures an execution

dependency between system calls of different processes in containerized application, e.g., after incident occurrence.

One or more embodiments block sequences of at least one system call that represents a threat, i.e., that could be employed by an attacker. The blocking is performed without preventing the container from working normally. One or more embodiments provide one or more of the following advantages:

- Reduce the attack surface further than existing solutions while allowing the container to function normally.
- Do not require offline analysis of the container images as security enforcement can be performed during runtime, without interruption.
- If the system calls used in the sequence are derived from provenance a graph, e.g., of the same system, compatibility and versioning issues are avoided.
- Most operations performed by a container involve using system call, and so do most attacks or vulnerability exploits. By enforcing security at the lowest possible level (the system call level), some embodiments can block any attack or exploit, no matter its nature or origin. Thus, it is proposed to restrict the container attack surface even further by blocking specific system calls if they are executed as part of sequences of system calls that are known (e.g., determined by a host node) to be malicious (i.e., employed to perform a malicious operation through the container). In some embodiments, blocking the specific system calls results in the attack being stopped.

According to one aspect, a host node configured to protect a software application from a security attack is described. The host node is configured to execute a kernel process associated with the software application and generate a state machine based on a first sequence of system calls identified as corresponding to the security attack and a plurality of parameters associated with each system call of the first sequence of system calls. The host node is further configured to perform, using the state machine, a first inspection of one or more system calls of the first sequence of system calls and one or more parameters of the plurality of parameters corresponding the one or more system calls. The kernel process is updated to monitor a next system call in a second sequence of system calls associated with the software application based at least in part on the performed first inspection. A second inspection of the monitored next system call is performed based at least on the updated kernel process. An execution of the next system call in the second

sequence of system calls is blocked or allowed based at least in part on the performed second inspection.

In some embodiments, the host node is further configured to one or more of obtain system call information; generate a provenance graph based in part on the system call information; perform forensic analysis using the provenance graph; identify the first sequence of system calls as corresponding to the security attack based one the forensic analysis; and determine the plurality of parameters associated with each system call of the first sequence of system calls.

In some other embodiments, the plurality of parameters include a system call name, arguments, and a process name of a process invoking the corresponding system call.

In some embodiments, the state machine includes a sequence of states and a plurality of transition labels. Each state of the sequence of states indicates one system call name and an action to be performed by the host node if the corresponding state is reached. A transition label of the plurality of transition labels links at least a first state to a second state consecutive to the first state and indicates at least one parameter associated with the second state.

In some other embodiments, the action includes one or more of block a corresponding system call of the second sequence of system calls; warn about the corresponding system call; and step the corresponding system call for release.

In some embodiments, the host node is further configured to, if information associated with the next system call matches the transition label, determine whether to block or allow the execution of the next system call in the second sequence of system calls based on the action indicated by the second state and trigger a transition to a subsequent state of the sequence of states.

In some other embodiments, the host node is further configured to, if information associated with the next system call does not match the transition label, allow the execution of the next system call in the second sequence of system calls.

In some embodiments, the host node is further configured to obtain the information from a notification provided by a kernel program.

In some other embodiments, blocking or allowing the execution of the next system call includes causing the kernel program to block or allow the execution the next system call.

In some embodiments, updating the kernel process to monitor the next system call includes one or both of (A) updating a monitoring profile of the kernel process using a

filter to match the next system call of the second sequence of system calls to one or more system calls of the first sequence of system calls; and (B) trigger the blocking or allowing the execution of the next system call when the next system call of the second sequence of system calls matches the one or more system calls of the first sequence of system calls.

In some other embodiments, performing the second inspection is based on a trigger signal associated with the monitoring profile.

In some embodiments, one or both of blocking or allowing the execution of the next system call is based a system call security criticality and at least the blocking of the execution of the next system call is associated with protecting the software application from the security attack.

In some other embodiments, one or both of the software application is a containerized application comprising a plurality of containers executable by the host node, and the plurality of containers being configured to cause at least the next system call in the second sequence of system calls to be executed.

According to another aspect, a method in a host node configured to protect a software application from a security attack is described. The host node is configured to execute a kernel process associated with the software application. The method includes generating a state machine based on a first sequence of system calls identified as corresponding to the security attack and a plurality of parameters associated with each system call of the first sequence of system calls, performing, using the state machine, a first inspection of one or more system calls of the first sequence of system calls and one or more parameters of the plurality of parameters corresponding the one or more system calls; and updating the kernel process to monitor a next system call in a second sequence of system calls associated with the software application based at least in part on the performed first inspection. The method further includes performing a second inspection of the monitored next system call based at least on the updated kernel process and blocking or allowing an execution of the next system call in the second sequence of system calls based at least in part on the performed second inspection.

In some embodiments, the method further includes one or more of obtaining system call information; generating a provenance graph based in part on the system call information; perform forensic analysis using the provenance graph; identifying the first sequence of system calls as corresponding to the security attack based one the forensic analysis; and determining the plurality of parameters associated with each system call of the first sequence of system calls.

In some other embodiments, the plurality of parameters include a system call name, arguments, and a process name of a process invoking the corresponding system call.

In some embodiments, the state machine includes a sequence of states and a plurality of transition labels. Each state of the sequence of states indicates one system call name and an action to be performed by the host node if the corresponding state is reached. A transition label of the plurality of transition labels links at least a first state to a second state consecutive to the first state and indicates at least one parameter associated with the second state.

In some other embodiments, the action includes one or more of block a corresponding system call of the second sequence of system calls; warn about the corresponding system call; and step the corresponding system call for release.

In some embodiments, the method further includes if information associated with the next system call matches the transition label, determining whether to block or allow the execution of the next system call in the second sequence of system calls based on the action indicated by the second state and triggering a transition to a subsequent state of the sequence of states.

In some other embodiments, the method further includes if information associated with the next system call does not match the transition label, allowing the execution of the next system call in the second sequence of system calls.

In some embodiments, the method further includes obtaining the information from a notification provided by a kernel program.

In some other embodiments, blocking or allowing the execution of the next system call includes causing the kernel program to block or allow the execution the next system call.

In some embodiments, updating the kernel process to monitor the next system call includes one or both of (A) updating a monitoring profile of the kernel process using a filter to match the next system call of the second sequence of system calls to one or more system calls of the first sequence of system calls; and (B) trigger the blocking or allowing the execution of the next system call when the next system call of the second sequence of system calls matches the one or more system calls of the first sequence of system calls.

In some other embodiments, performing the second inspection is based on a trigger signal associated with the monitoring profile.

In some embodiments, one or both of blocking or allowing the execution of the next system call is based a system call security criticality and at least the blocking of the

execution of the next system call is associated with protecting the software application from the security attack.

In some other embodiments, one or both of the software application is a containerized application comprising a plurality of containers executable by the host node, and the plurality of containers being configured to cause at least the next system call in the second sequence of system calls to be executed.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present embodiments, and the attendant advantages and features thereof, will be more readily understood by reference to the following detailed description when considered in conjunction with the accompanying drawings wherein:

FIG. 1 is a schematic diagram of an example network architecture illustrating a communication system connected via an intermediate network to a host computer according to the principles in the present disclosure;

FIG. 2 is a block diagram of a host computer communicating via a network node with a wireless device over an at least partially wireless connection according to some embodiments of the present disclosure;

FIG. 3 is a flowchart of an example process in a host node according to some embodiments of the present disclosure;

FIG. 4 is a flowchart of another example process in a host node according to some embodiments of the present disclosure;

FIG. 5 shows an example system overview according to some embodiments of the present disclosure;

FIG. 6 shows an example overview of an example container runtime protection component according to some embodiments of the present disclosure;

FIG. 7 shows an example state machine (e.g., state transition system) according to some embodiments of the present disclosure;

FIG. 8 shows an example execution flow according to some embodiments of the present disclosure;

FIG. 9 shows another example execution flow according to some embodiments of the present disclosure; and

FIG. 10 shows an example execution flow according to some embodiments of the present disclosure.

DETAILED DESCRIPTION

Before describing in detail example embodiments, it is noted that the embodiments reside primarily in combinations of apparatus components and processing steps related to securing containers and/or containerized applications/systems. Accordingly, components have been represented where appropriate by conventional symbols in the drawings, showing only those specific details that are pertinent to understanding the embodiments so as not to obscure the disclosure with details that will be readily apparent to those of ordinary skill in the art having the benefit of the description herein. Like numbers refer to like elements throughout the description.

As used herein, relational terms, such as "first" and "second," "top" and "bottom," and the like, may be used solely to distinguish one entity or element from another entity or element without necessarily requiring or implying any physical or logical relationship or order between such entities or elements. The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting of the concepts described herein. As used herein, the singular forms "a", "an" and "the" are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms "comprises," "comprising," "includes" and/or "including" when used herein, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components, and/or groups thereof.

In embodiments described herein, the joining term, "in communication with" and the like, may be used to indicate electrical or data communication, which may be accomplished by physical contact, induction, electromagnetic radiation, radio signaling, infrared signaling or optical signaling, for example. One having ordinary skill in the art will appreciate that multiple components may interoperate and modifications and variations are possible of achieving the electrical and data communication.

In some embodiments described herein, the term "coupled," "connected," and the like, may be used herein to indicate a connection, although not necessarily directly, and may include wired and/or wireless connections.

In some embodiments, the term "state machine" is used and may refer to a model such as behavior model. A state machine may include one or more states and transitions. A state may refer to a state of a system. The state machine may include an initial state (i.e.,

where the execution of the state machine begins). In some embodiments, a transition may determine or define for which input a state is changed.

The term "host node" used herein can be any kind of node such as a standalone node and/or a node comprised in a network which may further comprise any of a network node, virtual machine node, node comprising (and/or configurable to run) one or more containers and/or one or more containerized applications, a node comprising one or more operating systems such as Linux.

In some embodiments, a host node may be a network node, a wireless device, or any other device such as a devices configurable to support communication based on standards promulgated by 3GPP (The Third Generation Partnership Project (3GPP)). 3GPP has developed and is developing standards for Fourth Generation (4G) (also referred to as Long Term Evolution (LTE)), Fifth Generation (5G) (also referred to as New Radio (NR)), and Sixth Generation (6G) wireless communication systems. Such systems provide, among other features, broadband communication between network nodes, such as base stations, and mobile wireless devices (WD) such as user equipment (UE), as well as communication between network nodes and between WDs. Some network functions may be deployed as containerized applications which may leverage cloud native technology and containers.

