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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 June 2013 (20.06.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/090044 A1

(51) International Patent Classification:
G06F 21/60 (2013.01) *G06F 9/44* (2006.01)
G06F 15/16 (2006.01)

(21) International Application Number:
PCT/US2012/067660

(22) International Filing Date:
4 December 2012 (04.12.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/323,562 12 December 2011 (12.12.2011) US

(71) Applicant *(for all designated States except US):* **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **BAUMANN, Andrew A.;** c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **HUNT, Galen C ;** c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **PEINADO, Marcus;** c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available):* AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, 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, PA, 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, 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).

Declarations under Rule 4.17 :

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(II))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

(54) **Title:** FACILITATING SYSTEM SERVICE REQUEST INTERACTIONS FOR HARDWARE-PROTECTED APPLICATIONS

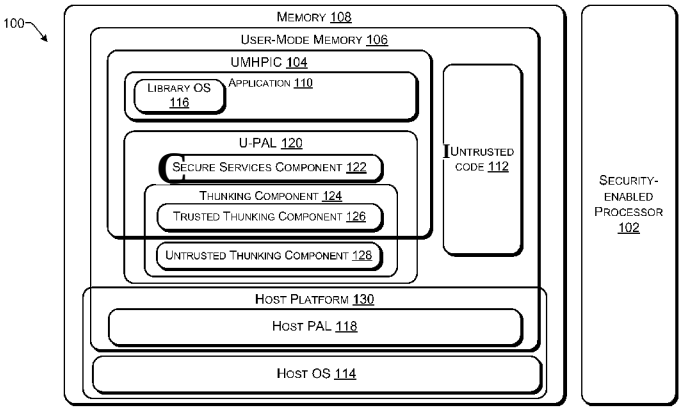

FIG. 1

(57) **Abstract:** Described herein are implementations for providing a platform adaptation layer that enables applications to execute inside a user-mode hardware-protected isolation container while utilizing host platform resources that reside outside of the isolation container. The platform adaptation layer facilitates a system service request interaction between the application and the host platform. As part of the facilitating, a secure services component of the platform adaptation layer performs a security-relevant action.

WO 2013/090044 A1

FACILITATING SYSTEM SERVICE REQUEST INTERACTIONS FOR HARDWARE-PROTECTED APPLICATIONS

BACKGROUND

[0001] Conventional software applications rely on various operating system functions. An operating system may provide a set of application programming interfaces (APIs) for providing basic computational services, such as thread scheduling, memory allocation, virtual memory, device access, and so forth. Additionally, an operating system may provide a rich feature set of APIs that provide additional operating system services such as graphical user interface (GUI) services, clipboard services, and the like.

[0002] Certain security-enabled processors are capable of providing a secure execution environment. Such security-enabled processors provide a protected memory space, and the security-enabled processors ensure that code and data stored in the protected memory space is inaccessible by code outside the protected memory space. The security-enabled processor provides well-defined exit and entry functions, hereafter referred to as gates, that permit execution to pass between code inside the protected memory space and code outside environment. The security-enabled processor does not allow access to input or output devices or kernel-mode execution within the protected memory space. As a result, the protected memory areas of security-enabled processors are too restrictive to run conventional software applications.

BRIEF SUMMARY

[0003] This Summary is provided in order to introduce simplified concepts of the present disclosure, which are further described below in the Detailed Description. This summary is not intended to identify essential features of the claimed subject matter, nor is it intended for use in determining the scope of the claimed subject matter.

[0004] An application executes inside a user-mode hardware-protected isolation container (UMHPIC) provided by a security-enabled processor. A library operating system executing within the UMHPIC as part of the application fulfills most operating system requests, including high-level requests. An UMHPIC-aware platform adaptation layer (U-PAL) enables applications with no special knowledge of the UMHPIC to execute inside the UMHPIC while utilizing operating system resources that reside outside of the UMHPIC. The U-PAL includes a secure services component that executes within the UMHPIC and ensures that thread scheduling, file system interactions, and resource allocations are handled properly on behalf of the application. The U-PAL includes a

thunking component with a trusted portion that resides inside the UMHPIC and an untrusted portion that executes in user mode in the outside execution environment. The trusted thunking component passes system calls to the untrusted thunking component via an exit gate controlled by the security-enabled processor. The untrusted thunking component passes system call results from the operating system to the UMHPIC via an entry gate controlled by the security-enabled processor.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The Detailed Description is set forth with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The use of the same reference numbers in different figures indicates similar or identical items.

