Title: ENSURING SYSTEM INTEGRITY USING LIMITED LOCAL MEMORY

Abstract: A multi-core system comprising a Privileged Processing Core ("PPC"), Non-Privileged Processing Cores ("NPPCs"), a Limited Local Memory ("LLM"), and a main memory.

The LLM is accessible exclusively by the PPC and has at least a first portion of a Secure Operating System ("SOS") and a Secure Memory Paging Mechanism ("SMPM") stored therein. The main memory is accessible by the PPC and NPPCs, and has at least a portion of an Unsecure Operating System ("UOS") stored therein. SOS and SMPM execute exclusively on the PPC, in parallel with the UOS executing exclusively on the NPPCs. SMPM is configured to guarantee the integrity of the SOS.
GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, 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))
ENSURING SYSTEM INTEGRITY USING LIMITED LOCAL MEMORY

REFERENCE TO RELATED APPLICATIONS

[0001] This application is a non-provisional application claiming priority to U.S. Provisional Patent Application Serial No. 61/564,712 filed November 29, 2011, which is hereby incorporated by reference as if fully disclosed herein.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] This invention was made with government support under Grant Nos. CNS-0831268, CNS-0915394, CNS-0931992 and CNS-095128 awarded by the National Science Foundation. Accordingly, the U.S. Government has certain rights in this invention.

[0003] This invention has been supported by the Core Research for Evolutional Science and Technology (CREST) project of the Japan Science and Technology Agency (JST), titled "Dependable Embedded Operating Systems for Practical Use, Dependable Operating Systems for Rich Functional Information Appliances."

FIELD OF THE DISCLOSURE

[0004] The present disclosure relates to secure data processing methods and systems. More specifically, the present disclosure relates to Limited Local Memory ("LLM") architectures and Secure Memory Paging Mechanisms ("SMPMs") employed by such LLM architectures.

BACKGROUND OF THE INVENTION

[0005] System Integrity Monitors ("SIMs"), such as rootkit detectors, rely critically on the ability to fetch and inspect pages containing code and data of the target system under study. SIMs must remain untampered by malicious or compromised targets. State of the art SIMs rely on hypervisor support to set up a tamper-proof inspection environment. In such an environment, the execution of a SIM cannot be compromised even if the target system is infected with malicious code.

[0006] However, there are two main disadvantages to the use of hypervisors. First, modern hypervisors are complex entities and are themselves prone to vulnerabilities and exploits. Modern hypervisors contain thousands of lines of code and vulnerabilities are routinely reported in them. It is no longer reasonable to assume that hypervisors can be easily verified, as is one of
the requirements for an entity that constitutes the Trusted Computing Base ("TCB"). Second, there are application domains for which hypervisor support either is not currently available, or may not see widespread adoption even if available. These domains include embedded systems for special-purpose domains (e.g., sensors on board vehicles and home appliances) and personal devices such as mobile phones, which have also been shown to be vulnerable to attack.

SUMMARY OF THE INVENTION

[0007] The invention concerns implementing systems and methods for providing a tamper-proof execution environment to operating system software. The methods involve: storing at least a first portion of a secure operating system and an SMPM in a first LLM; permitting access to the first LLM exclusively by a privileged processing core; and permitting access to a main memory by the privileged processing core and a plurality of non-privileged processing cores. The secure operating system is executed exclusively on the privileged processing core, while an unsecure operating system is executed exclusively on the plurality of non-privileged processing cores.

[0008] In some scenarios, the methods involve permitting access to each of a plurality of second LLMs by a respective one of said plurality of non-privileged processing cores. The second LLMs may also be accessible to the privileged processing core. During a boot process or normal operation, the privileged processing core may disable at least one non-privileged processing core from accessing a respective second LLM. The privileged processing core may also reset or halt operations of the non-privileged processing cores during use thereof.

[0009] Notably, the privileged processing core acquires control of a multi-core system during the boot process. Accordingly, the privileged processing core facilitates a loading of the unsecure operating system into the shared main memory.

[0010] In some scenarios, a second portion of the secure operating system is stored in the main memory.

[0011] The SMPM is generally operative to prevent guarantee the integrity of the secure operating system. Secure operating system integrity is guaranteed by: authenticating at least one page before being loaded in the first LLM; storing the page which has been authenticated in the first LLM; and allowing the privileged processing core to execute code and access data from the
page which has been loaded in the first LLM. The page may be authenticated by: reading the page from the main memory; computing a hash or checksum value for the page; and considering the page as being authentic if the hash or checksum value matches at least one pre-computed hash or checksum value stored in the first LLM.

[0012] The secure operating system is generally operative to monitor the unsecure operating system for an alert condition. The alert condition comprises at least one of: a violation of an integrity of at least one page of the secure operating system; a violation of at least one integrity policy of the unsecure operating system; and an attempt by the unsecure operating system executing on at least one of the non-privileged cores to access a second LLM associated with the at least one of said non-privileged cores. When an alert condition is detected, the privileged processing core can (a) halt operations of at least one non-privileged processing core and/or (b) acquire control of at least one of peripheral device of a multi-core system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] Embodiments will be described with reference to the following drawing figures, in which like numerals represent like items throughout the figures, and in which:

[0014] FIG. 1 is a block diagram of a conventional multi-core system;

[0015] FIG. 2 is a block diagram of an exemplary multi-core system comprising a plurality of processing cores that is useful for understanding the present invention;

[0016] FIG. 3 is a block diagram of an exemplary memory system;

[0017] FIG. 4 is a flow diagram of an exemplary method for secure memory paging; and

[0018] FIG. 5 is a flow diagram of an exemplary method for providing malware protection to a multi-core system.