More specifically, the host node may comprise a base station (BS), radio base station, base transceiver station (BTS), base station controller (BSC), radio network controller (RNC), g Node B (gNB), evolved Node B (eNB or eNodeB), Node B, multi-standard radio (MSR) radio node such as MSR BS, multi-cell/multicast coordination entity (MCE), integrated access and backhaul (IAB) node, relay node, donor node controlling relay, radio access point (AP), transmission points, transmission nodes, Remote Radio Unit (RRU) Remote Radio Head (RRH), a core network node (e.g., mobile management entity (MME), self-organizing network (SON) node, a coordinating node, positioning node, MDT node, etc.), an external node (e.g., 3rd party node, a node external to the current network), nodes in distributed antenna system (DAS), a spectrum access system (SAS) node, an element management system (EMS), etc.

The host node may also comprise test equipment and/or may be used to also denote a wireless device (WD) such as a wireless device (WD) or a radio network node. In some embodiments, the non-limiting terms wireless device (WD) or a user equipment (UE) are used interchangeably. The WD herein can be any type of wireless device capable of communicating with a network node or another WD over radio signals, such as wireless

device (WD). The WD may also be a radio communication device, target device, device to device (D2D) WD, machine type WD or WD capable of machine to machine communication (M2M), low-cost and/or low-complexity WD, a sensor equipped with WD, Tablet, mobile terminals, smart phone, laptop embedded equipped (LEE), laptop mounted equipment (LME), USB dongles, Customer Premises Equipment (CPE), an Internet of Things (IoT) device, or a Narrowband IoT (NB-IOT) device, etc.

Also, in some embodiments the generic term "radio network node" is used. It can be any kind of a radio network node which may comprise any of base station, radio base station, base transceiver station, base station controller, network controller, RNC, evolved Node B (eNB), Node B, gNB, Multi-cell/multicast Coordination Entity (MCE), IAB node, relay node, access point, radio access point, Remote Radio Unit (RRU) Remote Radio Head (RRH).

Note that although terminology from one particular wireless system, such as, for example, 3GPP LTE and/or New Radio (NR), may be used in this disclosure, this should not be seen as limiting the scope of the disclosure to only the aforementioned system. Other wireless systems, including without limitation virtual systems, container-based systems, Kubernetes systems, Wide Band Code Division Multiple Access (WCDMA), Worldwide Interoperability for Microwave Access (WiMax), Ultra Mobile Broadband (UMB) and Global System for Mobile Communications (GSM), may also benefit from exploiting the ideas covered within this disclosure.

Note further, that functions described herein as being performed by a host node may be distributed over a plurality of host nodes (e.g., nodes in a network and/or wireless devices and/or network nodes). In other words, it is contemplated that the functions of the host described herein are not limited to performance by a single physical device and, in fact, can be distributed among several physical devices (and/or virtual devices).

Unless otherwise defined, all terms (including technical and scientific terms) used herein have the same meaning as commonly understood by one of ordinary skill in the art to which this disclosure belongs. It will be further understood that terms used herein should be interpreted as having a meaning that is consistent with their meaning in the context of this specification and the relevant art and will not be interpreted in an idealized or overly formal sense unless expressly so defined herein.

Referring now to the drawing figures, in which like elements are referred to by like reference numerals, there is shown in FIG. 1 a schematic diagram of a system 10, according to an embodiment, which comprises one or more networks 12 (e.g., such as

networks 12a, 12b). The network 12 (e.g., network 12a) may comprise a plurality of host nodes 14a, 14b, 14c (referred to collectively as host nodes 14), such as nodes configurable to support one or more containerized applications. Similarly, another network 12 such as network 12b may comprise one or more host nodes 14. Each host node 14a, 14b, 14c is connectable with any other host nodes 14 and/or network 12 (e.g., networks 12a, 12b) over a wired or wireless connection 16 (e.g.,, connections 16a, 16b, 16c) and/or connection 17 (e.g., to/from network 12b). Network 12 may refer to a network associated with a container-based environment and/or an access network and/or a core network and/or a cloud network and/or any other type of network.

A host node 14 is configured to include a security unit 18 (e.g., sequence detection unit) which is configured to perform one or more host node 14 functions described herein such as any step and/or task and/or process and/or method and/or feature described in the present disclosure, e.g., monitor at least one system call of a sequence of system calls; update a kernel process of the host node to monitor a next system call in the sequence of system calls; and perform at least one action based at least on the updated kernel process.

Example implementations, in accordance with an embodiment, of the host node 14 discussed in the preceding paragraphs will now be described with reference to FIG. 2. In a system 10, host node 14 provided in a system 10 and including hardware 20 enabling it to perform one or more host node actions. The hardware 20 may include a communication interface 22 for setting up and maintaining a wired or wireless connection with an interface of a different device of the system 10, such as another host node 14. The communication interface 22 may be formed as or may include, for example, one or more RF transmitters, one or more RF receivers, and/or one or more RF transceivers. The hardware 20 may also include a radio interface 24 for setting up and maintaining a wireless connection with a wireless interface of a different device of the system 10, such as wireless device. The radio interface 24 may be formed as or may include, for example, one or more RF transmitters, one or more RF receivers, and/or one or more RF transceivers.

In the embodiment shown, the hardware 20 of the host node 14 further includes processing circuitry 26. The processing circuitry 26 may include a processor 28 and a memory 30. In particular, in addition to or instead of a processor, such as a central processing unit, and memory, the processing circuitry 26 may comprise integrated circuitry for processing and/or control, e.g., one or more processors and/or processor cores and/or FPGAs (Field Programmable Gate Array) and/or ASICs (Application Specific

Integrated Circuitry) adapted to execute instructions. The processor 28 may be configured to access (e.g., write to and/or read from) the memory 30, which may comprise any kind of volatile and/or nonvolatile memory, e.g., cache and/or buffer memory and/or RAM (Random Access Memory) and/or ROM (Read-Only Memory) and/or optical memory and/or EPROM (Erasable Programmable Read-Only Memory).

Thus, the host node 14 further has software 32 stored internally in, for example, memory 30, or stored in external memory (e.g., database, storage array, network storage device, etc.) accessible by the host node 14 via an external connection. Software 32 may include at least an operating system 34 and/or software application 36 and/or containers 38 (such as one or more containers associated with a containerized application). In some embodiments, containers 38 may be comprised by (and/or be) software application 36, e.g., a containerized application. The operating system may include a kernel 40 (i.e., an operating system kernel, kernel program, kernel process, etc.). In a nonlimiting example, the operating system is a Linux operating system, and kernel 40 is a Linux kernel. The software 32 (and/or any of its components) may be executable by the processing circuitry 26. The processing circuitry 26 may be configured to control any of the methods and/or processes described herein and/or to cause such methods, and/or processes to be performed, e.g., by host node 14. Processor 28 corresponds to one or more processors 28 for performing host node 14 functions described herein. The memory 30 is configured to store software 32 (and/or any of its components) data, programmatic software code and/or other information described herein. In some embodiments, the software 32 may include instructions that, when executed by the processor 28 and/or processing circuitry 26, causes the processor 28 and/or processing circuitry 26 to perform the processes described herein with respect to host node 14. For example, processing circuitry 26 of the host node 14 may include security unit 18 configured to perform one or more host node 14 functions described herein such as any step and/or task and/or process and/or method and/or feature described in the present disclosure, e.g., monitor at least one system call of a sequence of system calls; update a kernel process of the host node to monitor a next system call in the sequence of system calls; and perform at least one action based at least on the updated kernel process.

FIG. 3 is a flowchart of an example process (i.e., method) in a host node 14 according to some embodiments of the present disclosure. One or more blocks described herein may be performed by one or more elements of host node 14 such as by one or more of processing circuitry 26 (including the security unit 18), processor 28, radio interface 24

and/or communication interface 22. Host node 14 is configured to monitor (Block S100) at least one system call of a sequence of system calls, the sequence of system calls being associated with a containerized application and having been identified as a system attack, as described herein. The host node 14 is configured to update (Block S102) a kernel process of the host node 14 to monitor a next system call in the sequence of system calls based at least in part on the monitored at least one system call, as described herein. The host node 14 is configured to perform (Block S104) at least one action based at least on the updated kernel process, where the at least one action is performed for protecting at least the containerized application from the security attack, as described herein.

In some embodiments, the updating of the kernel process includes updating a monitor profile to include at least one filter to match a predetermined system call to be monitored and trigger the performing of the at least one action when the predetermined system call is matched.

In some other embodiment, the performing of the at least one action includes one of blocking and allowing an execution of the next system call of the sequence of system calls based at least in part on a system call security criticality.

In an embodiment, the blocking of the execution of the next system call is associated with blocking the security attack.

FIG. 4 is a flowchart of an example process (i.e., method) in a host node 14 according to some embodiments of the present disclosure. One or more blocks described herein may be performed by one or more elements of host node 14 such as by one or more of processing circuitry 26 (including the security unit 18), processor 28, radio interface 24 and/or communication interface 22. Host node 14 is configured to monitor (Block S100) at least one system call of a sequence of system calls, the sequence of system calls being associated with a containerized application and having been identified as a system attack, as described herein. The host node 14 is configured to protect a software application 36 from a security attack and execute a kernel process associated with the software application 36. The host node 14 is further configured to generate (Block S106) a state machine 50 based on a first sequence of system calls identified as corresponding to the security attack and a plurality of parameters associated with each system call of the first sequence of system calls, perform (Block S108), using the state machine 50, a first inspection of one or more system calls of the first sequence of system calls and one or more parameters of the plurality of parameters corresponding the one or more system calls, and update (Block S110) the kernel process to monitor a next system call in a second

sequence of system calls associated with the software application based at least in part on the performed first inspection. The host node 14 is also configured to perform (Block S112) a second inspection of the monitored next system call based at least on the updated kernel process and block or allow (Block S114) an execution of the next system call in the second sequence of system calls based at least in part on the performed second inspection.

In some embodiments, the method further includes one or more of obtaining system call information; generating a provenance graph based in part on the system call information; perform forensic analysis using the provenance graph; identifying the first sequence of system calls as corresponding to the security attack based one the forensic analysis; and determining the plurality of parameters associated with each system call of the first sequence of system calls.

In some other embodiments, the plurality of parameters include a system call name, arguments, and a process name of a process invoking the corresponding system call.

In some embodiments, the state machine 50 includes a sequence of states 52 and a plurality of transition labels 56. Each state of the sequence of states 52 indicates one system call name and an action to be performed by the host node 14 if the corresponding state is reached. A transition label 56 of the plurality of transition labels 56 links at least a first state 52a to a second state 52b consecutive to the first state 52a and indicates at least one parameter associated with the second state 52b.

In some other embodiments, the action includes one or more of block a corresponding system call of the second sequence of system calls, warn about the corresponding system call, and step the corresponding system call for release.

In some embodiments, the method further includes if information associated with the next system call matches the transition label, determining whether to block or allow the execution of the next system call in the second sequence of system calls based on the action indicated by the second state 52b and triggering a transition to a subsequent state 52 of the sequence of states 52.

In some other embodiments, the method further includes if information associated with the next system call does not match the transition label, allowing the execution of the next system call in the second sequence of system calls.

