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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 July 2013 (04.07.20 13)

W IP O | P C T

(10) International Publication Number
WO 2013/101083 Al

(51) **International Patent Classification:**
G06F 11/28 (2006.01) ***G06F 21/00*** (2013.01)
G06F 11/30 (2006.01)

(21) **International Application Number:**
PCT/US201 1/067877

(22) **International Filing Date:**
29 December 201 1 (29. 12.201 1)

(25) **Filing Language:** English

(26) **Publication Language:** English

(71) **Applicant** *(for all designated States except US):* **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, MS: RNB-4-150, Santa Clara, CA 95052 (US).

(72) **Inventors; and**
(75) **Inventors/Applicants** *(for US only):* **HIREMANE, Radhakrisha (rk)** [US/US]; 16924 NW Mesa View Ln, Beaverton, OR 97006 (US). **KESHAVAMURTHY, Anil, S** [US/US]; 5365 NW Skycrest Pkwy, Portland, OR 97229 (US).

(74) **Agents: VINCENT, Lester, J.** et al; Blakely, Sokoloff, Taylor & Zafhian LLP, 1279 Oakmead Parkway, Sunnyvale, CA 94085-4040 (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, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Published:

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

(54) Title: AN APPARATUS FOR HARDWARE ACCELERATED RUNTIME INTEGRITY MEASUREMENT

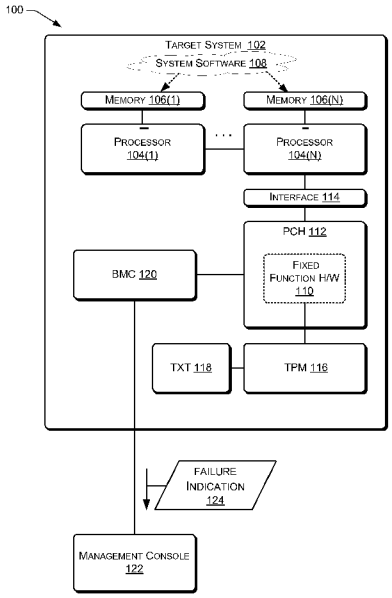

FIG. 1

(57) Abstract: Techniques are described for providing processor-based dedicated fixed function hardware to perform runtime integrity measurements for detecting attacks on system supervisory software, such as a hypervisor or native Operating System (OS). The dedicated fixed function hardware is provided with memory addresses of the system supervisory software for monitoring. After obtaining the memory addresses and other information required to facilitate integrity monitoring, the dedicated fixed function hardware activates a lock-out to prevent reception of any additional information, such as information from a corrupted version of the system supervisory software. The dedicated fixed function hardware then automatically performs periodic integrity measurements of the system supervisory software. Upon detection of an integrity failure, the dedicated fixed function hardware uses out-of-band signaling to report that an integrity failure has occurred. The dedicated fixed function hardware provides for runtime integrity verification of a platform in a secure manner without impacting the performance of the platform.

WO 2013/101083 A1

AN APPARATUS FOR HARDWARE ACCELERATED RUNTIME INTEGRITY MEASUREMENT

TECHNICAL FIELD

[0001] Embodiments described herein generally relate to operation of processors. More particularly, some embodiments relate to dedicated fixed function hardware that automatically performs integrity verification of software at run-time.

BACKGROUND ART

[0002] A hypervisor, also called a virtual machine manager (VMM), is an example of system supervisory software that implements a hardware virtualization technique to allow multiple operating systems, termed guests, to run concurrently on a host computer. Since malware can attack system software at boot-time or at run-time, hypervisor security has become one of the key concerns in the field of virtualization and cloud computing. Run-time manipulations of system supervisory software by malware, that may open backdoors that allow for exploitation of the system supervisory software, have proven to be difficult to detect.

[0003] Technologies such as the Intel ® Trusted Execution Environment may be used to ensure trusted boots of hypervisors. Using these technologies, hypervisors that have been manipulated by malware may be detected, and prevented from booting, during a trusted boot of the hypervisor. Using trusted boot technology, administrators of cloud computing data centers may routinely reboot servers to verify the integrity of the server's hypervisor(s), as well as other system supervisory software hosted by the server. However, periodically rebooting servers may require a high degree of planning and coordination, especially in a large data center, to assure that service requirements are met and service availability in not negatively impacted.

[0004] Software based methods may be implemented to monitor the run-time integrity of system software, such as hypervisors. However, these software based methods may actually steal clock cycles from the central processing unit (CPU) that is being used to execute the hypervisor itself, thus negatively impacting system performance. Alternatively, these software methods may steal clock cycles from other CPUs being used, or that could be used, to provide services by the data center. Additionally, these software methods may be subject to the same malware attacks as the system software they are trying to protect, thus creating a security concern. Consequently, assuring the integrity of system software in a virtualized environment, such as a cloud computing environment, is a costly endeavor.

[0005] Today's processor-based hardware lacks visibility into the run-time integrity of a hypervisor or VMM running on a system with full access privileges. Malware that has attacked the privileged system software can easily hide itself from Anti-Malware agents and hence go undetected at runtime.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] Fig. 1 depicts an example environment usable to perform a runtime integrity measurement of software.

[0007] Fig. 2 depicts an example hardware architecture to perform a runtime software integrity measurement.

[0008] Fig. 3 is a flowchart showing an illustrative method that includes enabling dedicated fixed function hardware to perform run-time integrity verification of system software.

[0009] Fig. 4 is a flowchart showing an illustrative method of performing a run-time integrity verification of system software.

[0010] Fig. 5 is a flowchart showing an illustrative method of preparing for a run-time integrity verification of system software.

DETAILED DESCRIPTION

OVERVIEW

[0011] This disclosure describes embodiments of a dedicated fixed function hardware component (i.e., processing unit) for performing runtime integrity measurements of system software (i.e., low-level system software, system supervisory software or a supervisory software component). In the context of this disclosure, system software may be defined as a hypervisor or virtual machine manager (VMM) running directly on the hardware of the processor (e.g., the bare metal); a native operating system (OS) running directly on the hardware of the processor; a hypervisor, VMM or OS that is hosted by a hypervisor, VMM or OS; system software used to configure and/or manage hardware of a processor, or the like.

[0012] In an embodiment, the dedicated fixed function hardware component may include hardware that is part of the core of a processor. The core of the processor may contain components of the processor involved with the execution of instructions, such as a central processing unit (CPU), an arithmetic logic unit (ALU), a floating-point unit (FPU), level 1 (LI) CPU cache memory, or the like.