DETAILED DESCRIPTION

[0019] It will be readily understood that the components of the embodiments as generally described herein and illustrated in the appended figures could be arranged and designed in a wide variety of different configurations. Thus, the following more detailed description of various embodiments, as represented in the figures, is not intended to limit the scope of the present disclosure, but is merely representative of various embodiments. While the various aspects of
the embodiments are presented in drawings, the drawings are not necessarily drawn to scale unless specifically indicated.

[0020] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by this detailed description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

[0021] Reference throughout this specification to features, advantages, or similar language does not imply that all of the features and advantages that may be realized with the present invention should be or are in any single embodiment of the invention. Rather, language referring to the features and advantages is understood to mean that a specific feature, advantage, or characteristic described in connection with an embodiment is included in at least one embodiment of the present invention. Thus, discussions of the features and advantages, and similar language, throughout the specification may, but do not necessarily, refer to the same embodiment.

[0022] Furthermore, the described features, advantages and characteristics of the invention may be combined in any suitable manner in one or more embodiments. One skilled in the relevant art will recognize, in light of the description herein, that the invention can be practiced without one or more of the specific features or advantages of a particular embodiment. In other instances, additional features and advantages may be recognized in certain embodiments that may not be present in all embodiments of the invention.

[0023] Reference throughout this specification to "one embodiment", "an embodiment", or similar language means that a particular feature, structure, or characteristic described in connection with the indicated embodiment is included in at least one embodiment of the present invention. Thus, the phrases "in one embodiment", "in an embodiment", and similar language throughout this specification may, but do not necessarily, all refer to the same embodiment.

[0024] As used in this document, the singular forms "a," "an," and "the" include plural references unless the context clearly dictates otherwise. Unless defined otherwise, all technical and scientific terms used herein have the same meanings as commonly understood by one of
ordinary skill in the art. Nothing in this disclosure is to be construed as an admission that the embodiments described in this disclosure are not entitled to antedate such disclosure by virtue of prior invention. As used in this document, the term "comprising" means "including, but not limited to."

[0025] As used in this document, a statement that a device or system is "in electronic communication with" another device or system means that devices or systems are configured to send data, commands and/or queries to each other via a communications network. The network may be a wired or wireless network such as a local area network, a wide area network, an intranet, the Internet or another network.

[0026] A "computing device" refers to a computer, a processor and/or any other component, device or system that performs one or more operations according to one or more programming instructions.

[0027] The term "data" may refer to physical signals that indicate or include information. A "data bit" may refer to a single unit of data.

[0028] An "electronic device" refers to a device that includes an imaging device, a processor and tangible, computer-readable memory. The memory may contain programming instructions in the form of a software application that, when executed by the processor, causes the device to perform one or more barcode scanning operations according to the programming instructions. Examples of suitable devices include portable electronic devices such as smart phones, personal digital assistants, cameras, tablet devices, electronic readers, personal computers, media players, satellite navigation devices and the like.

[0029] In the present disclosure, a multi-core system architecture is described. The multi-core system architecture is leveraged to set up a tamper proof execution environment for operating system software. In this regard, the multi-core system architecture leverages recent trends in multi-core chip design to equip each processing core thereof with access to an LLM. The combination of the LLMs and a Secure Memory Paging Mechanism ("SMPM") is sufficient to establish the tamper proof execution environment. The SMPM will be described in detail below. Still, it should be understood that the SMPM is generally configured to guarantee the integrity of the secure operating system.
The utility of the combined LLMs and SMPM mechanism can be demonstrated by building a SIM that leverages the key features of the LLMs. The SIM can safely inspect an unsecure operating system without itself becoming the victim of infection by malware. This feature of such a SIM will become more evident as the discussion progresses.

A schematic illustration of a conventional multi-core system 100 is provided in FIG. 1. The multi-core system 100 includes a plurality of processing cores 102, 104, 106, 108 each connected to a system bus 101 via a respective connection 103, 105, 107, 109. System bus 101 communicatively couples the processing cores 102, 104, 106, 108 to a main memory 110 via a memory bus 111. The main memory 110 is equally shared by and equally accessible to all of the processing cores 102, 104, 106, 108. As such, the processing cores 102, 104, 106, 108 are symmetric, i.e., all of the processing cores are equally privileged and can access all of shared main memory 110 in the same way. Consequently, during a boot process, the processing cores 102, 104, 106, 108 are initialized, for example, in an order determined by a Basic Input/Output System ("BIOS"), boot code or other firmware/software implementation. The software used to define the initialization order of the processing cores is referred collectively herein as a "bootloader."

The bootloader is typically resident on a storage device 130 coupled with the system bus 101. Storage device 130 may be a read only memory ("ROM"), a flash memory, a hard disk, a network storage device, and the like. The bootloader loads the operating system or other run-time environment into main memory 110 for execution on symmetric processing cores 102, 104, 106, 108. In this scenario, all startup functions for the conventional multi-core system 100 are handled by the bootloader. When the operating system or other run-time environment has been fully loaded, operational functions will be handled by the operating system. Since all processing cores 102, 104, 106, 108 are equally privileged, the initialization order and process is a design consideration that may be configured to fit a user's needs.

Referring now to FIG. 2, there is provided a schematic illustration of an exemplary multi-core system 200 that is useful for understanding the present invention. As shown in FIG. 2 the multi-core system 200 comprises a plurality of processing cores 202, 204, 206, 208, a main memory 210, and a secure storage device 230. Secure storage device 230 may be a read only memory ("ROM"), a flash memory, a hard disk, a network storage device, and the like.
Similar to the conventional multi-core system 100, the processing cores 202, 204, 206, 208 are communicatively coupled to the main memory 210 via a system bus 201 and a memory bus 211. The main memory 210 is equally accessible to the processing cores 202, 204, 206, 208. Despite these similarities, multi-core system 200 is different than the conventional multi-core system 100 in at least two ways. First, multi-core system 200 includes a privileged processing core 202 and a plurality of non-privileged processing cores 204, 206, 208. Second, each processing core 202, 204, 206, 208 is equipped with a respective LLM 212, 214, 216, 218. The significance of these two distinguishing features of multi-core system 200 will become evident as the discussion progresses.