In some embodiments, the method further includes obtaining the information from a notification provided by a kernel program.

In some other embodiments, blocking or allowing the execution of the next system call includes causing the kernel program to block or allow the execution the next system call.

In some embodiments, updating the kernel process to monitor the next system call includes one or both of (A) updating a monitoring profile of the kernel process using a filter to match the next system call of the second sequence of system calls to one or more system calls of the first sequence of system calls; and (B) trigger the blocking or allowing the execution of the next system call when the next system call of the second sequence of system calls matches the one or more system calls of the first sequence of system calls.

In some other embodiments, performing the second inspection is based on a trigger signal associated with the monitoring profile.

In some embodiments, one or both of blocking or allowing the execution of the next system call is based a system call security criticality and at least the blocking of the execution of the next system call is associated with protecting the software application from the security attack.

In some other embodiments, one or both of the software application is a containerized application comprising a plurality of containers executable by the host node, and the plurality of containers being configured to cause at least the next system call in the second sequence of system calls to be executed.

Having described the general process flow of arrangements of the disclosure and having provided examples of hardware and software arrangements for implementing the processes and functions of the disclosure, the sections below provide details and examples of arrangements for securing containers and/or containerized applications/systems such as by, for example, at least detecting one or more system calls in a sequence of system calls that had been previously identified as being associated with a malicious or security attack.

One or more host node 14 functions described below may be performed by one or more of processing circuitry 26, processor 28, security unit 18, hardware 20, software 32, etc. In some embodiments, the term a state transition system is used and may refer to a state machine 50.

In some embodiments provide a system (e.g., runtime defending system) for protecting software applications 36 (e.g., containerized applications) from security attacks. Some security attacks may include zero-day attacks. One or more embodiments enable efficient runtime monitoring and blocking of an ordered execution of one or more malicious sequences of system calls. The system calls may be performed by the set of

processes inside (and/or associated with) one or more containers. A sequence of systems calls may include one or more systems calls, where the one or more system calls are associated with a containerized application. In some embodiments, the sequence of system calls may define an attack (e.g., a security attack may be determined based on the sequence of system calls and/or information associated with the sequence). In some other embodiments, a kernel process (e.g., the in-kernel mechanism such as a seccomp profile) is dynamically updated, e.g.,. so that each specific system call in a given sequence is monitored at a time based on the observation of the precedent system call in the sequence and/or by selectively blocking a given critical system call in the sequence (e.g., to stop (or abort) the security attack). In one or more embodiments, the kernel process is updated with a next system call in the sequence if the preceding system call was seen (i.e., determined, detected, determined to be associated with an attack, etc.). In an embodiment, the updating of the kernel process (e.g., kernel) is used to enable monitoring of the execution of each system call in the sequence.

In some other embodiments, an ongoing security attack and/or the progress of the ongoing security attack at runtime is stopped when there is a match with a subset of the sequence of system calls.

FIG. 5 shows an example system overview according to some embodiments of the present disclosure. System 10 comprises one or more components such as software components and/or hardware components. In some embodiments, system 10 includes a network 12a (e.g., a containerized-based environment), a kernel 40 (e.g., Linux kernel) configured to receive one or more system calls such as from one or more containers and/or processes associated with the containers. System 10 may include one or more other components such as comprised (and/or performed, executed) by security unit 18. In some other embodiments, system 10 may further include network 12b (e.g., another container-based environment) which is configured to communicate with network 12a and may include one or more containers and/or containerized applications, etc. In an embodiment, system 10 includes a combination of network 12a and network 12b.

Further, system 10 (e.g., Phoenix) may be a runtime defending system for protecting host node 14 and/or software applications 36 (e.g., containerized applications) and/or containers 38 from zero-day security attacks originating in containers 38. System 10 (e.g., Phoenix) may receive a set of malicious sequences of enriched system calls. Enriched system calls may include system calls and a set of parameters related to each system call in a sequence of system calls (e.g., a plurality of system calls which may be

ordered in a predetermined manner such as in a sequence). The sequence(s) may be learned from a provenance-based forensics analysis performed on a provenance graph, which may capture the dependency between system calls used by several processes within the software application 36 (e.g., containerized application that system 10 aims to protect at runtime). Further, system 10 (e.g., Phoenix) may comprise of two main components: a Container Restarting component and a Container Runtime Protection component. The Container Restarting component may identify at least one compromised container to be protected and/or restart the container and/or prepare the container to be hardened. The Container Runtime Protection component may apply protective countermeasures (e.g., any task performed by any of the components of system 10) to a container 38 (e.g., compromised container) so that the container 38 is protected such as during its execution (e.g., at runtime).

In some embodiments, enriched system call sequence may refer to a sequence of system calls where each system call may be enriched with a name of the system call, argument(s) of the system call, and/or a process name of the process invoking the system call. In one non limiting example, one or more enriched system calls (e.g., enriched system call sequence) may include one or more parameters (e.g., may be annotated) as shown in Table 1 below.

Order of the syscall in the sequence	1	2	3	4	5
System call name	open	connect	execve	connect	open
Calling process	java	java	java	curl	curl
Arguments	'log4j.jar'	'172.16.240.11','1389'	'curl'	'172.16.240.11', '80'	'/tmp/
Action	STEP	WARN	WARN	WARN	BLOCK

Table 1. - Example of tuples (each column is a tuple) of enriched system calls sequence including <order in sequence, syscall name, calling process, arguments> annotated with an action for each system call.

In an embodiment, each system call in the enriched system call sequence may be annotated with an action to determine what to do with the corresponding system call (i.e., block or release) and/or what the next steps to be performed by system 10 (e.g., Phoenix) and/or host node 14. Although three possible actions such as STEP, WARN, and BLOCK are described, any other actions may be specified (e.g., triggers a given service to consume this system call). The annotation of actions on each system call (e.g., syscalls) may be performed by user (e.g., a security expert) such as after analyzing system logs. Further, the annotation of actions may be handled by automated tools external to system 10 (e.g., Phoenix) such as a component configured to perform Provenance Graph analysis.

FIG. 6 shows an example overview of an example Container Runtime Protection component according to some embodiments of the present disclosure.

More specifically, the Container Runtime Protection (CRP) component of system 10 may be configured to interact with the network 12 (e.g., container-based environment), particularly, with the Linux user-space and kernel-space where the containers are deployed. The CRP component may be used to monitor and protect the containers. The interactions may include updates to the Seccomp monitor profile to dynamically monitor the execution of a specific system call and interactions with other components of system 10 (e.g., the "ptrace: program) to monitor the enriched system call execution, to notify any component of system 10, and to trigger the actions requested by system 10 (e.g., by host node 14) on the currently observed system calls.

In some embodiments, Seccomp refers to a Linux Kernel feature that may be applied to processes and/or restrict and/or modify their usage of system calls, e.g., by implementing filters. In some other embodiments, "ptrace" refers to a kernel program (e.g., combination of software and hardware) that may attach to another process. Further, a Seccomp filter can specify (e.g., request) to send a signal to ptrace upon interception of a specified system call. Ptrace may then inspect the process memory (and thus its syscall, arguments, etc.). Ptrace may also enforce actions on a process by writing into its memory (and thus block its syscall, modify its arguments etc.)

The CRP component may include a Sequence preprocessing and annotation (SPA) module, Dynamic Seccomp Monitor (DSM) module, Sequence State Monitoring and Matching (SSMM) module, i.e., modules/components (e.g., a combination of software and/or hardware) such as performed by (and/or comprised by) security unit 18 of host node 14.

The SPA module may be based on a received sequence of enriched system calls as sequence of tuples. Each tuple may include one or more of the following elements: order of the system call in the sequence, its name, a calling process identifier, and an argument of the system call. SPA may first preprocess the sequence to identify each syscall and its related parameters, then add annotation to each tuple denoting the action to be performed by system 10 (e.g., Phoenix), e.g., by host node 14, when each of these system calls (e.g., step, warn, block) are detected/observed/determined. An action added may be either automatically generated by system 10 (e.g., Phoenix) and/or any of its components such that syscalls, before the last one, are labeled "Step" and the last one in the sequence is labeled "Block" or based on some pre-defined association between system calls and actions (e.g., based on the criticality of the syscall from security point of view). For example, executing a shell may be more critical than opening a file.

Further, SPA may generate a state transition system (i.e., state machine 50) from the tuples where a state 52 denotes the currently observed system call name and the action to be performed (by system 10 (e.g., Phoenix) and/or any of its components) if the state 52 is reached. Transitions 54 may be used to link two (or more) consecutive states 52. Labels (i.e., transition labels 56) may also be used and may include an expected next system call with the corresponding calling process and arguments. If the label of the transition 54 is matched, the next state 52 (destination of the transition) can be reached.

FIG. 7 shows an example state machine 50 (e.g., state transition system) which may include a states 52a, 52b, 52c, 52d, 52e, 52f (collectively referred to as states 52) which are linked by transitions 54. States 52 may include information about the state (e.g., initial state) and/or system call name (e.g., OPEN, CONNECT, EXECVE, etc.) and/or actions (e.g., step, warn, block, etc.). A transition may occur based on information included in transition labels 56 which may include parameters associated with system calls. In some embodiments, state machine 50 may be referred to as linked list of states. A method may be associated with state machine 50 corresponding to a sequence of system calls (e.g., as shown in Table 1) generated by the SPA module. At step S200, a transition is determined/performed if syscalls equals OPEN, process equals java, and Arguments equals 'log4j.jar.' At step S202, a transition is determined/performed if syscalls equals CONNECT, process equals java, and Arguments equals '172.16.240.11', '1389'. At step S204, a transition is determined/performed if syscalls equals EXECVE, process equals java, and Arguments equals 'curl'. At step S206, a transition is determined/performed if syscalls equals CONNECT, process equals curl, and Arguments equals '172.16.240.11', '80'. At step S208, a transition is determined/performed if syscalls equals OPEN, process equals curl, and Arguments equals '/tmp/'.

FIG. 8 shows an example execution flow (e.g., S210-S216 performed by SPA, host node 14, security unit 18, etc.). At step 210, a malicious sequence of system calls is preprocessed. At step S212, an action for each system call is associated (e.g., using an annotation). At step S214, a state transition system is generated (e.g., one or more transitions are generated). At step S216, the state transition system is sent to SSMM.

The DSM module may be configured to interact with Seccomp and dynamically update the Seccomp monitor profile using a specific filter to match a specific system call to be monitored at runtime by system 10 (e.g., Phoenix) and/or any of its components such as security unit 18 of host node 14. DSM receives the name of system call(s) to be monitored from the sequence state monitoring and matching module (SSMM).