[0013] In another embodiment, the dedicated fixed function hardware component may include hardware that is part of the uncore of a processor. The uncore may include hardware components that are not in the core, but which are essential for performing functions of the core. Examples of uncore components may include, for example, an Intel ® QuickPath Interconnect (QPI), a memory controller, a bus controller, or the like.

[0014] In another embodiment, the dedicated fixed function hardware component may include hardware that is part of a chip set associated with a processor. As an example, the dedicated fixed function hardware component may include hardware that is part of a platform controller hub (PCH) microchip, hardware associated with a Peripheral Component Interconnect (PCI) device, or the like.

[0015] In the context of this disclosure, a hardware component specifically includes physical hardware, as opposed to a software component that may merely include executable instructions, such as a computer software program.

[0016] In addition to being a hardware component, the dedicated fixed function hardware component (i.e., fixed function hardware), as described herein, has a hardware structure or hardware architecture that is fabricated to perform a dedicated fixed function.

ILLUSTRATIVE SYSTEM ARCHITECTURE

[0017] Fig. 1 depicts an example environment 100 usable to perform a runtime integrity measurement of system software executing on target system 102. Target system 102 may include a chip set having one or more processors 104(1) to 104(N). Processor 104(1) may have an associated memory 106(1). Memory 106(1) may include any memory accessible by processor 104(1). Memory 106(1) may include level 1 (LI), L2 and/or L3 cache as well as Random Access Memory (RAM), Read Only Memory (ROM), System Management RAM (SMRAM), or the like.

[0018] Processor 104(N) may have an associated memory 106(N). Similar to memory 106(1), memory 106(N) may include any memory accessible by processor 104(N). Memory 106(N) may include LI, L2 and/or L3 cache as well as Random Access Memory (RAM), Read Only Memory (ROM), System Management RAM (SMRAM), or the like. Additionally, processor 104(1) may have full or partial access to memory 106(N) and processor 104(N) may have full or partial access to memory 106(1). Any one or more of processors 104(1) to 104(N) may use at least a part of memory 106(1) to 106(N) to load, boot and/or execute system software 108.

[0019] Dedicated fixed function hardware component 110, used to monitor the run-time integrity of system software 108, is shown in Fig. 1 as

integrated with a platform controller hub (PCH) 112. However, as described above, fixed function hardware component 110 may be part of the core, uncore, an associated chip set (not shown), a PCI device, an edge device associated with target system 102, or the like. PCH 112 may interface to one or more of processors 104(1) to 104(N) via interface 114. For example, interface 114 may be a Direct Media Interface (DMI).

[0020] Dedicated fixed function hardware component 110 may also securely interface to a trusted module, such as trusted platform module (TPM) 116. TPM 116 may securely store trusted information associated with system software 108, such as a trusted initial hash value of system software 108. TPM 116 may also work in conjunction with a hardware extension that includes a trusted execution technology (TXT), such as TXT 118. TXT 118 may be used to facilitate a secure boot of system software 108.

[0021] In an embodiment, during a trusted or secure boot, system software 108 may provide a list of memory addresses to fixed function hardware component 110 via, for example, in-band interface 114. This list of memory addresses may be pointers to static text and/or static data segments of system software 108. In an alternate embodiment, TXT 118 may facilitate the transfer of the list of addresses associated with system software 108 to fixed function hardware component 110.

[0022] After securely booting system software 108, during the run-time execution of system software 108 by at least one of processors 104(1) to 104(N), fixed function hardware component 110 may perform a hashing, for example, of the static text segments and/or static data segments of system software 108, to create a run-time hash value of system software 108. Fixed function hardware component 110 may then retrieve the trusted initial hash value of system software 108 from TPM 116, and compare the trusted initial hash value with the run-time hash value.

[0023] In an embodiment, if the comparing performed by fixed function hardware component 110 indicates an integrity failure of system

software 108, fixed function hardware component 110 may report or indicate the integrity failure to a reporting agent and/or manageability engine, such as baseband management controller (BMC) 120. The integrity measurement failure of system software 108 may then be indicated at management console 122 via failure indication 124. In an embodiment, failure indication 124 may be sent out-of-band. For example, failure report 124 may be sent via an Intelligent Platform Management Interface (IPMI) or BMC protocol 124. This out-of-band reporting of failure indication 124 is advantageous because it may occur independent of system software 108. Additionally, out-of-band reporting of failure indication 124 may provide for an additional layer of integrity failure reporting security, since system software 108 may not have control of, or visibility into, the out-of-band reporting channel.

[0024] In an alternate embodiment, upon detecting an integrity failure of system software 108, fixed function hardware component 110 may send an interrupt to a reporting agent and/or manageability engine (e.g., BMC 120) that informs management software associated with management console 122 of an integrity measurement failure.

ILLUSTRATIVE FIXED FUNCTION ARCHITECTURE

[0025] Fig. 2 depicts an example hardware architecture 200 of dedicated fixed function hardware component 110. As discussed previously, hardware architecture 200 may be implemented as a hardware extension of a microprocessor on one or more respective chipsets associated with target system 102.

[0026] For example, hardware architecture 200 includes one or more dedicated processor(s) 202 that facilitate integrity measurement of system software 108. Dedicated processor(s) 202 are separate from processors 104(1) to 104(N) of target system 102, such that integrity measurements performed by fixed function hardware component 110 do not steal any clock cycles from

processors 104(1) to 104(N). Dedicated processor(s) 202 may also be isolated from target system 102, such that any other components of target system 102 may not directly control or monitor any state or condition of dedicated processor(s) 202. Therefore, fixed function hardware component 110 may perform integrity verification of system software 108 independent of any control or triggering by any other of the components of target system 102. Additionally, dedicated processor(s) 202 may access any of memory 106(1) to 106(N), as well as any other memory locations accessible by target system 102.

[0027] Fixed function hardware component 110 may also include dedicated memory 204. Dedicated memory 204 may only be addressable by dedicated processor(s) 202. As such, dedicated memory 204 is separate and distinct from any of memory 106(1) to 106(N) of target system 102. Dedicated memory 204 is used by fixed function hardware component 110, at least in part, for performing integrity measurements of system software 108. Thus, dedicated memory 204 may not support Direct Memory Access (DMA) by any other components of target system 102, as well as any hardware subsystems external to target system 102.