Although in FIG 2, the LLMs 212, 214, 216, 218 are provided external to the processing cores 202, 204, 206, 208, the invention is not limited in this regard. For example, at least one of the LLMs 212, 214, 216, 218 may alternatively be embedded within a respective processing core 202, 204, 206, 208. In some scenarios, each LLM 212, 214, 216, 218 is accessible only by the processing core for which it was provided. For example, a privileged LLM 212 is exclusively accessible by privileged processing core 202. Similarly, LLM 214 can be accessed exclusively by non-privileged processing core 204. Likewise, each LLM 216, 218 can be accessed exclusively by non-privileged processing core 206 or 208. Also, physical memory access may be modified so that when a processing core 202, 204, 206, 208 generates an address within a certain predefined range, that address resolves to the LLM area private to that processing core. In other scenarios, the LLMs 214, 216, 218 may each be accessible to the privileged processing core 202. However, in these scenarios, the privileged LLM 212 is not accessible to the non-privileged processing cores 204, 206, 208. For reasons that will become apparent, the only limitation to the described local memory scenarios is that non-privileged processing cores 204, 206, 208 do not have access to the privileged LLM 212. Accordingly, LLMs 214, 216, 218 may be omitted without degrading the security features illustrated by the scenarios described herein.

The privileged processing core 202 operates differently than the non-privileged processing cores in at least two ways. First, the privileged processing core 202 is configured to reset or halt operations of the non-privileged processing cores 204, 206, 208 during use thereof. Accordingly, privileged processing core 202 is the first processing core that is initialized at boot
time, and the last processing core that continues to execute before shutdown of the multi-core system 200. As a consequence of being the first processing core to be initialized, the privileged processing core 202 acquires control of the multi-core system 200 during the boot process. Thereafter, the privileged processing core 202 facilitates the loading of at least a portion of an Unsecure Operating System ("UOS") into the main memory 210 from secure storage 230.

[0037] Second, privileged processing core 202 can disable each non-privileged processing core 204, 206, 208 from accessing its respective LLM 214, 216, 218. If a non-privileged processing core 204, 206, 208 attempts to read data from or write data to its respective LLM 214, 216, 218 when its "LLM access capability" is disabled, a hardware exception is raised, and the read/write operation is handled by the privileged processing core 202.

[0038] For the security framework described herein, only the privileged processing core 202 needs to be equipped with an LLM. However, an LLM may be provided for all processing cores 202, 204, 206, 208 to ease engineering from a hardware design perspective. Hardware vendors may find it easier to design a chip in which all processing cores address memory in the same or similar way. In this scenario, the privileged processing core 202 may be configured to optionally disable certain operations of the non-privileged processing cores 204, 206, 208 for accessing their LLMs 214, 216, 218. Alternatively or additionally, the privileged processing core 202 may be configured to access the LLMs 214, 216, 218 associated with the non-privileged processing cores 204, 206, 208. Accordingly, the LLMs facilitate the provision of a tamper-proof execution environment for operating system software.

[0039] In an exemplary implementation of the present invention, Resident Memory Pages ("RMP") 270 are loaded into privileged LLM 212 during the operation of multi-core system 200. RMP 270 comprises code and data that are resident in the privileged LLM 212. Accordingly, the RMP 270 includes, but is not limited to, at least a portion of a Secure Operating System ("SOS") 272 and a SMPM 274. In some scenarios, SMPM 274 remains resident for the duration of the operation of multi-core system 200 and is never paged to the main memory 210. In some scenarios, SOS 272 is configured to detect rootkits and/or other types of malware in other software installed on multi-core system 200.
During operation, the SOS 272 executes on the privileged processing core 202. In some scenarios, the secure operating system 272 executes exclusively out of privileged LLM 212. As such, a completely secure environment is provided for the execution of SOS 272. If the SOS 272 is too large to be loaded into its entirety into privileged LLM 212, then Secure Memory Pages ("SMP") 222 are loaded into the main memory 210, as shown in FIG. 2. However, embodiments of the present invention are not limited in this regard. Notably, SOS 272 shares main memory 210 with an Unsecure Operating System ("UOS") 260 stored in Unsecure Memory Pages ("UMP") 226 that are loaded in main memory 210. As shown in FIG. 2, multi-core system 200 can be logically split into a privileged subsystem 220 and a non-privileged subsystem 224. The privileged subsystem 220 runs the SOS 272. The non-privileged subsystem 224 runs the UOS 260.

Despite the fact that main memory 210 is shared by all processing cores 202, 204, 206, 208, multi-core system 200 can remain un-tampered when the UOS 260 is infected with malware. The tamper protection is achieved using the SMPM 274, mentioned above. The SMPM 274 is configured to ensure that: (1) all code execution and data access operations performed by the privileged processing core 202 occurs only from pages loaded into the privileged LLM 212; and (2) all code and memory pages are first authenticated before they are loaded into the privileged LLM 212 from main memory 210. As a result of (1) and (2), SMPM 274 can guarantee the integrity of the SOS 272, e.g. prevent an attack that attempts to corrupt SMP 222 while stored on shared main memory 210. Each term "page", "code page" and "memory page", as used herein, refers to a variable or fixed-length contiguous block of memory that is the smallest unit of code or data for (a) memory allocation performed by an operating system and/or (b) transfer between main memory and any other auxiliary data store. The operations of the SMPM 274 will be described in detail below in relation to FIG. 3.