In some embodiments, a filter may be defined. The following is an example of Seccomp filter:

```
{
  "defaultAction":"SCMP_ACT_ALLOW",
  "syscalls":[
    {
      "name":"connect",
```

```
            "action":"SCMP_ACT_KILL",
            "args":[

            ]
          },
          {
            "name":"open",
            "action":"SCMP_ACT_TRACE",
            "args":[

            ]
          }
        ]
      }
```


FIG. 9 shows a flow of execution performed by DSM (e.g., performed by security unit 18). A step S218, a Seccomp profile is created with a new rule to monitor the system call, and at step S220, the Seccomp profile of the container is replaced with a new one. After the filter is updated, when the system call occurs, Seccomp may send a signal to ptrace so ptrace later inspects it in more details (argument and calling process).

The monitored system call may be held till a decision on the action to be performed on this system call is made (e.g., by system 10, host node 14, security unit 18), which may be handled/processed by the SSMM module as follows.

The SSMM module receives a state transition system (such as shown in FIG. 6) from the SPA module. The state transition system corresponds to a specific malicious sequence of enriched system calls (e.g., that may follow the ordered execution of the malicious sequence by the container that is to be protected).

SSMM uses the state transition system to perform one or more of the following: (1) save internally the current state of execution, i.e., the last seen system call from the sequence, and take the action corresponding to when it occurred; (2) determine the next system call to be monitored in the sequence and send it to DSM; and (3) identify the arguments and calling process of the expected next system call in the sequence, which may be used to be matched with those of the system call actually being executed by the container to be protected (i.e., matching the currently executing system call and parameters with the enriched system call captured in the malicious sequence). Arguments and calling process may be determined by ptrace, which may notify any component of system 10 (e.g., Phoenix) about the occurrence of the system call monitored by Seccomp. Once the matching is determined to happen, SSM may send a signal to ptrace with the action to be performed on the currently observed system call.

FIG. 10 shows an execution flow performed by the SSMM module and interactions with ptrace and DSM module. At step S222, a state transition system is received. At step S224, an instance of the state transition system is initialized. At step, a next system call is determined from the next transition. At step S228, the next system call is sent to DSM. At step S230, a notification from ptrace is received and/or processed. At step S232, the current system call execution information is matched with the label of the next transition in the state transition system. At step S234, whether there is a match is determined. If there is a match, at step S236, the transition to the next state 52 in the state transition system is triggered to update the last seen system call. If there is no match, at step S238, a decision is sent to ptrace, e.g., to release the system call. At step S240, SSMM determines whether there is an action to be performed. If there is an action, such as STEP, at step S242, a decision is sent to ptrace, e.g., to release the system call. If there is an action such as BLOCK, at step S244, a decision is sent to ptrace, e.g., to block the system call.

In some embodiments, one or more of the following may be performed by host node 14:

1. Build a state machine 50 (e.g., state transition system) corresponding to an identified system attack. Depending on current and previous states 52 of the process, the state machine allows or blocks system calls. This offers the possibility of having a system call X being allowed or blocked for the same process based on the state machine 50 at different moments based the life cycle of the process. In addition, the state machine 50 can be modified.

2. Monitor at least one system call of a sequence of system calls, the sequence of system calls being associated with a containerized application (e.g., software application 36 comprising containers 38) and having been identified as a system attack.
3. Update a kernel process (associated with kernel 40) of the host node 14 to monitor a next system call in the sequence of system calls based at least in part on the monitored at least one system call.
    - Updating the kernel process may include updating a monitor profile to include at least one filter to match a predetermined system call to be monitored and trigger the performing of at least one action when the predetermined system call is matched.
4. Perform a further inspection of the monitored system call based on the trigger signal of the monitoring profile.
    - Performing the further inspection may include inspecting one or more parameter of the containerized application at the time of interception and matching it against a parameter in the sequence of system calls identified as a system attack.
5. Perform at least one action based at least on the updated kernel process and the further inspection, where the at least one action is performed for protecting at least the containerized application from the security attack.
    - Performing at least one action may include blocking or allowing an execution of the next system call in the sequence of system calls based at least in part on a system call security criticality.

In some embodiments, making the "further inspection" operation may be less "resource costly" as a result of the "updating a kernel process" (bullet #3) above.

The following is a nonlimiting list of example embodiments.

Embodiment A1. A host node being configured to, and/or comprising processing circuitry configured to:

monitor at least one system call of a sequence of system calls, the sequence of system calls being associated with a containerized application and having been identified as a system attack;

update a kernel process of the host node to monitor a next system call in the sequence of system calls based at least in part on the monitored at least one system call; and

perform at least one action based at least on the updated kernel process, the at least one action being performed for protecting at least the containerized application from the security attack.

Embodiment A2. The host node of Embodiment A1, wherein the updating of the kernel process includes:

updating a monitor profile to include at least one filter to match a predetermined system call to be monitored and trigger the performing of the at least one action when the predetermined system call is matched.

Embodiment A3. The host node of any one of Embodiments A1 and A2, wherein the performing of the at least one action includes:

one of blocking and allowing an execution of the next system call in the sequence of system calls based at least in part on a system call security criticality.

Embodiment A4. The host node of Embodiment A3, wherein the blocking of the execution of the next system call is associated with blocking the security attack.

Embodiment B1. A method in a host node, the method comprising:

monitoring at least one system call of a sequence of system calls, the sequence of system calls being associated with a containerized application and having been identified as a system attack;

updating a kernel process of the host node to monitor a next system call in the sequence of system calls based at least in part on the monitored at least one system call; and

performing at least one action based at least on the updated kernel process, the at least one action being performed for protecting at least the containerized application from the security attack.

Embodiment B2. The method of any one of Embodiment B1, wherein the updating of the kernel process includes:

updating a monitor profile to include at least one filter to match a predetermined system call to be monitored and trigger the performing of the at least one action when the predetermined system call is matched.

Embodiment B3. The method of any one of Embodiments B1 and B2, wherein the performing of the at least one action includes:

one of blocking and allowing an execution of the next system call of the sequence of system calls based at least in part on a system call security criticality.

Embodiment B4. The method of Embodiment B3, wherein the blocking of the execution of the next system call is associated with blocking the security attack.

As will be appreciated by one of skill in the art, the concepts described herein may be embodied as a method, data processing system, computer program product and/or computer storage media storing an executable computer program. Accordingly, the concepts described herein may take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects all generally referred to herein as a "circuit" or "module." Any process, step, action and/or functionality described herein may be performed by, and/or associated to, a corresponding module, which may be implemented in software and/or firmware and/or hardware. Furthermore, the disclosure may take the form of a computer program product on a tangible computer usable storage medium having computer program code embodied in the medium that can be executed by a computer. Any suitable tangible computer readable medium may be utilized including hard disks, CD-ROMs, electronic storage devices, optical storage devices, or magnetic storage devices.

Some embodiments are described herein with reference to flowchart illustrations and/or block diagrams of methods, systems and computer program products. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer (to thereby create a special purpose computer), special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable memory or storage medium that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer readable memory produce an article of manufacture including instruction means which implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be

performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide steps for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

It is to be understood that the functions/acts noted in the blocks may occur out of the order noted in the operational illustrations. For example, two blocks shown in succession may in fact be executed substantially concurrently or the blocks may sometimes be executed in the reverse order, depending upon the functionality/acts involved. Although some of the diagrams include arrows on communication paths to show a primary direction of communication, it is to be understood that communication may occur in the opposite direction to the depicted arrows.

Computer program code for carrying out operations of the concepts described herein may be written in an object oriented programming language such as Python, Java® or C++. However, the computer program code for carrying out operations of the disclosure may also be written in conventional procedural programming languages, such as the "C" programming language. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer. In the latter scenario, the remote computer may be connected to the user's computer through a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

Many different embodiments have been disclosed herein, in connection with the above description and the drawings. It will be understood that it would be unduly repetitious and obfuscating to literally describe and illustrate every combination and subcombination of these embodiments. Accordingly, all embodiments can be combined in any way and/or combination, and the present specification, including the drawings, shall be construed to constitute a complete written description of all combinations and subcombinations of the embodiments described herein, and of the manner and process of making and using them, and shall support claims to any such combination or subcombination.

It will be appreciated by persons skilled in the art that the embodiments described herein are not limited to what has been particularly shown and described herein above. In addition, unless mention was made above to the contrary, it should be noted that all of the

accompanying drawings are not to scale. A variety of modifications and variations are possible in light of the above teachings without departing from the scope of the following claims.

What is claimed is:

1. A host node (14) configured to protect a software application (36) from a security attack, the host node (14) being configured to execute a kernel process associated with the software application (36), the host node (14) being configured to:

generate a state machine (50) based on a first sequence of system calls identified as corresponding to the security attack and a plurality of parameters associated with each system call of the first sequence of system calls;

perform, using the state machine (50), a first inspection of one or more system calls of the first sequence of system calls and one or more parameters of the plurality of parameters corresponding the one or more system calls;

update the kernel process to monitor a next system call in a second sequence of system calls associated with the software application (36) based at least in part on the performed first inspection;

perform a second inspection of the monitored next system call based at least on the updated kernel process; and

block or allow an execution of the next system call in the second sequence of system calls based at least in part on the performed second inspection.

2. The host node (14) of Claim 1, wherein the host node (14) is further configured to one or more of:

obtain system call information;

generate a provenance graph based in part on the system call information;

perform forensic analysis using the provenance graph;

identify the first sequence of system calls as corresponding to the security attack based one the forensic analysis; and

determine the plurality of parameters associated with each system call of the first sequence of system calls.

3. The host node (14) of any one of Claims 1 and 2, wherein the plurality of parameters include:

a system call name;

arguments; and

a process name of a process invoking the corresponding system call.

4. The host node (14) of any one of Claims 1-3, wherein the state machine (50) includes a sequence of states (52) and a plurality of transition labels (56), each state (52) of the sequence of states (52) indicating one system call name and an action to be performed by the host node (14) if the corresponding state (52) is reached, a transition label (56) of the plurality of transition labels (56) linking at least a first state (52) to a second state (52) consecutive to the first state (52) and indicating at least one parameter associated with the second state (52).

5. The host node (14) of Claims 4, wherein the action includes one or more of:

block a corresponding system call of the second sequence of system calls;

warn about the corresponding system call; and

step the corresponding system call for release.

6. The host node (14) of any one of Claims 4 and 5, wherein the host node (14) is further configured to:

if information associated with the next system call matches the transition label (56):

determine whether to block or allow the execution of the next system call in the second sequence of system calls based on the action indicated by the second state (52); and

trigger a transition to a subsequent state (52) of the sequence of states (52).

7. The host node (14) of any one of Claims 4-6, wherein the host node (14) is further configured to:

if information associated with the next system call does not match the transition label (56):

allow the execution of the next system call in the second sequence of system calls.

8. The host node (14) of any one of Claims 6 and 7, wherein the host node (14) is further configured to:

obtain the information from a notification provided by a kernel program.