[0028] Fixed function hardware component 110 may also include dedicated timer 206. Dedicated timer 206 may be used by fixed function hardware component 110 to trigger or initiate an integrity measurement of system software 108. In an embodiment, dedicated timer 206 is a dedicated resource of fixed function hardware component 110, such that dedicated timer 206 may not be controlled or configured by any hardware or software subsystems external to hardware architecture 200. Dedicated timer 206 may operate in a synchronous manner, an asynchronous manner, or both. In an embodiment, dedicated timer 206 may operate under the control of dedicated processor(s) 202. In another embodiment, dedicated timer 206 may operate independent of dedicated processor(s) 202, such that dedicated timer 206 and/or fixed function hardware component 110 may be pre-programmed to periodically perform an integrity measurement of system software 108 in a predefined manner.

[0029] Fixed function hardware component 110 may also include an in-band interface 208 and an out-of-band interface 210. In-band interface 208 and out-of-band interface 210 may be logically or physically separated to facilitate communications with fixed function hardware 110. For example, in-band interface 208 may be used to receive or obtain a range and/or list of memory addresses for integrity monitoring by fixed function hardware 110. Out-of-band interface 210 may be used by fixed function hardware component 110 to indicate an integrity failure, for example, via BMC 120. In-band interface 208, out-of-band interface 210, a combination of in-band interface 208 and out-of-band interface 210, or a standard or proprietary interface (not shown) may be used to facilitate secure communications between fixed function hardware component 110 and TPM 116.

[0030] In an embodiment, after an initial receipt of the range or list of memory addresses for integrity monitoring by fixed function hardware 110, lock-out 212 may be invoked by fixed function hardware 110 to prevent any further reception of memory addresses by fixed function hardware 110.

[0031] For example, during a trusted boot or initial trusted load of system software 108 by target system 102, the complete range or list of memory addresses for integrity monitoring may be provided to fixed function hardware 110 via in-band interface 208. In an embodiment, once the complete range or list of memory addresses are received by fixed function hardware 110, lock-out 212 may be invoked to prevent any further reception of memory addresses and all other information by fixed function hardware 110. In another embodiment, the complete range or list of memory addresses may be followed by an enablement signal, thus invoking lock-out 212 to prevent any further reception of memory addresses by fixed function hardware 110. In these embodiments, lock-out 212 may be used to prevent a corrupted version of system software 108 from providing different memory address to fixed function hardware 110 at a later time.

ILLUSTRATIVE SYSTEM OPERATION

[0032] Figs. 3-5 are example flowcharts illustrating various aspects of the integrity checking system described herein.

[0033] Fig. 3 is a flowchart showing an illustrative method 300 that includes enabling dedicated fixed function hardware to perform run-time integrity verification of system software.

[0034] At 302, a boot of system software 108 is initiated. The system software that is booted may include, but is not limited to, a hypervisor, a VMM, a monolithic kernel (e.g., Linux, Windows), a microkernel, or any other supervisory software that directly controls, for example, the hardware of target system 102.

[0035] At 304, target system 102 determines if dedicated fixed function hardware component 110 is available. In an embodiment, system software 108 performs this determination during a trusted boot. In an alternate embodiment, this determination is facilitated by TXT 118.

[0036] At 304, if it is determined that dedicated fixed function hardware component 110 is not available, control passes to 306 where a normal boot of system software 108 continues. If it is determined that dedicated fixed function hardware component 110 is available, control passes to 308.

[0037] At 308, one or more ranges and/or one or more lists of memory addresses are provided to fixed function hardware 110. In an embodiment, these memory addresses may be associated with static text and/or static data segments of system software 108 used to previously calculate an initial hash value of system software 108. A pointer that points to the initial hash value and/or an associated hash key may also be provided to fixed function hardware 110 before, during or after the memory addresses are provided.

[0038] At 310, fixed function hardware 110 retrieves or reads the initial hash value. In an embodiment, the initial hash value may be retrieved or read from a secure memory store, such as TPM 116. The pointer, as well as the hash key, may be used to facilitate retrieval or reading of the initial hash value from

TPM 116. Fixed function hardware 110 may retrieve or read the initial hash value before, during or after obtaining the memory addresses.

[0039] At 312, fixed function hardware 110 is enabled. In an embodiment, fixed function hardware 110 is self enabled, such that based on obtaining at least one of the memory addresses, the initial hash value pointer, the hash key, or the initial hash value, fixed function hardware 110 determines to enable itself. In another embodiment, fixed function hardware 110 may receive an enabling signal associated with at least one of the memory addresses, the initial hash value pointer, the hash key, or the initial hash value. Fixed function hardware 110 may then become enabled, or enter an enabled state, based on the enabling signal.

[0040] Once enabled, fixed function hardware 110 may activate lock-out 212. Once lock-out 212 is activated, fixed function hardware 110 will no longer accept additional information, such as memory addresses, pointers or hash keys. In an embodiment, fixed function hardware 110 may deactivate lock-out 212 upon detecting a re-boot of system software 108, or upon detecting a re-boot of target system 102. At 306, normal boot of system software 108 continues.

[0041] After booting, during a run-time execution of system software 108, memory locations associated with these memory addresses may be hashed. Assuming system software 108 has not been corrupted, for example, by malware, an appropriate hash of the memory locations obtained during a run-time execution of system software 108 should yield or indicate the initial hash value.

[0042] Fig. 4 is a flowchart showing an illustrative method 400 of an embodiment that includes performing a run-time integrity verification of system software after fixed function hardware 110 is enabled.

[0043] At 402, dedicated timer 206 triggers fixed function hardware 110 to perform an integrity verification measurement of system software 108.

At 404, fixed function hardware 110 accesses memory locations indicated by the previously obtained memory addresses.

[0044] At 406, fixed function hardware 110 calculates a run-time hash value associated with the memory locations. The run-time hash value may include one or more hash values associated with the memory locations. The one or more hash values may be combined, all or in part, in any known fashion, to create the run-time hash value.

[0045] At 408, the run-time hash value is compared to the initial hash value. The comparison is performed to determine if the run-time hash value is appropriately associated with the initial hash value. The comparison may include a numeric or alpha-numeric equivalency comparison, however, the comparison is not so constrained. Any form of comparison may be performed to determine whether the run-time hash value is appropriately associated with the initial hash value. If the comparison indicates that the run-time hash value is appropriately associated with the initial hash value, then the comparison indicates a match between the run-time hash value and the initial hash value. If the comparison indicates that the run-time hash value is not appropriately associated with the initial hash value, then the comparison indicates a mis-match between the run-time hash value and the initial hash value.