Finally, multi-core system 200 leverages multi-core hardware, allowing the SOS 272 to execute on a dedicated processing core (i.e., the privileged processing core 202), in parallel with the execution of the UOS 260 on the remaining processing cores (i.e., the non-privileged processing cores 204, 206, 208). Such a multi-core configuration provides certain performance benefits because the SOS 272 executes on a dedicated processing core, and therefore does not require any processing resources used by the UOS 260. For example, the SOS 272 may be a
SIM. In this scenario, the UOS 260 may be a user operating system, such as the MICROSOFT WINDOWS operating system. By leveraging the multi-core hardware as described above, the SIM can run on the privileged processing core 202 without taking away resources from the user operating system, which executes on the three remaining processing cores 204, 206, 208.

In various scenarios, SOS 272 and secure subsystem 220 are configured to comply with three invariant rules. First, a memory page to be executed by a privileged processing core 202 must first be loaded into the privileged LLM 212. Because of the limited space available in the privileged LLM 212, the SMP 222 may be loaded in the main memory 210, as described above. Second, a memory page belonging to the SOS 272 must first be loaded into the privileged LLM 212 before it can be read or updated. Third, the memory pages belonging to the SOS 272 must first be verified and authenticated before they are loaded into the privileged LLM 212. These rules are enforced through the operation of SMPM 274, described in more detail below.

Referring now to FIG. 3, there is provided a block diagram that is useful for understanding operations of an SMPM 302. FIG. 3 illustrates the interactions between the SMPM 302, resident on LLM 312, with main memory 310. Both LLM 312 and main memory 310 store code and data in memory pages. For example, code and data loaded in LLM 312 is stored in Secure LLM Memory Pages ("SLMP") 306. As explained above, the code and data stored in SLMP 306 is that required for execution of an SOS executing on a privileged core, e.g. SOS 272 executing on privileged core 202 shown in FIG. 2. Code and data loaded in main memory 310 is likewise stored in memory pages. However, because the SOS shares main memory with a UOS, e.g. UOS 260, the memory pages stored on main memory are accessible the SOS and the UOS. As a result, the malicious software infecting the UOS can corrupt the memory pages storing code and data for the SOS on the main memory. To maintain a secure operating environment for the SOS, all code executed by the SOS must first be authenticated and loaded into the LLM 312. Accordingly, a number of memory pages must be resident in LLM 312 at all times during the operation of the SOS. At minimum, the memory pages required to remain resident in LLM 312 are those that contain code and data necessary to execute SMPM 302 and to store a whitelist 304, which contains a set of hash values, or checksums, for SOS memory pages stored on main memory 310. In some scenarios, the whitelist 304 may be
included within SMPM 302. A more detailed discussion of the operation of SMPM 302 and whitelist 304 follows below.

[0045] SMPM 274 of FIG. 2 can be the same as or similar to SMPM 302. As such, the discussion of SMPM 302 is sufficient for understanding SMPM 274. The operations of SMPM 302 will be described in relation to a privileged LLM 312 and a main memory 310. In some scenarios, privileged LLM 312 may be the same as or similar to privileged LLM 212 of FIG. 2. Similarly, main memory 310 may be the same as or similar to main memory 210 of FIG. 2. SMPM 302 and associated data structures may reside in privileged LLM 312 for the duration of its operation.

[0046] Main memory 310 includes secure memory pages ("SMP") 301 and unsecure memory pages ("UMP") 303. A plurality of memory pages are loaded in privileged main memory 301, e.g. memory pages 314 and 320. The memory pages 314, 320 may include code and data used by an SOS (e.g., SOS 272 of FIG. 2) during operations thereof. As explained above, the SMPM 302 and whitelist 304 may remain resident in LLM 312 for the duration of operation. In this scenario, all other portions of the SOS may be stored in main memory 310 when not in use. For example, the memory pages 314, 320 may include, but are not limited to, code and data necessary for the operation of the SOS but not required to remain resident in LLM 312. One of skill in the art will recognize that portions of the SOS need only be stored in the main memory if the SOS as a whole is too large to be loaded into LLM 312 in its entirety. SMP 301 and UMP 303 may or may not be logically and/or physically contiguous memory regions. Embodiments of the present invention are not limited in this regard.

[0047] In some scenarios, privileged LLM 312 includes two resident memory modules, SMPM 302 and a whitelist 304. SMPM 302 includes all code and data necessary for executing a secure paging scheme of an SOS (e.g., SOS 272 of FIG. 2). The remainder of privileged LLM 312 includes SLMP 306. SLMP 306 includes memory pages 316 and 318, similar to memory pages 314 and 320 described above. Whitelist 304 contains stored hash values, or checksums, for each memory page loaded in SMP 301 and/or SLMP 306, e.g., memory pages 314, 316, 318, 320. The memory pages that store SMPM 302 and the whitelist 304 remain resident in privileged LLM 312 for the lifetime of the system, i.e., they are never paged out into main memory 310 or to a disk.
In some scenarios, an attempt to access memory pages 314, 320 that are stored in the SMP 301 results in a page fault that is handled by SMPM 302. SMPM 302 loads a requested memory page 314, 320 into LLM 312 from main memory 310 only after authenticating the memory page stored in main memory 310. For example, as illustrated by arrow 322, if data is required from memory page 314, SMPM 302 reads the memory page 314 from the main memory 310, computes a hash or checksum value for the memory page 314, and compares the computed hash or checksum value against the contents of whitelist 304 loaded in privileged LLM 312. If there is a match between the computed hash or checksum value and the contents of whitelist 304, then the memory page 314 is loaded into privileged LLM 312, as illustrated by arrow 324. The version of memory page 314 that is loaded into privileged LLM 312 is referred to herein as memory page 316. Memory page 316 can be read, modified or executed, as required. If there is no match between the computed hash or checksum value and the contents of whitelist 304, then SMPM 302 triggers an alert, as discussed below.