9. The host node (14) of Claim 8, wherein blocking or allowing the execution of the next system call includes:

causing the kernel program to block or allow the execution of the next system call.

10. The host node (14) of any one of Claims 1-9, wherein updating the kernel process to monitor the next system call includes one or both of:

updating a monitoring profile of the kernel process using a filter to match the next system call of the second sequence of system calls to one or more system calls of the first sequence of system calls; and

trigger the blocking or allowing the execution of the next system call when the next system call of the second sequence of system calls matches the one or more system calls of the first sequence of system calls.

11. The host node (14) of Claim 10, wherein performing the second inspection is based on a trigger signal associated with the monitoring profile.

12. The host node (14) of any one of Claims 1-11, wherein one or both of:

blocking or allowing the execution of the next system call is based a system call security criticality; and

at least the blocking of the execution of the next system call is associated with protecting the software application (36) from the security attack.

13. The host node (14) of any one of Claims 1-12, wherein one or both of:

the software application (36) is a containerized application comprising a plurality of containers (38) executable by the host node (14); and

the plurality of containers (38) being configured to cause at least the next system call in the second sequence of system calls to be executed.

14. A method in a host node (14) configured to protect a software application (36) from a security attack, the host node (14) being configured to execute a kernel process associated with the software application (36), the method comprising:

generating (S106) a state machine (50) based on a first sequence of system calls identified as corresponding to the security attack and a plurality of parameters associated with each system call of the first sequence of system calls;

performing (S108), using the state machine (50), a first inspection of one or more system calls of the first sequence of system calls and one or more parameters of the plurality of parameters corresponding the one or more system calls;

updating (S110) the kernel process to monitor a next system call in a second sequence of system calls associated with the software application (36) based at least in part on the performed first inspection;

performing (S112) a second inspection of the monitored next system call based at least on the updated kernel process; and

blocking or allowing (S114) an execution of the next system call in the second sequence of system calls based at least in part on the performed second inspection.

15. The method of Claim 14, wherein the method further comprises one or more of:

obtaining system call information;

generating a provenance graph based in part on the system call information;

performing forensic analysis using the provenance graph;

identifying the first sequence of system calls as corresponding to the security attack based one the forensic analysis; and

determining the plurality of parameters associated with each system call of the first sequence of system calls.

16. The method of any one of Claims 14 and 15, wherein the plurality of parameters include:

a system call name;

arguments; and

a process name of a process invoking the corresponding system call.

17. The method of any one of Claims 14-16, wherein the state machine (50) includes a sequence of states (52) and a plurality of transition labels (56), each state (52) of the sequence of states (52) indicating one system call name and an action to be performed by the host node (14) if the corresponding state (52) is reached, a transition label (56) of the plurality of transition labels (56) linking at least a first state (52) to a second state (52) consecutive to the first state (52) and indicating at least one parameter associated with the second state (52).

18. The method of Claims 17, wherein the action includes one or more of:

block a corresponding system call of the second sequence of system calls;

warn about the corresponding system call; and

step the corresponding system call for release.

19. The method of any one of Claims 17 and 18, wherein the method further comprises:

if information associated with the next system call matches the transition label (56):

determining whether to block or allow the execution of the next system call in the second sequence of system calls based on the action indicated by the second state (52); and

triggering a transition to a subsequent state (52) of the sequence of states (52).

20. The method of any one of Claims 17-19, wherein the method further comprises:

if information associated with the next system call does not match the transition label (56):

allowing the execution of the next system call in the second sequence of system calls.

21. The method of any one of Claims 19 and 20, wherein the method further comprises:

obtaining the information from a notification provided by a kernel program.

22. The method of Claim 21, wherein blocking or allowing the execution of the next system call includes:

causing the kernel program to block or allow the execution of the next system call.

23. The method of any one of Claims 14-22, wherein updating the kernel process to monitor the next system call includes one or both of:

updating a monitoring profile of the kernel process using a filter to match the next system call of the second sequence of system calls to one or more system calls of the first sequence of system calls; and

trigger the blocking or allowing the execution of the next system call when the next system call of the second sequence of system calls matches the one or more system calls of the first sequence of system calls.

24. The method of Claim 23, wherein performing the second inspection is based on a trigger signal associated with the monitoring profile.

25. The method of any one of Claims 14-24, wherein one or both of:

blocking or allowing the execution of the next system call is based a system call security criticality; and

at least the blocking of the execution of the next system call is associated with protecting the software application (36) from the security attack.

26. The method of any one of Claims 14-25, wherein one or both of:

the software application (36) is a containerized application comprising a plurality of containers (38) executable by the host node (14); and

the plurality of containers (38) being configured to cause at least the next system call in the second sequence of system calls to be executed.

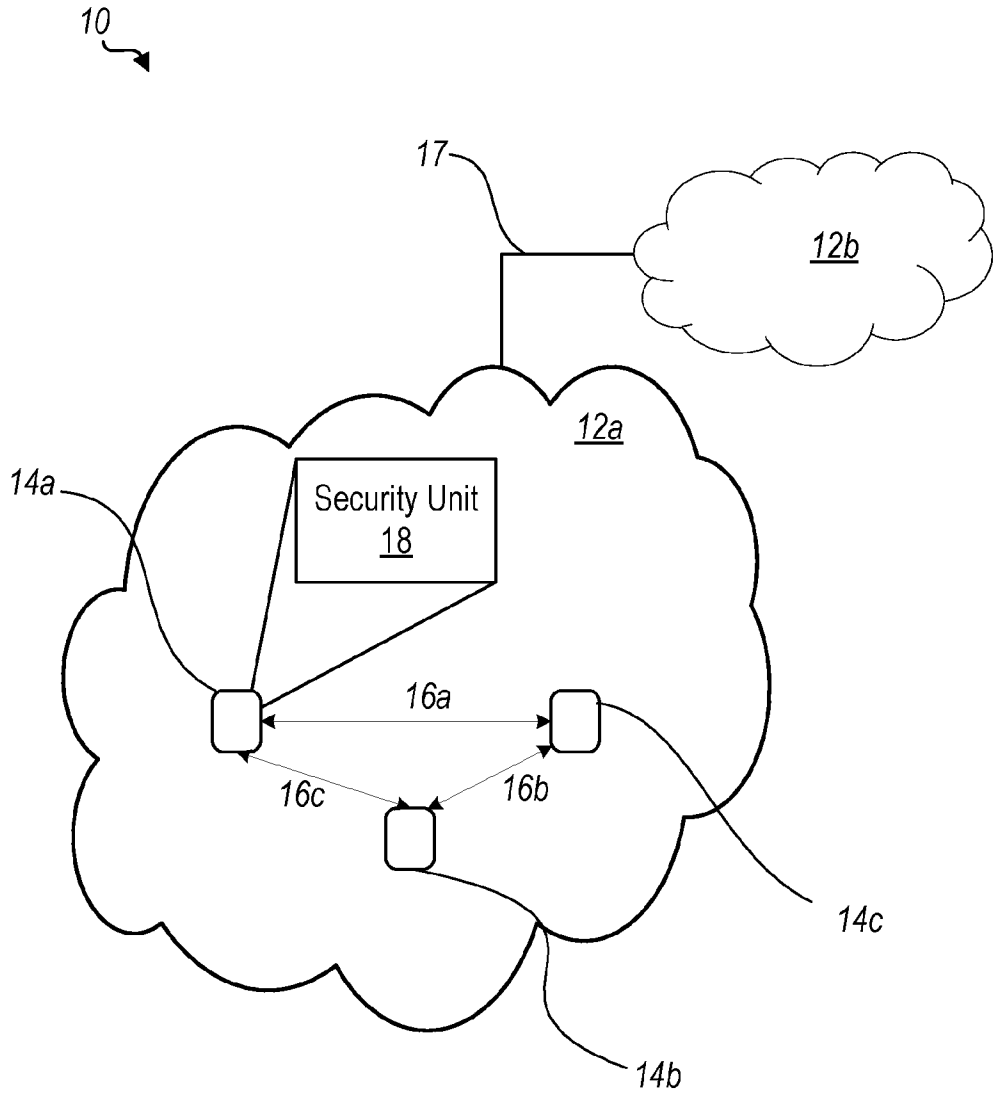

FIG. 1

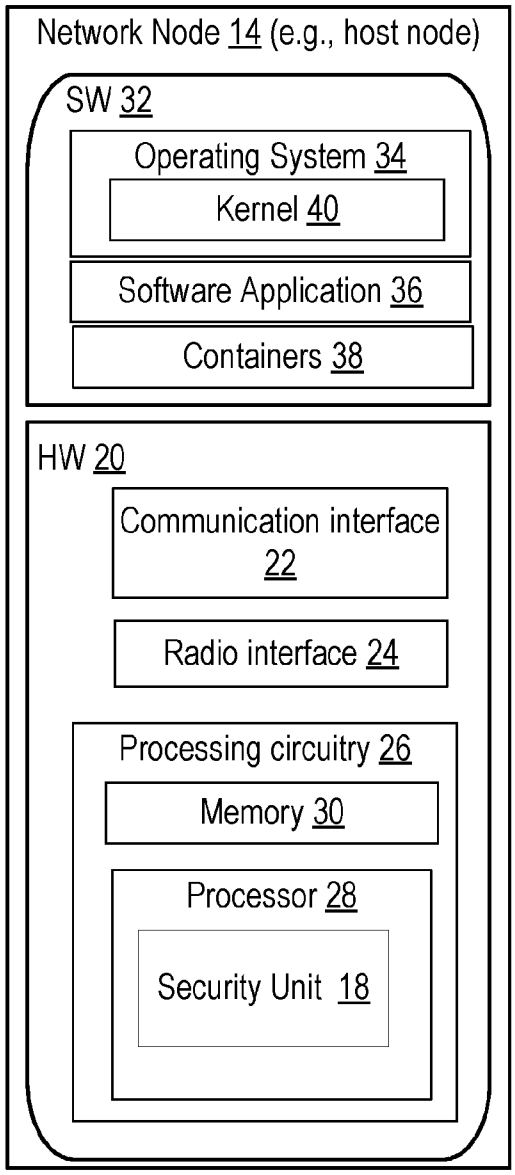
Network Node 14 (e.g., host node)
SW 32
Operating System 34
Kernel 40
Software Application 36
Containers 38
HW 20
Communication interface 22
Radio interface 24
Processing circuitry 26
Memory 30
Processor 28
Security Unit 18