[0046] At 410, if the comparison indicates that the run-time hash value matches the initial hash value, then control passes to 412. At 412, fixed function hardware 110 may enter an idle state waiting for the next trigger from dedicated timer 206, indicating the initiation of another integrity verification cycle.

[0047] On the other hand, at 410, if the comparison indicates that the run-time hash value mis-matches the initial hash value, then control passes to 414. At 414, fixed function hardware component 110 may send an interrupt or other signal to a reporting agent and/or a manageability engine (e.g., BMC 120) that causes a display or indication of a run-time integrity measurement failure of system software 108 at, for example, management console 122.

[0048] Fig. 5 is a flowchart showing an illustrative method 500 of an embodiment that includes preparing for a run-time integrity verification of system software.

[0049] **At** 502, fixed function hardware 110 detects that a boot of system software 108 has been initiated. The boot may be an initial boot or a re-boot of system software 108. The boot may also include a trusted boot of system software 108, as facilitated by TXT 118.

[0050] **At** 504, fixed function hardware 110 detects whether lock-out 212 is enabled. If fixed function hardware 110 detects that lock-out 212 is enabled, then control passes to 506. In an embodiment, at 506, fixed function hardware 110 disables lock-out 212 if a boot of system software 108 has been detected. In another embodiment, at 506, fixed function hardware 110 disables lock-out 212 only if a trusted boot of system software 108 has been detected.

[0051] **At** 504, if fixed function hardware 110 detects that lock-out 212 is not enabled, then control passes to 508.

[0052] **At** 508, fixed function hardware 110 obtains memory addresses associated, for example, with static text and static data segments of system software 108. The memory addresses may be provided to fixed function hardware 110 by at least one of system software 108, TXT 118 or target system 102. Fixed function hardware 110 may also obtain the initial hash value associated with system software 108 from a secure store, such as TPM 116. In an embodiment, the initial hash value may have been placed in the secure store prior to booting system software 108. In a different embodiment, the initial hash value may be placed in the secure store during a trusted booting of system software 108.

[0053] **At** 510, fixed function hardware 110 is enabled, which causes fixed function hardware 110 to activate lock-out 212.

[0054] **At** 512, dedicated timer 206 is set to trigger a periodic start of fixed function hardware 110.

[0055] In an embodiment, once dedicated timer 206 is set, fixed function hardware 110 takes over performing integrity measurements of system software 108 by hashing the content of the memory addresses, checking the hash value against the initial hash value, reporting the status when required and automatically retriggering the hashing event at periodic intervals. Fixed function hardware 110 performs these integrity measurements independent of processors 104(1) to 104(N) of target system 102, and without intervention of any external software, including system software 108.

[0056] An advantage of one or more embodiments as described herein is that a fully hardware based approach is fast, because it uses dedicated processor(s) 202 and dedicated memory 204 and does not steal any clock cycles from processors 104(1) to 104(N) during integrity measurements. Additionally, a fully hardware based approach is secure, because once fixed function hardware 110 activates lock-out 212, fixed function hardware 110 will deny any and all attempts to be reprogrammed with a new set of memory addresses or any other related values or pointers during run-time execution of system software 108. The speed and security of fixed function hardware 110 is assured by its independent operation and dedicated nature. Fixed function hardware 110 has total independent control regarding when, and how often, integrity measurements will be performed. Additionally, by utilizing lock-out and secure out-of-band integrity failure reporting, malicious software, such as malware or kernel rootkits, cannot redirect, modify or block the reporting of integrity failures by fixed function hardware 110.

[0057] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims.

[0058] For instance, all optional features of an apparatus or processor described above may also be implemented with respect to the method or process described herein. Specifics in the examples may be used anywhere in one or more embodiments.

CLAIMS

1. A processor comprising logic to:

obtain memory addresses that point to at least a portion of a supervisory software component;

activate a lock-out to prevent obtaining further memory addresses; and

activate a dedicated timer to periodically trigger a run-time integrity measurement of the supervisory software component.

2. The processor of claim 1, wherein the logic to report a run-time integrity failure of the supervisory software component.

3. The processor of claim 2, wherein the logic to:

include a first interface and a second interface;

obtain the memory addresses via the first interface; and

report the run-time integrity failure via the second interface.

4. The processor of claim 1, wherein the logic to obtain the memory addresses during a load-time of the supervisory software component.

5. The processor of claim 1, wherein the logic to:

securely obtain an initial hash value of the supervisory software component from a secure store;

calculate a hash value of contents of memory at the memory addresses; and

compare the hash value to the initial hash value.

6. The processor of claim 5, wherein the logic to detect a run-time integrity failure of the supervisory software component if there is a mis-match between the hash value and the initial hash value.

7. The processor of claim 5, wherein the logic to report a failure indication if there is a mis-match between the hash value and the initial hash value.

8. The processor of claim 1, wherein the logic to obtain the memory addresses that point to at least a portion of the supervisory software component to include at least one of:

a hypervisor or a virtual machine monitor (VMM);

an operating system;

system software used to configure hardware; or

system software used to supervise hardware.

9. The processor of claim 1, wherein the logic to store the memory addresses to point to at least one of a static text segment or a static data segment of the supervisory software component.

10. A method comprising:

obtaining memory addresses of system software by a hardware component of a processor;

activating a lock-out by the hardware component to prevent obtaining further memory addresses; and

activating a dedicated timer of the hardware component to periodically trigger a run-time integrity measurement of the system software.

11. The method of claim 10, further comprising reporting a run-time integrity failure of the system software by the hardware component.

12. The method of claim 11, wherein the obtaining of the memory addresses is to occur in a first band that differs from a second band used for the reporting of the run-time integrity failure.

13. The method of claim 10, wherein the obtaining memory addresses of system software by the hardware component comprises obtaining the memory addresses from a hypervisor or a virtual machine monitor (VMM) directly running on bare metal of the processor or a bare metal operating system natively running on the processor.

14. The method of claim 10, wherein the obtaining memory addresses of system software by the hardware component comprises obtaining the memory addresses that point to at least a portion of a hypervisor, a virtual machine monitor (VMM) or an operating system.

15. The method of claim 10, wherein the obtaining memory addresses occurs during a trusted boot of the system software.

16. The method of claim 10, further comprising:

calculating a hash value associated with the memory addresses by the hardware component; and

comparing the hash value to an initial hash value associated with the system software, wherein the initial hash value includes a hash of at least a portion of the system software.