Because privileged LLM 312 is limited in space, memory pages 316, 318 may need to be removed therefrom to make room for newly requested memory pages (not shown). SMPM 302 handles such removals by: selecting the memory page to be removed; examining the memory page for any modifications made thereto; computing a hash value if the selected memory page has been modified; storing the computed hash value in the whitelist 304 loaded into privileged LLM 312; copying the contents of the memory page to the shared main memory 310; and/or deleting the selected memory page from the privileged LLM 312. For example, memory page 318 may be selected for removal (or "eviction") from privileged LLM 312. Subsequently, as illustrated by arrow 326, memory page 318 is read from privileged LLM 312 and examined for modification. If memory page 318 has been modified, then a checksum is computed therefore and stored in whitelist 304. The content of memory page 318 is then copied to memory page 320 in privileged main memory 301, as illustrated by arrow 328. The memory page 318 is then deleted from privileged LLM 312. Under normal operation, the memory page 318 is not modified when it is resident in secure main memory 301. Therefore, the computed hash value and checksum will match when the SMPM 302 attempts to load the memory page 318 into privileged LLM 312. In the event that memory page 308 has not been modified, it can simply be deleted from privileged LLM 312.
Referring now to FIG. 4, there is provided a flow chart of an exemplary method 400 for secure memory paging. The secure memory paging is at least partially facilitated by an SOS (e.g., SOS 272 of FIG. 2). In some scenarios, the SOS is loaded into a privileged LLM (e.g., privileged LLM 212 of FIG. 2 or 312 of FIG. 3) and executes on a privileged processing core (e.g., privileged processing core 202 of FIG. 2). The SOS includes an SMPM (e.g., SMPM 274 of FIG. 2 or 302 of FIG. 3) implementing at least a portion of method 400.

As shown in FIG. 4, method 400 begins at 402 and continues with step 404. In step 404, the SOS requests access to a memory page (e.g., memory page 314, 316, 318, or 320 of FIG. 3). The memory page can include, but is not limited to, information that is needed by the privileged processing core to perform one or more of its intended operations as described herein. In some scenarios, the memory page may remain in the privileged LLM for the lifetime of a multi-core system (e.g., multi-core system 200 of FIG. 2). The requested memory page may reside within the privileged LLM or a main memory (e.g., main memory 210 of FIG. 2 or 310 of FIG. 3).

In a next step 406, a determination is made as to whether the requested memory page is stored in the privileged LLM. If the requested memory page is stored in the privileged LLM [406:YES], then access is granted to the memory page, as shown by step 408. Thereafter, the method 400 continues to step 420 and waits for additional processing. The additional processing can include, but is not limited to, receiving a new memory page request, receiving a shutdown command, and the like.

If the requested memory page is not stored in privileged LLM [406:NO], then a page fault is generated by the SOS, as shown by step 410. The page fault occurs as a result of the permissions set in a privileged main memory (e.g., privileged main memory 301 of FIG. 3). Notably, the SOS remains secure because it is executed exclusively out of the privileged LLM. To ensure security, the SMPM sets permissions on data stored in the privileged main memory so that the SOS is unable to access the data until the SMPM has verified the data's authenticity. In this scenario, a page fault is generated within the SOS that is handled by the SMPM.

Next, the SMPM determines if there is a sufficient amount of memory available to store the requested memory page in privileged LLM, as shown by step 412. If the SMPM
determines that there is a sufficient amount of memory available in the privileged LLM for storing the requested memory page [412:YES], then step 414 is performed. In step 414, the SMPM generates a hash or checksum value for the requested memory page. Thereafter, the hash or checksum value is compared to the contents of a whitelist (e.g., whitelist 304 of FIG.3), as shown by step 416.

[0055] If the hash or checksum value matches a value contained in the whitelist [416:YES], then step 418 is performed where the memory page is copied from the main memory and loaded into the privileged LLM. After completing step 418, the method 400 continues to step 420 and waits for additional processing. The additional processing can include, but is not limited to, receiving a new memory page request, receiving a shutdown command, and the like.

[0056] If the hash or checksum value does not match a value contained in the whitelist [416:NO], then the memory page is assumed to be infected or otherwise corrupted. Consequently, the SMPM generates an error message as shown by step 422. The error message is then communicated from the SMPM to a user of the multi-processor system. Upon completing step 422, the method 400 continues to step 420 and waits for additional processing. The additional processing can include, but is not limited to, receiving a new memory page request, receiving a shutdown command, and the like.

[0057] Referring again to step 412, if the SMPM determines that a sufficient amount of memory is not available to store the requested memory page in the privileged LMM [412:NO], then step 424 is performed where the SMPM selects a memory page in the privileged LLM to remove or evict. The selected memory page includes a memory page that is no longer needed for operations by the SOS, and therefore is does not need to be resident in the privileged LLM. One of skill in the art will recognize that any number of known algorithms may be used to determine which memory page to remove or evict. For example, the least recently used memory page may be selected. Alternatively, the oldest memory page may be selected. Any known or to be known algorithm may be used to select a memory page to remove or evict.