FIG. 2

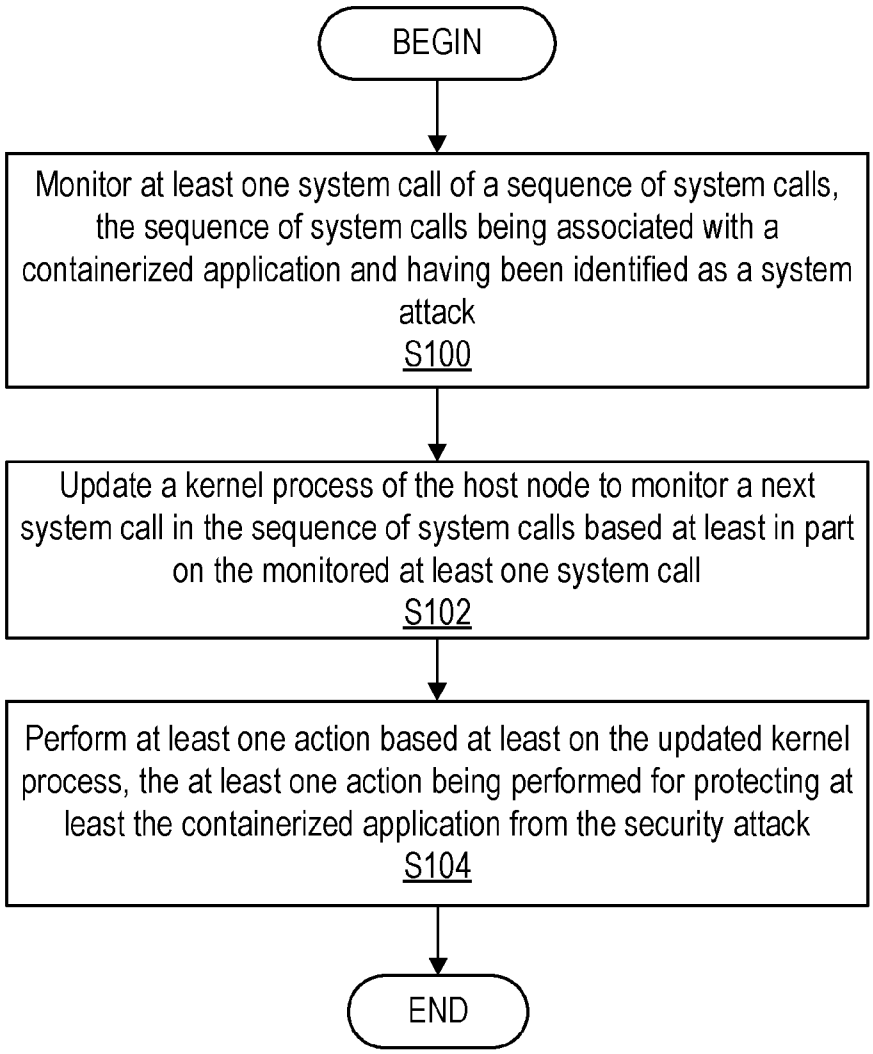
BEGIN
Monitor at least one system call of a sequence of system calls, the sequence of system calls being associated with a containerized application and having been identified as a system attack
S100
Update a kernel process of the host node to monitor a next system call in the sequence of system calls based at least in part on the monitored at least one system call
S102
Perform at least one action based at least on the updated kernel process, the at least one action being performed for protecting at least the containerized application from the security attack
S104
END

FIG. 3

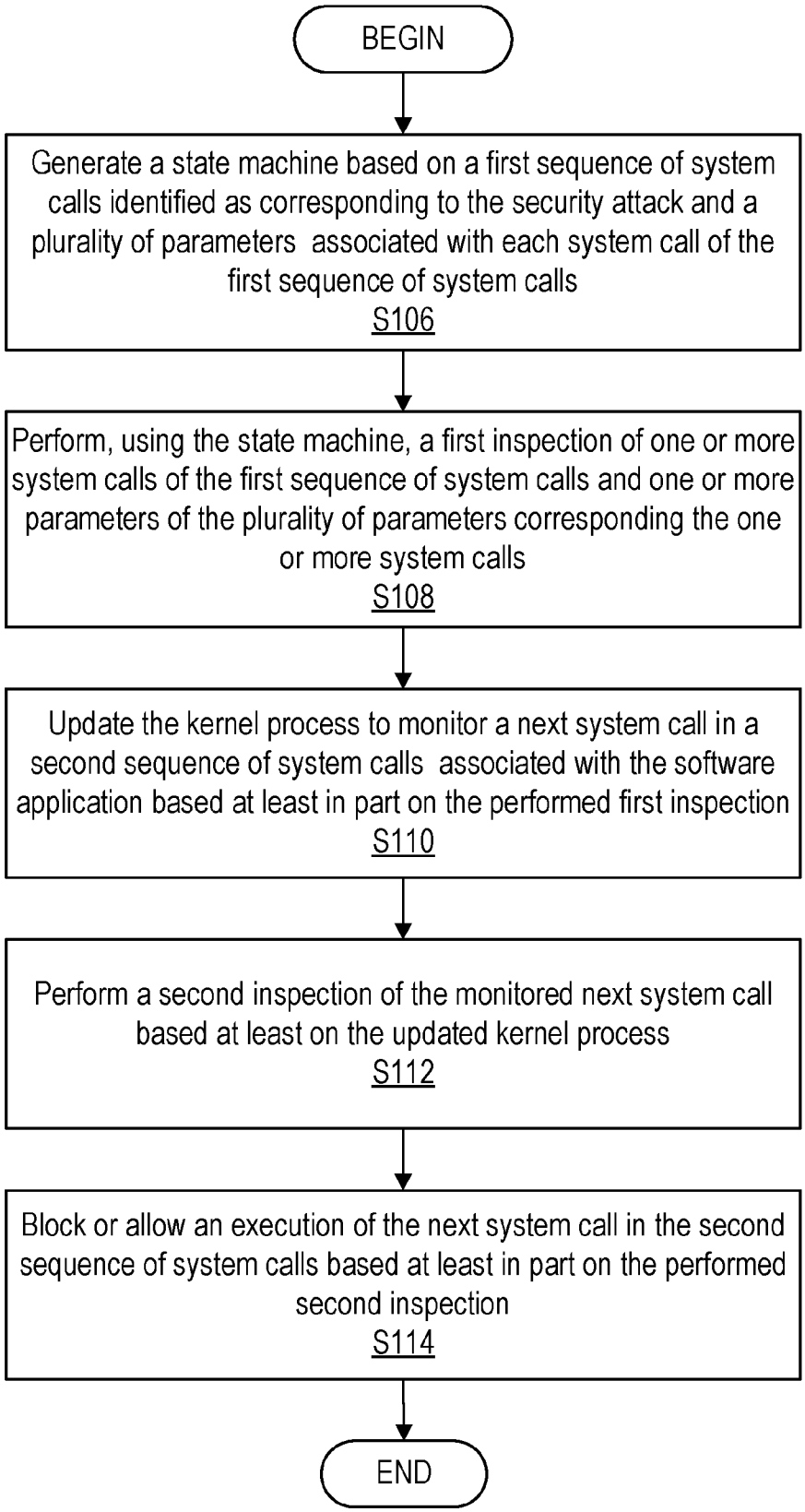
BEGIN
Generate a state machine based on a first sequence of system calls identified as corresponding to the security attack and a plurality of parameters associated with each system call of the first sequence of system calls
S106
Perform, using the state machine, a first inspection of one or more system calls of the first sequence of system calls and one or more parameters of the plurality of parameters corresponding the one or more system calls
S108
Update the kernel process to monitor a next system call in a second sequence of system calls associated with the software application based at least in part on the performed first inspection
S110
Perform a second inspection of the monitored next system call based at least on the updated kernel process
S112
Block or allow an execution of the next system call in the second sequence of system calls based at least in part on the performed second inspection
S114
END

FIG. 4

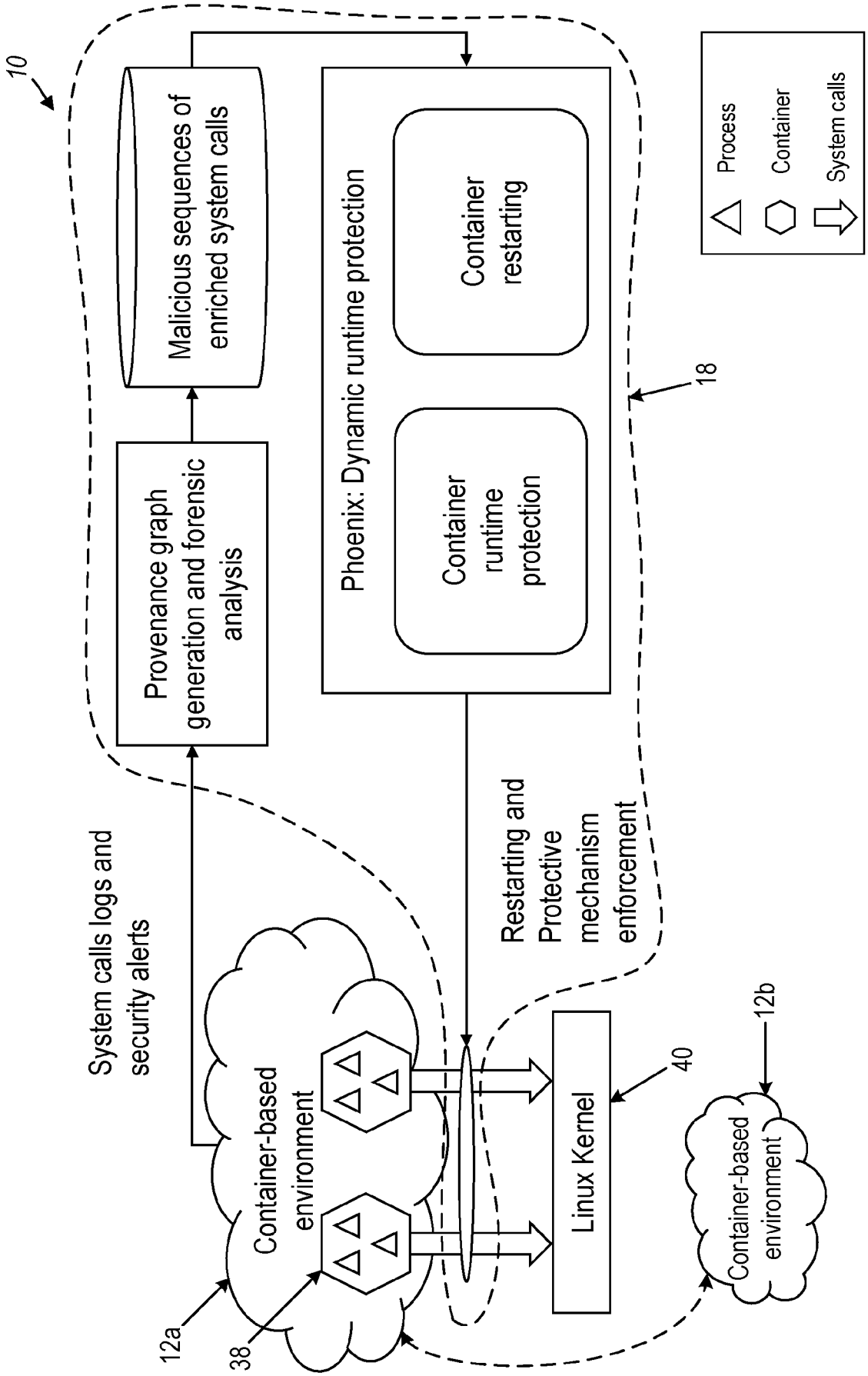
10
Malicious sequences of enriched system calls
Provenance graph generation and forensic analysis
Phoenix: Dynamic runtime protection
Container restarting
Container runtime protection
18
Process
Container
System calls
System calls logs and security alerts
Restarting and Protective mechanism enforcement
Container-based environment
Linux Kernel
40
12b
Container-based environment
12a
38