17. The method of claim 16, wherein the at least a portion of the system software is to include at least one of a static text segment or a static data segment of the system software.

18. The method of claim 16, wherein the detecting the run-time integrity failure comprises detecting a mis-match between the hash value and the initial hash value.

19. The method of claim 16, wherein the initial hash value is securely obtained from a secure store by the hardware component.

20. A system comprising:

a configuration of execution resources to:

obtain memory addresses of system software;

activate a lock-out to prevent obtaining further memory addresses;

activate a dedicated timer to periodically trigger a run-time integrity verification of the system software; and

report a run-time integrity failure of the system software.

21. The system of claim 20, wherein the configuration of execution resources to:

obtain the memory addresses via a first interface; and

report the run-time integrity failure via a second interface.

22. The system of claim 20, wherein the configuration of execution resources to:

securely obtain an initial hash value of the system software from a secure store;

calculate a hash value of contents of memory at the memory addresses; and

compare the hash value to the initial hash value.

23. The system of claim 22, wherein the configuration of execution resources to detect the run-time integrity failure of the system software if there is a mis-match between the hash value and the initial hash value.

24. The system of claim 20, wherein the configuration of execution resources to obtain the memory addresses of the system software to include at least one of:

a hypervisor or a virtual machine monitor (VMM);

an operating system;

software used to configure hardware; or

software used to supervise hardware.

25. The system of claim 20, wherein the configuration of execution resources to obtain the memory addresses that point to at least one of a static text segment or a static data segment of the system software.

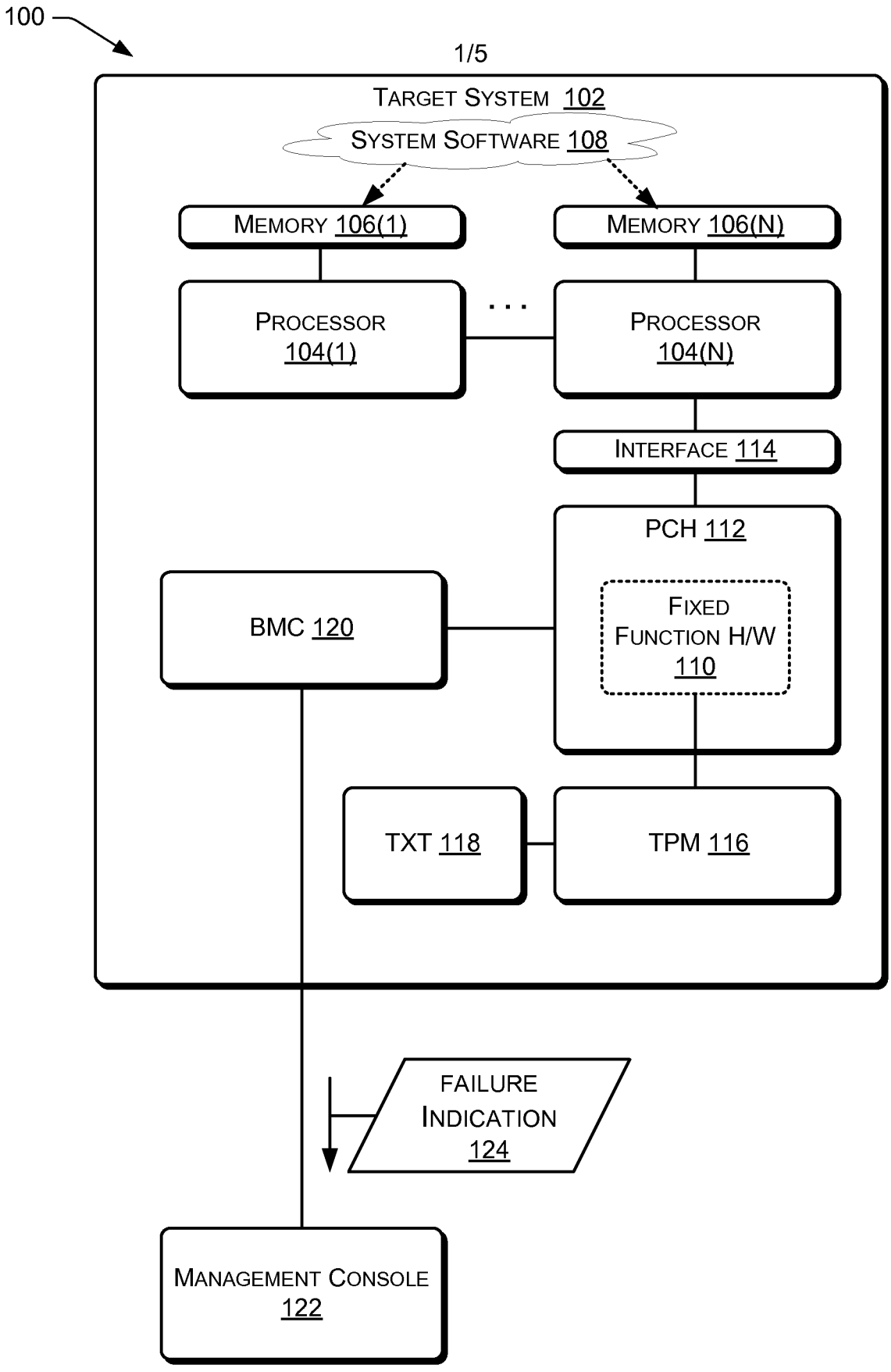
100
TARGET SYSTEM 102
SYSTEM SOFTWARE 108
MEMORY 106(1)
MEMORY 106(N)
PROCESSOR 104(1)
PROCESSOR 104(N)
INTERFACE 114
PCH 112
FIXED FUNCTION H/W 110
BMC 120
TXT 118
TPM 116
FAILURE INDICATION 124
MANAGEMENT CONSOLE 122