[0058] After the memory page is selected in step 424, the SMPM determines in step 426 whether the selected memory page has been modified. This determination may be based on a modification bit value, a hash value or a checksum value. In the modification bit scenarios, a
value of the modification bit can specify whether a modification of the memory page has or has not occurred. In the hash or checksum scenarios, the hash or checksum value may be generated from the memory page and compared to the contents of the whitelist. If the hash or checksum value matches a value contained in the whitelist, then the memory page is considered by the SMPM as being unmodified. In contrast, if the checksum value does not match a value contained in the whitelist, then the memory page is considered by the SMPM as having been modified. Embodiments of the present invention are not limited in this regard. Other techniques for determining if a memory page has been modified may be employed herein without limitation.

[0059] If the SMPM determined that the selected memory page has not been modified [426:NO], then the memory page is deleted from the privileged LLM in step 428. Notably, a copy of the deleted memory page is still present in main memory. Thereafter, at least some of steps 414-422 are performed. Steps 414-422 are described above.

[0060] If the SMPM determines that the selected memory page has been modified [426:YES], then step 430 is performed where the SMPM computes a hash or checksum value for the selected memory page if one was not computed in previous step 424. The computed hash or checksum value is then stored in the whitelist, as also shown by step 430. Subsequent to completing step 430, step 428 is performed. In step 428, the memory page is copied back to the main memory and deleted from the privileged LLM. The method 400 then continues to steps 414-422. Steps 414-422 are described above.

[0061] Referring now to FIG. 5, there is provided a flow chart of an exemplary method 500 for providing malware protection to a multi-core system (e.g., multi-core system 200 of FIG. 2). Method 500 begins with step 502 and continues with step 504. In step 504, a privileged processing core (e.g., privileged processing core 202 of FIG. 2) is initialized. Methods for initializing processing cores are well known in the art, and therefore will not be described herein in detail. Any known or to be known method for initializing a processing core can be used herein without limitation. Still, it should be understood that the initialization of the privileged processing core generally involves: loading an SOS (e.g., SOS 272 of FIG. 2) into memory of the privileged processing core; obtaining any initialization parameters that are to be used by the SOS; and providing the initialization parameters to the privileged processing core. Upon
completing step 504, the privileged processing core performs operations to execute the SOS, as shown by step 506.

[0062] In a next step 508, a privileged LLM (e.g., LLM 212 of FIG. 2) is initialized. The initialization of the privileged LLM generally involves: loading an SMPM (e.g., SMPM 274 of FIG. 2 or 302 of FIG. 3) into the LLM; obtaining any initialization parameters that are to be used by the SMPM; and providing the initialization parameters to the SMPM. The SMPM is configured to bootstrap a tamper-proof environment for the SOS and the user mode processes thereof.

[0063] Subsequent to loading the SMPM, at least one first memory page (e.g., memory 316 and/or 318 of FIG. 3) is loaded into the privileged LLM, as shown by step 510. The first memory page includes, but is not limited to, information required for operation of the SOS. In a next step 512, at least one second memory page (e.g., memory page 314 and/or 320 of FIG. 3) is loaded into a main memory (e.g., SMP 301 of FIG. 3). The second memory page includes information required for operation of the SOS, and which does not fit within the privileged LLM.

[0064] In step 514, the SOS initializes the non-privileged processing cores and loads a UOS (e.g., UOS 260 of FIG. 2) into the main memory (e.g., UMP 303 of FIG. 3). In some scenarios, the SOS may also disable local memory access to all non-privileged processing cores, thereby preventing the UOS from using any non-privileged local memory (e.g., LLMs 214, 216 and/or 218 of FIG. 2). Booting the UOS on the non-privileged processing cores is similar to booting an operating system on a traditional multi-core machine, except that the UOS is initialized on the non-privileged processing cores by the SOS rather than by a bootloader or other bootstrapping firmware. One skilled in the art will recognize that, to boot on an LLM as described above, operating systems may be modified to avoid utilizing privileged main memory that stores the SOS's memory pages. Note that privileged main memory may still be accessible to the UOS. Malicious software that compromises the UOS may attempt to modify the privileged main memory. Any attempted or actual modification of the privileged main memory by the UOS will be detected by the SMPM when memory pages are authenticated before being loading in an LLM.
In step 516, the SOS monitors the UOS for alert conditions. In some scenarios, all peripheral devices connected to the multi-core system are controlled by the UOS. This includes the disk drive, monitor, and all input devices. The SOS does not control any device during the course of normal operation. Because the SOS and the UOS share main memory, the SOS is vulnerable to attacks from a compromised UOS.

The alert conditions may include: (1) a violation of an integrity of one or more memory pages of the SOS; (2) a violation of one or more integrity policies of the UOS; and/or (3) attempts by the UOS to access a non-privileged LLM (e.g., LLM 214, 216 and/or 218 of FIG. 2) in which said non-privileged processing cores do not have permission to access. With regard to alert condition (1), a compromised UOS may attempt to corrupt or otherwise modify the SOS's memory pages stored in the secure main memory. The SMPM can detect attempts by the compromised UOS to corrupt the SOS's memory pages stored in secure main memory. With regard to alert condition (2), a rootkit or other malicious software may compromise the UPS (e.g., by corrupting a system call table or function pointers within kernel mode software), thereby violating code or data integrity of the UOS. If supplied with a suitable policy, the SOS can check the UOS's memory pages to detect any integrity violation. With regard to alert condition (3), the multi-core system can be configured so that the SOS disables local memory access for the non-privileged cores during start-up. Thus, if there are any accesses to non-privileged local memory at runtime, then the UOS is deemed to have been compromised by malicious software. Any such access attempt will raise an exception that will then be handled by the SOS.