FIG. 5

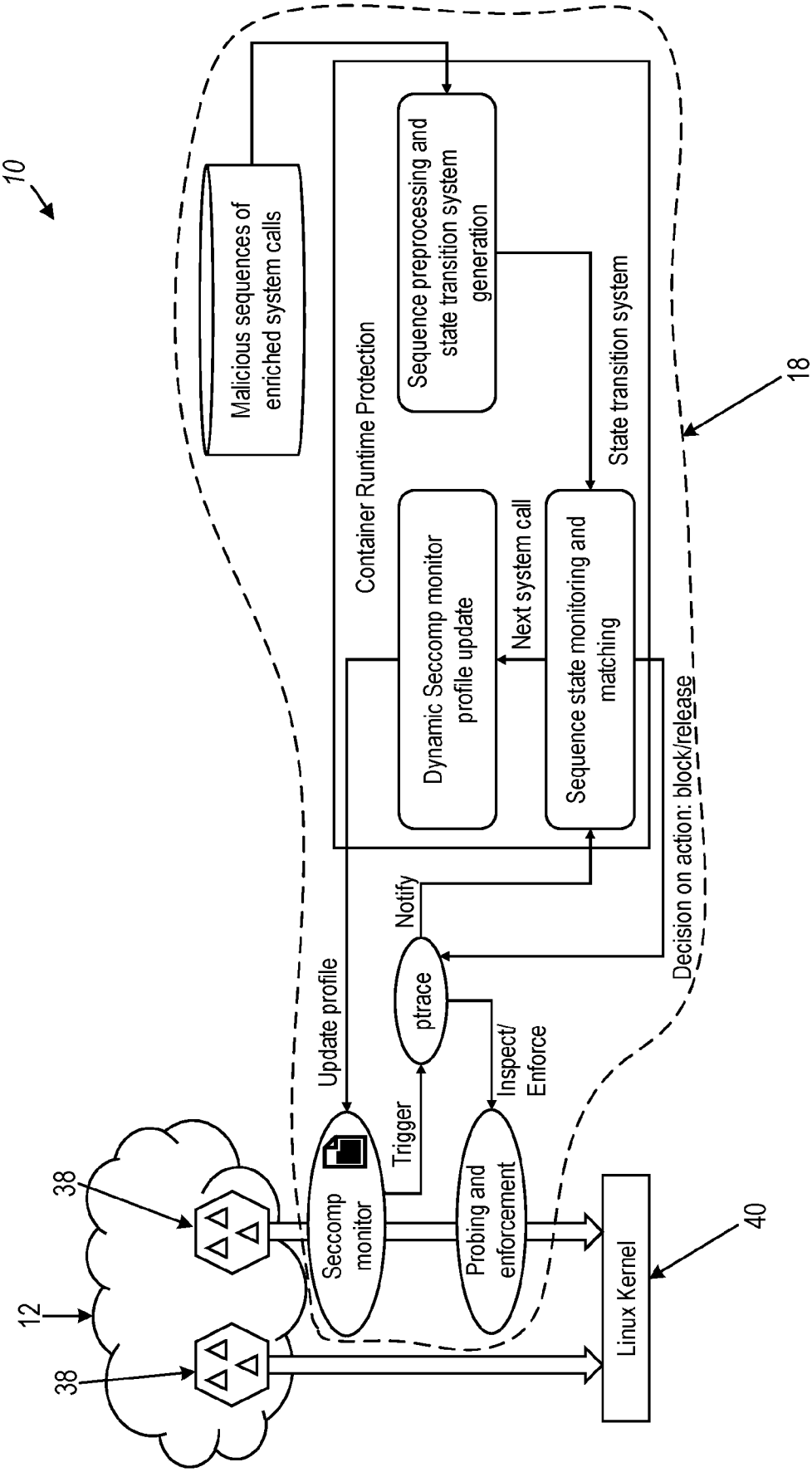
10
Malicious sequences of enriched system calls
Container Runtime Protection
Sequence preprocessing and state transition system generation
State transition system
18
Dynamic Seccomp monitor profile update
Next system call
Sequence state monitoring and matching
Notify
Update profile
ptrace
Inspect/ Enforce
Trigger
Decision on action: block/release
38
Seccomp monitor
Probing and enforcement
12
38
Linux Kernel
40

FIG. 6

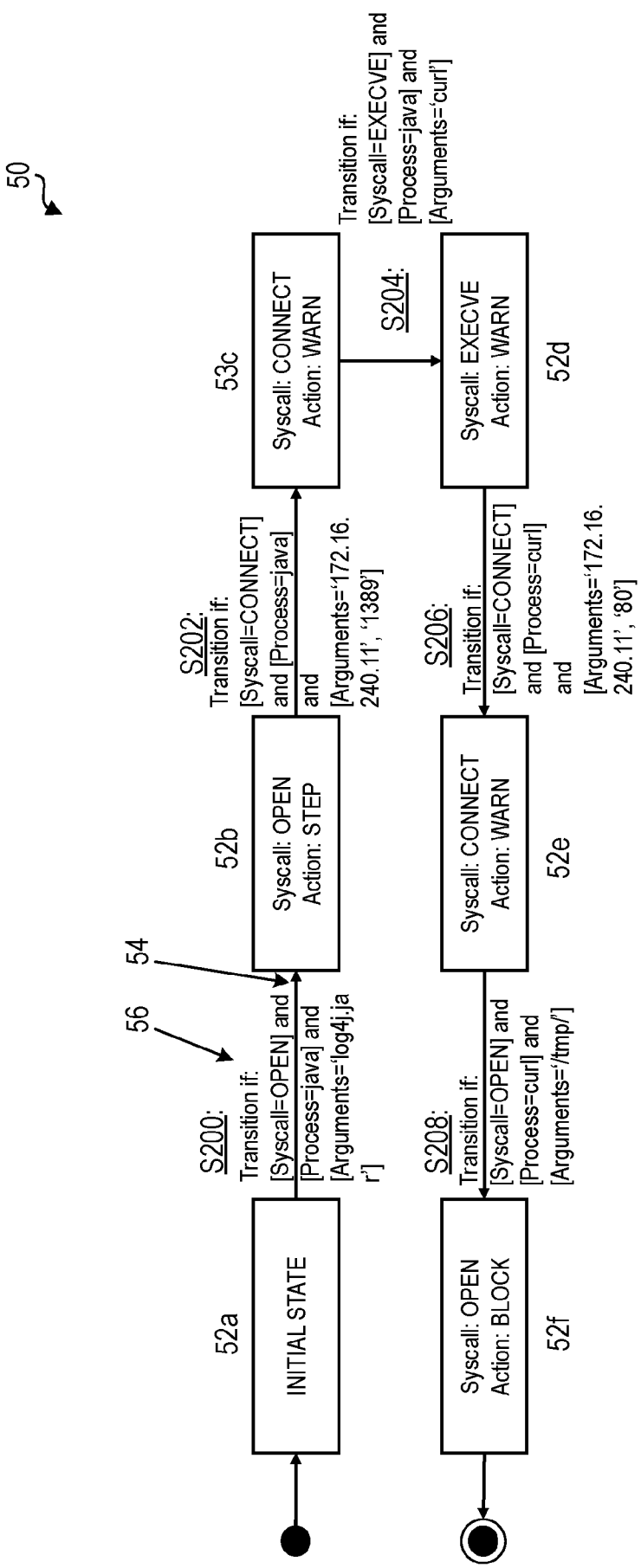
50
52a
INITIAL STATE
S200:
Transition if:
[Syscall=OPEN] and
[Process=java] and
[Arguments='log4j.jar']
54
56
52b
Syscall: OPEN
Action: STEP
S202:
Transition if:
[Syscall=CONNECT]
and [Process=java]
and
[Arguments='172.16.240.11', '1389']
53c
Syscall: CONNECT
Action: WARN
S204:
Transition if:
[Syscall=EXECVE] and
[Process=java] and
[Arguments='curl']
52d
Syscall: EXECVE
Action: WARN
S206:
Transition if:
[Syscall=CONNECT]
and [Process=curl]
and
[Arguments='172.16.240.11', '80']
52e
Syscall: CONNECT
Action: WARN
S208:
Transition if:
[Syscall=OPEN] and
[Process=curl] and
[Arguments='/tmp/']
52f
Syscall: OPEN
Action: BLOCK