FIG. 1

200

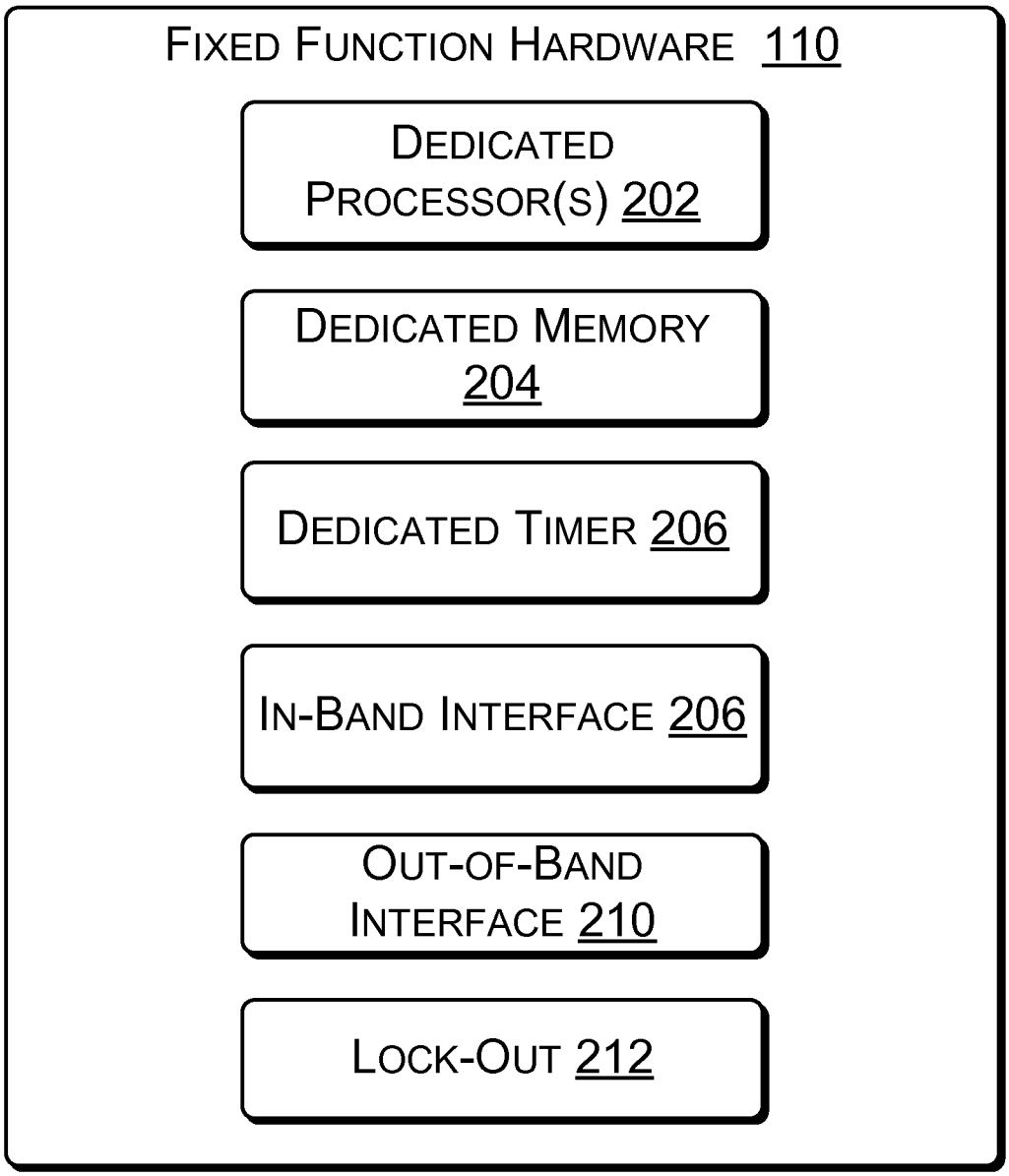
FIXED FUNCTION HARDWARE 110
DEDICATED PROCESSOR(S) 202
DEDICATED MEMORY 204
DEDICATED TIMER 206
IN-BAND INTERFACE 206
OUT-OF-BAND INTERFACE 210
LOCK-OUT 212

FIG. 2

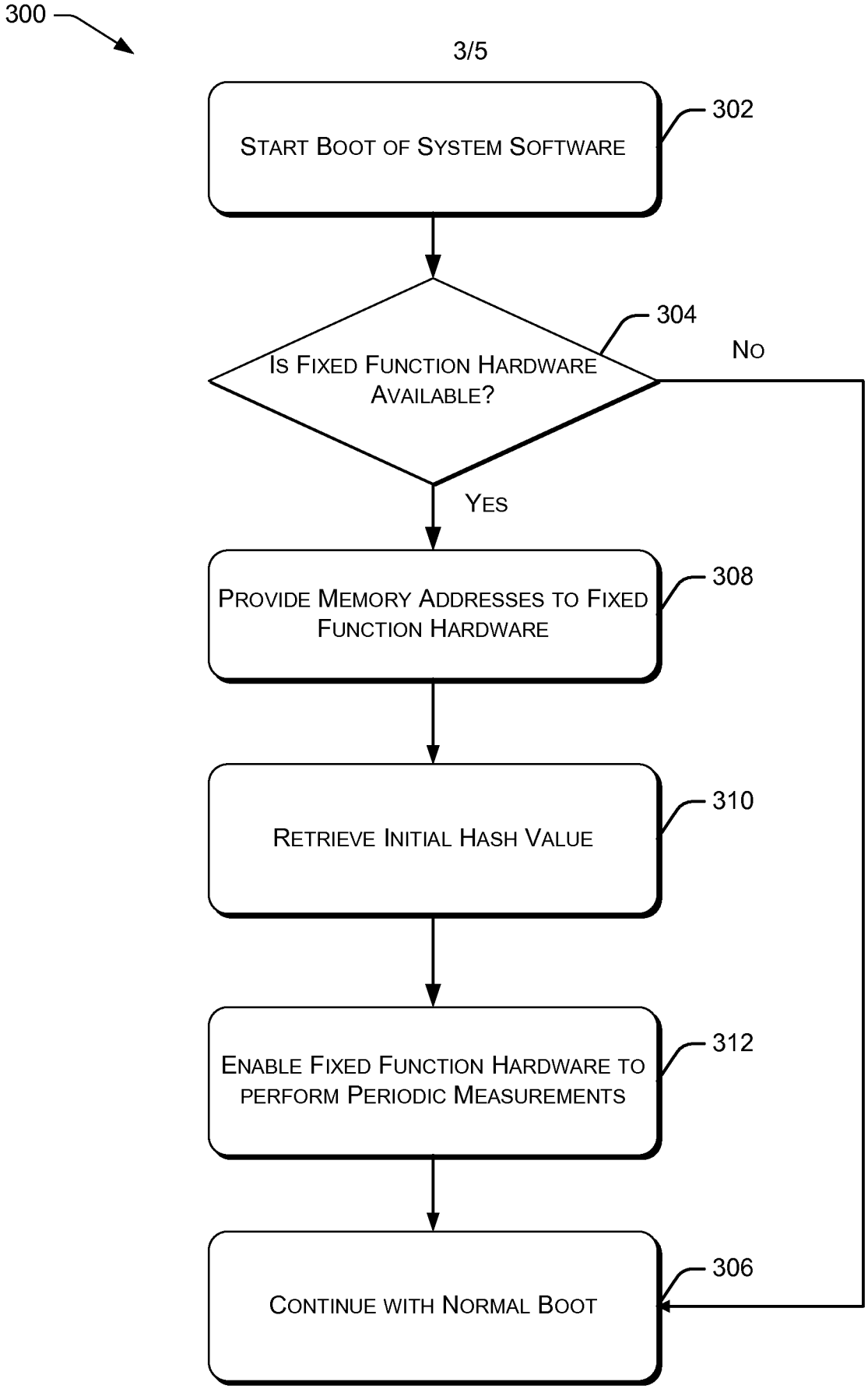
300
START BOOT OF SYSTEM SOFTWARE
302
IS FIXED FUNCTION HARDWARE AVAILABLE?
304
NO
YES
PROVIDE MEMORY ADDRESSES TO FIXED FUNCTION HARDWARE
308
RETRIEVE INITIAL HASH VALUE
310
ENABLE FIXED FUNCTION HARDWARE TO PERFORM PERIODIC MEASUREMENTS
312
CONTINUE WITH NORMAL BOOT
306