In step 518, if one of the alert conditions (1)-(3) is met, then the SOS issues an inter-processor interrupt that will halt the execution of the affected non-privileged processing core(s), returning control to itself. The UOS will therefore be halted and unable to make further changes to shared main memory. At step 520, the SOS acquires control of a peripheral device (e.g., a console, a serial port, or a hard disk) to emit diagnostic information. The diagnostic information may include, but is not limited to, a warning on a console and/or a snapshot of the shared main memory for forensic purposes. At this point, the end-user can take appropriate action. For example, the user may restart the multi-core system or perform operations to clean up the malware infection.
[0068] In summary, the present invention provides a novel multi-core system architecture and SMPM. The multi-core system mitigates at least two problems that have been identified with traditional architectures. In this regard, it should be understood that the multi-core system can provide tamper-proof execution of SEVs without requiring hypervisor support and any associated code complexity. Also, the multi-core system can be used with portable electronic devices (e.g., mobile phones) with only minor modifications to commodity multi-core architectures employed thereby. As such, the barrier to adoption of the present invention by major vendors should be relatively low. Accordingly, the present invention can provide tamper-proof execution of SIMs on such electronic devices without investing the manpower needed to port software solutions thereto.

[0069] The foregoing examples and description of the preferred embodiments should be taken as illustrating, rather than as limiting the present invention as defined by the claims. As will be readily appreciated, numerous variations and combinations of the features set forth above can be utilized without departing from the present invention as set forth in the claims. Such variations are not regarded as a departure from the spirit and script of the invention, and all such variations are intended to be included within the scope of the following claims.
CLAIMS

We claim:

1. A multi-core system, comprising:
   a privileged processing core;
   a plurality of non-privileged processing cores;
   at least one first Limited Local Memory ("LLM") accessible exclusively by said privileged processing core, and having at least a first portion of a secure operating system and a secure paging mechanism stored therein; and
   a main memory accessible by said privileged processing core and said plurality of non-privileged processing cores, and having at least a portion of an unsecure operating system stored therein;
   wherein said secure operating system and said secure paging mechanism execute exclusively on said privileged processing core, in parallel with said unsecure operating system executing exclusively on said plurality of non-privileged processing cores; and
   wherein said secure paging mechanism is configured to guarantee the integrity of said secure operating system.

2. The multi-core system according to claim 1, further comprising a plurality of second LLMs each accessible by a respective one of said plurality of non-privileged processing cores.

3. The multi-core system according to claim 2, wherein said plurality of second LLMs are further accessible by said privileged processing core.

4. The multi-core system according to claim 2, wherein said privileged processing core is configured to disable at least one non-privileged processing core of said plurality of non-privileged processing cores from accessing a respective LLM of said plurality of second LLMs.
5. The multi-core system according to claim 1, wherein said privileged processing core is configured to reset or halt operations of said plurality of non-privileged processing cores during use thereof.

6. The multi-core system according to claim 1, wherein said privileged processing core is configured to
   acquire control of said multi-core system during a boot process, and
   facilitate a loading of said unsecure operating system to said main memory.

7. The multi-core system according to claim 1, wherein at least a second portion of said secure operating system is stored in a plurality of secure memory pages in said main memory.

8. The multi-core system according to claim 7, wherein said secure paging mechanism comprises a whitelist that includes a stored hash value for each of said plurality of secure memory pages in said main memory.

9. The multi-core system according to claim 8, wherein said secure paging mechanism guarantees the integrity of said secure operating system by:
   authenticating a first secure memory page before being loaded in said first LLM;
   loading said first secure memory page which has been authenticated in said first LLM;
   and
   allowing said privileged processing core to execute code and access data from said secure memory page which has been loaded in said first LLM.

10. The multi-core system according to claim 9, wherein said first secure memory page is authenticated by:
    reading said first secure memory page from said main memory;
    computing a first computed hash value for said first secure memory page; and
    considering said first secure memory page as being authentic if said first computed hash value matches a first stored hash value for said first secure memory page.
11. The multi-core system according to claim 9, wherein said secure paging mechanism
further guarantees the integrity of said secure operating system by:
   reading a second secure memory page loaded in said first LLM;
   computing a second computed hash value for said second secure memory page;
   storing said second computed hash value for said second secure memory page in said
   whitelist;
   storing said second secure memory page in said main memory; and
   deleting said second secure memory page from said first LLM.

12. The multi-core system according to claim 9, wherein said secure paging mechanism
further guarantees the integrity of said secure operating system by:
   reading a second secure memory page loaded in said first LLM; and
   on a condition that said second secure memory page has not been modified while loaded
   in said first LLM, deleting said second secure memory page from said first LLM.

13. The multi-core system according to claim 1, wherein said secure operating system is a
system integrity monitor and said unsecure operating system is a user operating system.

14. The multi-core system according to claim 1, wherein said secure operating system is
configured to monitor said unsecure operating system for an alert condition.

15. The multi-core system according to claim 14, wherein said alert condition comprises at
least one of:
   a violation of an integrity of at least one page of said secure operating system;
   a violation of at least one integrity policy of said unsecure operating system; and
   an attempt by said unsecure operating system executing on at least one of said non-
   privileged cores to access a second LLM associated with said at least one of said non-privileged
   cores.

16. The multi-core system according to claim 14, wherein said privileged processing core
(a) halts operations of at least one of said plurality of non-privileged processing cores or (b) acquires control of at least one peripheral device of the multi-core system, when an alert condition is detected.

17. A method for providing a tamper proof execution environment for operating system software, comprising:

storing at least a first portion of a secure operating system and a secure paging mechanism in a first Limited Local Memory ("LLM");

permitting access to said first LLM exclusively by a privileged processing core;

permitting access to a main memory by said privileged processing core and a plurality of non-privileged processing cores;

simultaneously executing said secure operating system exclusively on said privileged processing core, and an unsecure operating system exclusively on said plurality of non-privileged processing cores; and

performing operations by at least said secure paging mechanism to guarantee the integrity of said secure operating system.