FIG. 7

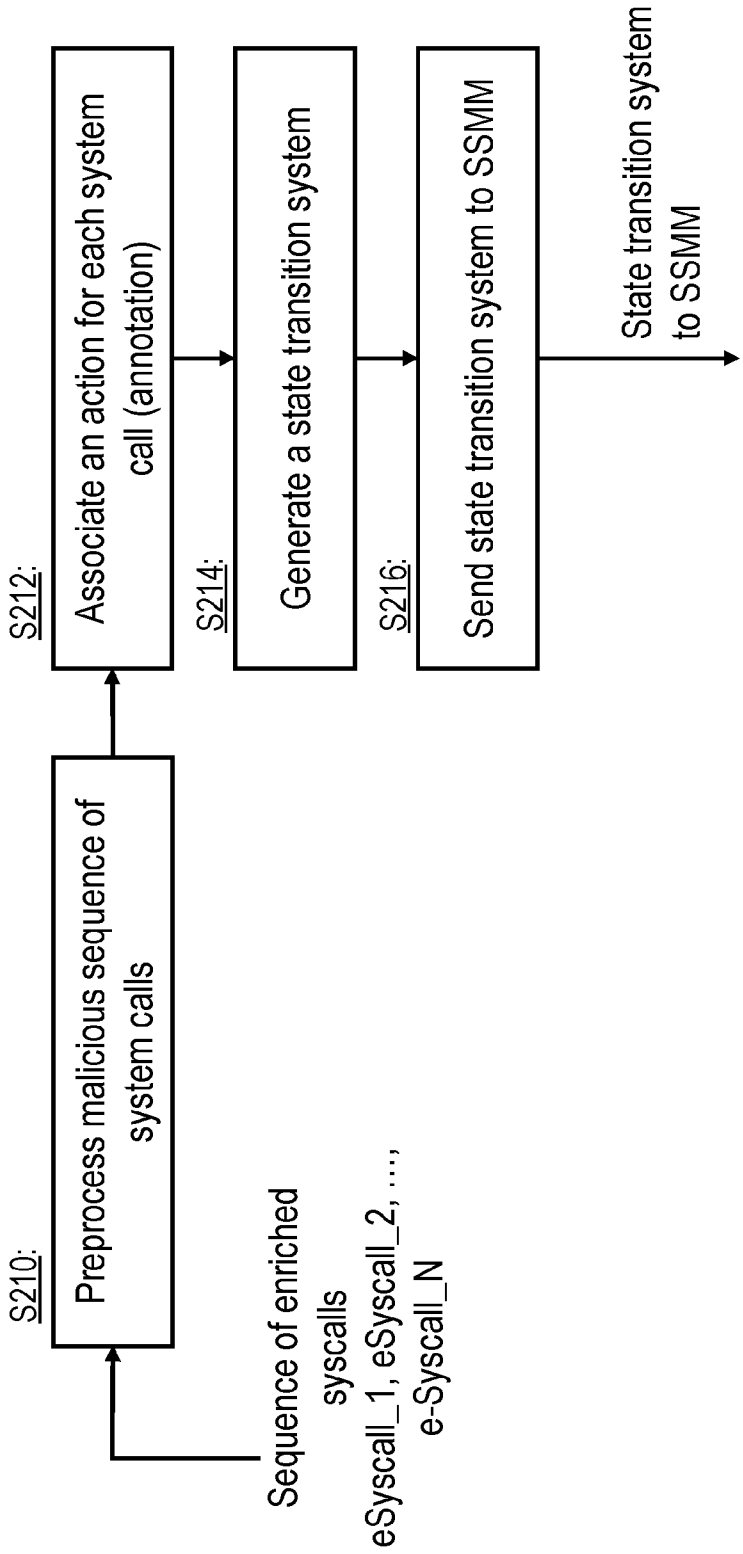
Sequence of enriched syscalls eSyscall_1, eSyscall_2, ..., e-Syscall_N
S210:
Preprocess malicious sequence of system calls
S212:
Associate an action for each system call (annotation)
S214:
Generate a state transition system
S216:
Send state transition system to SSMM
State transition system to SSMM

FIG. 8

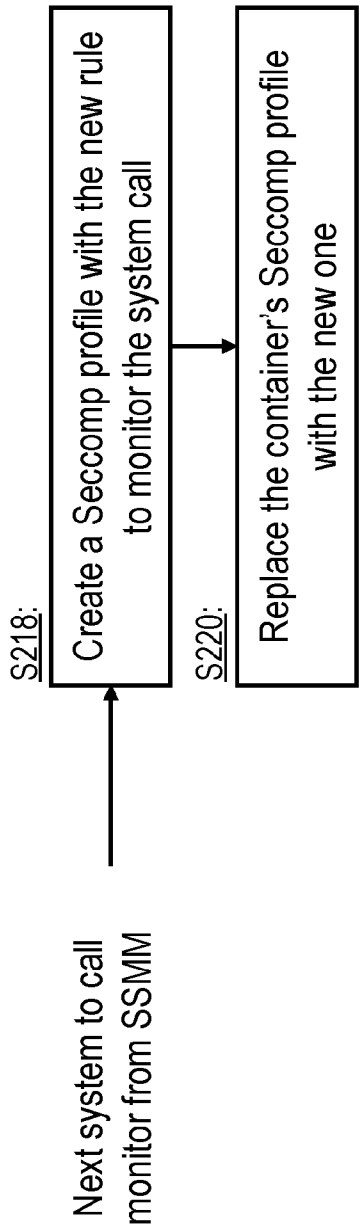
Next system to call monitor from SSMM
S218:
Create a Seccomp profile with the new rule to monitor the system call
S220:
Replace the container's Seccomp profile with the new one

FIG. 9

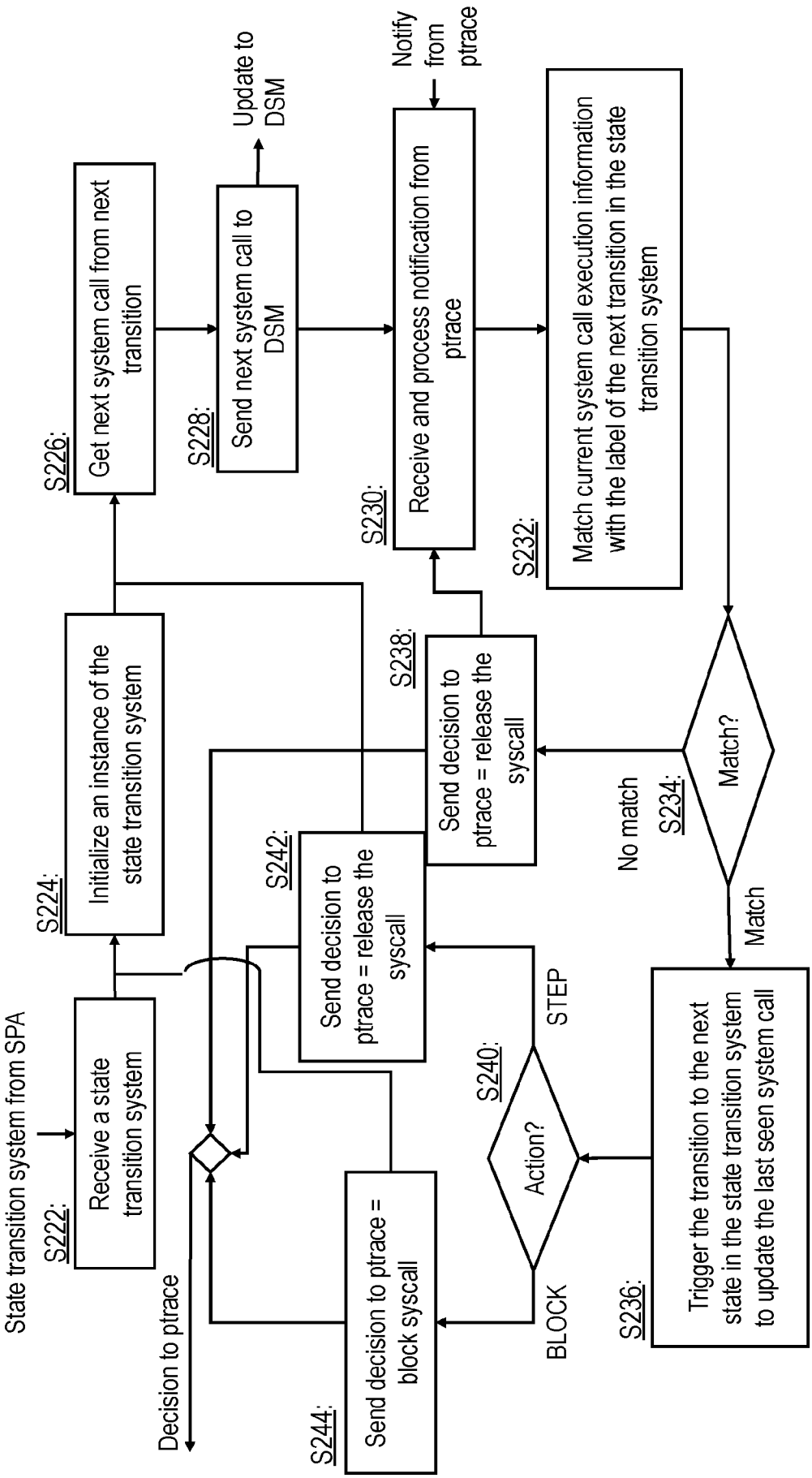
State transition system from SPA
S222:
Receive a state transition system
Decision to ptrace
S224:
Initialize an instance of the state transition system
S226:
Get next system call from next transition
S228:
Send next system call to DSM
Update to DSM
S230:
Receive and process notification from ptrace
Notify from ptrace
S232:
Match current system call execution information with the label of the next transition in the state transition system
S234:
Match?
Match
No match
S236:
Trigger the transition to the next state in the state transition system to update the last seen system call
S238:
Send decision to ptrace = release the syscall
S240:
Action?
STEP
BLOCK
S242:
Send decision to ptrace = release the syscall
S244:
Send decision to ptrace = block syscall

FIG. 10

INTERNATIONAL SEARCH REPORT

International application No
PCT/IB2023/058297

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F21/52 G06F21/55 G06F21/56
ADD.

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

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 11 301 563 B2 (IBM [US]) 12 April 2022 (2022-04-12) column 1, line 59 - column 2, line 10 column 4, line 18 - line 32 column 4, line 54 - column 5, line 11 column 5, line 12 - line 65 column 6, line 31 - line 49 column 13, line 27 - line 38 ----- -/--	1-26

[x] Further documents are listed in the continuation of Box C.

[x] See patent family annex.

* 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	Date of mailing of the international search report
10 November 2023	23/11/2023

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Oliveira, Joel

1

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/IB2023/058297

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CLAUDIO CANELLA ET AL: "SFIP: Coarse-Grained Syscall-Flow-Integrity Protection in Modern Systems", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 28 February 2022 (2022-02-28), pages 1-15, XP091166092, section "A. State-Machine Extraction"; page 5 - page 6 section "C. Installation"; page 7 section "D. Kernel Enforcement"; page 7 - page 8	1-26
X	"Flow-graph analysis of system calls for exploit detection ED - Darl Kuhn", IP.COM, IP.COM INC., WEST HENRIETTA, NY, US, 27 June 2018 (2018-06-27), pages 1-7, XP013179205, ISSN: 1533-0001 abstract page 4, paragraph 2 - paragraph 3 page 5, paragraph 2 - paragraph 4 section "Conclusion"; page 6	1-26
X	FORREST S ET AL: "A sense of self for Unix processes", SECURITY AND PRIVACY, 1996. PROCEEDINGS., 1996 IEEE SYMPOSIUM ON OAKLAND, CA, USA 6-8 MAY 1996, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, 6 May 1996 (1996-05-06), pages 120-128, XP010164931, DOI: 10.1109/SECPRI.1996.502675 ISBN: 978-0-8186-7417-4 abstract page 120, column 2, paragraph 3 - page 121, column 1, paragraph 1 page 121, column 2, paragraph 2 - paragraph 4 section "3.1 Details"; page 122, column 1 page 126, column 1, paragraph 2 section "5. Discussion"; page 126	1-26

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/IB2023/058297

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 11301563 B2	12-04-2022	US 2020293653 A1	17-09-2020
		US 2022207137 A1	30-06-2022