FIG. 3

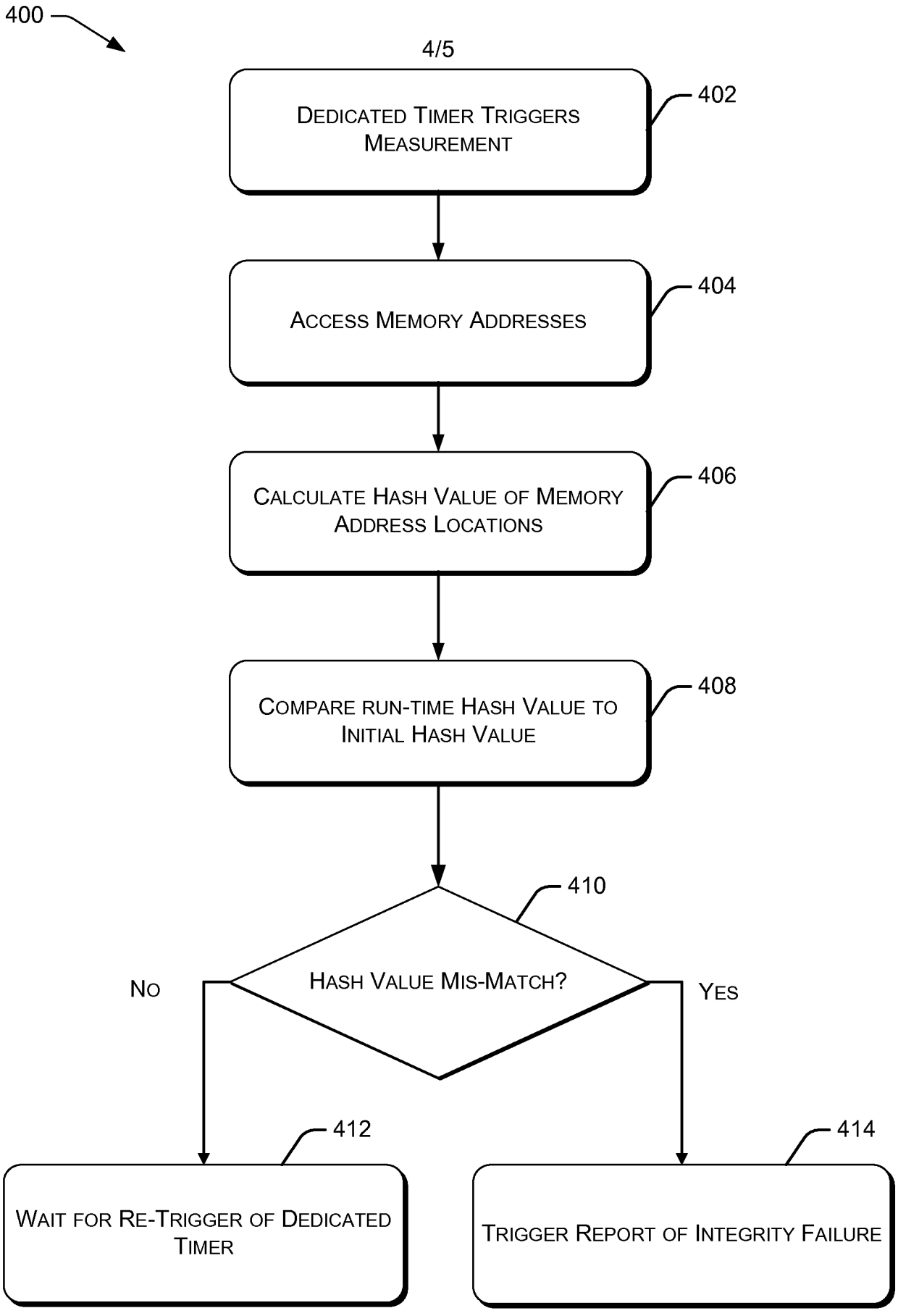
400
DEDICATED TIMER TRIGGERS MEASUREMENT
402
ACCESS MEMORY ADDRESSES
404
CALCULATE HASH VALUE OF MEMORY ADDRESS LOCATIONS
406
COMPARE RUN-TIME HASH VALUE TO INITIAL HASH VALUE
408
410
HASH VALUE MIS-MATCH?
NO
YES
412
WAIT FOR RE-TRIGGER OF DEDICATED TIMER
414
TRIGGER REPORT OF INTEGRITY FAILURE