18. The method according to claim 17, further comprising permitting access to each of a plurality of second LLMs by a respective one of said plurality of non-privileged processing cores.

19. The method according to claim 18, further comprising permitting access to each of said plurality of second LLMs by said privileged processing core.

20. The method according to claim 18, further comprising performing operations by said privileged processing core to disable at least one non-privileged processing core of said plurality of non-privileged processing cores from accessing a respective LLM of said plurality of second LLMs.

21. The method according to claim 17, further comprising performing operations by said privileged processing core to reset or halt operations of said plurality of non-privileged processing cores during use thereof.
22. The method according to claim 17, further comprising performing operations by said privileged processing core to
   acquire control of a multi-core system during a boot process, and
   facilitate a loading of said unsecure operating system into said main memory.

23. The method according to claim 17, further comprising storing at least a second portion of said secure operating system in a plurality of secure memory pages in said main memory.

24. The method according to claim 23, wherein said secure paging mechanism comprises a whitelist that includes a stored hash value for each of said plurality of secure memory pages in said main memory.

25. The method according to claim 24, wherein said secure paging mechanism guarantees the integrity of said secure operating system by:
   authenticating a first secure memory page before being loaded in said first LLM;
   loading said first secure memory page which has been authenticated in said first LLM;
   and
   allowing said privileged processing core to execute code and access data from said first secure memory page which has been loaded in said first LLM.

26. The method according to claim 25, wherein said first secure memory page page is authenticated by:
   reading said first secure memory page from said main memory;
   computing a first computed hash value for said first secure memory page; and
   considering said first secure memory page as being authentic if said first computed hash value matches a first stored hash value for said first secure memory page.

27. The method according to claim 25, wherein said secure paging mechanism further guarantees the integrity of said secure operating system by:
   reading a second secure memory page loaded in said first LLM;
computing a second computed hash value for said second secure memory page;

storing said second computed hash value for said second secure memory page in said whitelist;

storing said second secure memory page in said main memory; and

deleting said second secure memory page from said first LLM.

28. The method according to claim 25, wherein said secure paging mechanism further guarantees the integrity of said secure operating system by:

reading a second secure memory page loaded in said first LLM; and

on a condition that said second secure memory page has not been modified while loaded in said first LLM, deleting said second secure memory page from said first LLM.

29. The method according to claim 17, wherein said secure operating system is a system integrity monitor and said unsecure operating system is a user operating system.

30. The method according to claim 17, further comprising monitoring said unsecure operating system for an alert condition.

31. The method according to claim 30, wherein said alert condition comprises at least one of:

a violation of an integrity of at least one page of said secure operating system;

a violation of at least one integrity policy of said unsecure operating system; and

an attempt by said unsecure operating system to access a second LLM in which said non-privileged processing cores do not have permission to access.

32. The method according to claim 30, further comprising performing operations by said privileged processing core, when an alert condition is detected, to (a) halt operations of at least one of said plurality of non-privileged processing cores or (b) acquire control of at least one peripheral device of a multi-core system.
FIG. 1
(Prior Art)
FIG. 4

Begin
Request access to a memory page
memory page in local memory?
No
Deny access and generate a page fault

Yes
Grant access to memory page

Select memory page in local memory to evict
memory page modified?
No
Copy and/or delete memory page from local memory

Yes
Compute checksum of selected memory page and store in whitelist and copy memory page to main memory

Generate checksum of memory page stored on main memory
Checksum matches whitelist?
No
Copy memory page from main memory into local memory for execution
Wait for additional processing

Generate output error message
Begin

Initialize privileged processing core at least by loading a secure operating system into memory thereof

Perform operations by the privileged processing core to execute the secure operating system

Initialize a privileged Limited Local Memory ("LLM") at least by loading a Secure Memory Paging Mechanism ("SMPM") therein

Load at least one memory page into the privileged LLM

Load at least one memory page into a secure main memory

By the secure operating system, initialize non-privileged cores and load unsecure operating system into an unsecure main memory

By the secure operating system, monitor unsecure operating system for alert conditions

If alert conditions are met, halt the execution of one or more of the non-privileged cores

Output warning to user

End

FIG. 5
INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER

IPC(8) - G06F 13/00 (201 3.01)
USPC - 7 11/163

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)
IPC(8): G06F 13/00 (2013.01)
USPC: 711/163

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 712/32; 712/1

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Patbase, PubWEST; Google(Scholan Patent); core, multi-core, processor, secure, protected, privilege, LLM, limited local memory, page, paging, memory

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>US 2011/0173363 A1 (Conti et al.) 14 July 2011 (14.07.2011) entire document (especially para [0106]-[0107], [01 14]-[0115], [0118], [0166], [0195], [0198], [0233], [0210], [0239]-[0241], [0251]-[0253], [0262]-[0263], [0534])</td>
<td>1-32</td>
</tr>
<tr>
<td>Y</td>
<td>US 2007/0294496 A1 (Goss) 20 December 2007 (20.12.2007) entire document (especially para [0018H0019], [0099H0100], [0173]-[0174], [0176], [0282], [0297], [0454])</td>
<td>9-21, 25-28</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C.

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

- 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

- 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

- 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

Date of the actual completion of the international search: 20 January 2013 (20.01.2013)

Date of mailing of the international search report: 08 FEB 2013

Name and mailing address of the ISA/US
Mail Stop PCT, Attn: ISA/US, Commissioner for Patents
P.O. Box 1450, Alexandria, Virginia 22313-1450
Facsimile No. 571-273-3201

Authorized officer: Lee W. Young
PCT Helpdesk: 571-272-4300
PCT OGP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)