FIG. 4

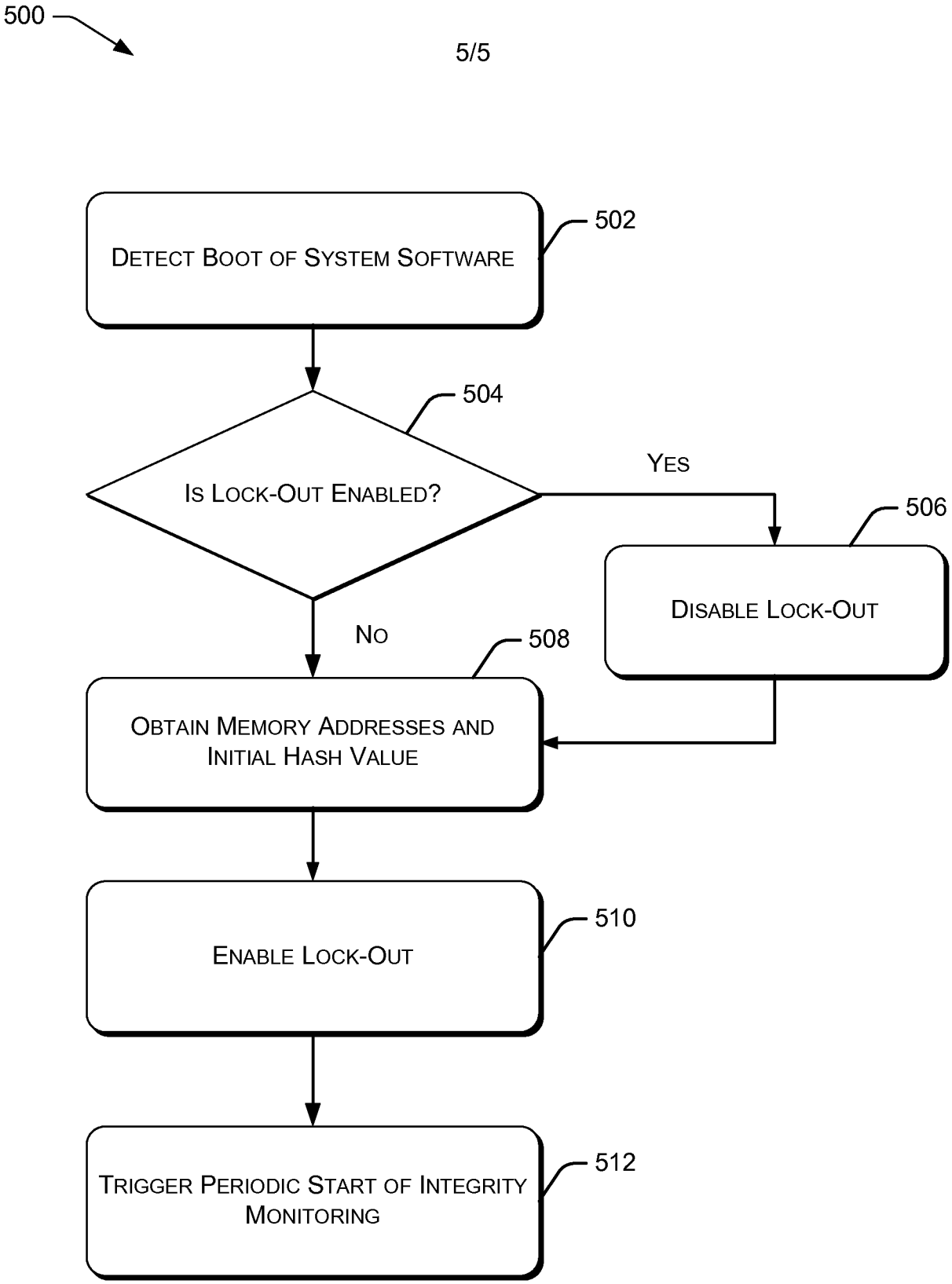
500
DETECT BOOT OF SYSTEM SOFTWARE
502
IS LOCK-OUT ENABLED?
504
YES
DISABLE LOCK-OUT
506
NO
OBTAIN MEMORY ADDRESSES AND INITIAL HASH VALUE
508
ENABLE LOCK-OUT
510
TRIGGER PERIODIC START OF INTEGRITY MONITORING
512

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US201 1/067877

A. CLASSIFICATION OF SUBJECT MATTER

G06F ll/28(2006.01)i, G06F 11/30(2006.01)1, G06F 21/00(2006.01)1

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 1 1/28; G06F 12/02; H04L 1/22; G06F 9/455; G06F 21/00; G06F 1 1/00

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

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

eKOMPASS(KIPO internal) & Keywords: "Detect, runtime, error, failure, hypervisor, periodically"

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2009-0164770 Al (ZIMMER VINCENT J. et al.) 25 June 2009 See abstract; paragraphs [0010]-[0030] and figures 1, 3.	1-25
A	US 2008-0163209 Al (R0ZAS CARLOS V. et al.) 03 July 2008 See abstract; paragraphs [0011]-[0048] and figures 1, 2.	1-25
A	US 2004-0181708 Al (MICHAEL A. ROTHMAN et al.) 16 September 2004 See abstract; paragraphs [0019]-[0043] and figures 2, 5.	1-25
A	US 6697972 Bl (OSHIMA; SATOSHI et al.) 24 February 2004 See abstract; column 4 line 9 - column 5 line 25 and figure 2.	1-25

☐ Further documents are listed in the continuation of Box C.

☒ 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 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
27 SEPTEMBER 2012 (27.09.2012)	**27 SEPTEMBER 2012 (27.09.2012)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan Ci[t]y, 302-70 1, Re[p]ub[l]ic o[f] Korea

Facsimile No. 82-42-472-7140

Authorized officer

Lee By g So

Telep hone N o. 82-42 481 5697

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US20 11/067877

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2009-0164770 A1	25.06.2009	US 7962738 B2	14.06.201 1
US 2008-0163209 A1	03.07.2008	None	
US 2004-0181708 A1	16.09.2004	CN 1754153 A	29.03.2006
		CN 1754153 CO	23.04.2008
		DE 112004000334 T5	09.03.2006
		EP 1606610 A2	21.12.2005
		GB 0513405 DO	03.08.2005
		GB 2414318 A	23.11.2005
		GB 2414318 B	09.08.2006
		JP 04-603487 B2	08.10.2010
		JP 2006-514309 A	27.04.2006
		TW 261748 B	11.09.2006
		US 7318171 B2	08.01.2008
		Wo 2004-08 1920 A2	23.09.2004
		Wo 2004-08 1920 A3	23.09.2004
		wo 2004-08 1920 A3	16.12.2004
		wo 2004-085988 A2	07.10.2004
		wo 2004-085988 A3	07.10.2004
US 6697972 B1	24.02.2004	EP 1094389 A2	25.04.2001
		JP 2001-10 1033 A	13.04.2001
		TW 476877 B	21.02.2002
		us 2004-0153834 A1	05.08.2004
		us 7 134054 B2	07.11.2006