[0006] FIG. 1 is a schematic diagram of a secure execution environment for executing an application in an UMHPIC.

[0007] FIG. 2 is a block diagram of an example computing system usable to provide a secure execution environment according to embodiments.

[0008] FIG. 3 is a flow diagram showing an example process for providing security to an application executing in an UMHPIC.

[0009] FIG. 4 is a flow diagram showing an example process for facilitating a resource request interaction between an application executing within an UMHPIC and a host platform executing outside of the UMHPIC.

[0010] FIG. 5 is a flow diagram showing an example process for handling requests for execution and asynchronous notifications from a host platform.

DETAILED DESCRIPTION

Overview

[0011] As discussed above, conventional software applications rely on a rich feature set of operating system functions. Embodiments of the present application enable an application not written specifically to run within a secure execution environment (an unmodified application) to execute within a secure execution environment - hereinafter referred to as a user-mode hardware-protected isolation container (UMHPIC) - without trusting any code outside the secure execution environment. While code outside the UMHPIC, such a host operating system, may deny services to code inside the UMHPIC, code outside the UMHPIC cannot tamper with or snoop on the execution of code within the UMHPIC. Executing a conventional unmodified application within an UMHPIC might normally entail trusting the "host" operating system running outside the UMHPIC.

An unmodified application normally trusts a host operating system for three reasons: first, conventional applications exploit rich and varied services from the operating system; second, an unmodified application is not configured to invoke the specific entry and exit gates of the security-enabled processor to enable execution and data to be passed between the UMHPIC and the outside execution environment; and third, the application relies upon the correct functioning of the operating system for its security. Embodiments of the present Detailed Description also enable a sandboxed environment, such as those described in U.S. App. No. 12/834,895 ("ULTRA-LOW COST SANDBOXING FOR APPLICATION APPLIANCES"), to be executed within an UMHPIC.

[0012] Embodiments described in the present Detailed Description include an UMHPIC-aware platform adaptation layer (U-PAL) that resides partially inside an UMHPIC and partially in the outside execution environment. The U-PAL facilitates system resource request interactions between an application executing in the UMHPIC and a host platform executing outside of the UMHPIC. And, as part of the facilitating, one or more security-relevant actions are performed. The security-relevant actions, among other things, allow the application to utilize the host platform for access to resources even though the host platform executes outside of the UMHPIC, and is therefore "untrusted" code.

[0013] In embodiments, applications include a trusted library operating system inside their UMHPIC; most operating system requests are fulfilled by the library operating system, which invokes services of the U-PAL as needed; the U-PAL in turn invokes services from the host operating system. The trusted portion of the U-PAL (i.e., the portion that resides and executes inside the UMHPIC) presents a service interface to the application executing within the UMHPIC. The untrusted portion of the U-PAL (i.e., the portion that resides and executes in user mode outside of the UMHPIC) interfaces with the operating system, such as through a platform adaptation layer (PAL) provided by the host computer system.

[0014] The U-PAL includes a secure services component and a thunking component. The secure services component executes within the UMHPIC, and is therefore a "trusted" component from the perspective of the application. The secure services component verifies that all system services, including thread scheduling, file system interactions, and resource allocations are handled properly on behalf of the unmodified application thereby allowing for an untrusted host operating system. The thunking component includes both a trusted portion that resides inside the UMHPIC and an untrusted portion that executes in

user mode outside of the UMHPIC. The thunking component provides the trusted portion of the U-PAL, including the secure services component, with access to host operating system functions that execute outside of the UMHPIC. The thunking component and the secure services component together allow an unmodified application, such as an application packaged with a library OS, to execute within an UMHPIC.

[0015] The processes, systems, and devices described herein may be implemented in a number of ways. Example implementations are provided below with reference to the following figures.

Example Environment for Executing an Unmodified Application in an UMHPIC

[0016] FIG. 1 is a schematic diagram of a secure execution environment for executing an application in an UMHPIC. Aspects of system 100 may be implemented on various suitable computing device types that are capable of implementing a secure execution environment for an unmodified application. Suitable computing device or devices may include, or be part of, one or more personal computers, servers, server farms, datacenters, special purpose computers, tablet computers, game consoles, smartphones, combinations of these, or any other computing device(s) capable of storing and executing all or part of a secure execution environment for executing an unmodified application.

[0017] System 100 includes a security-enabled processor 102 configured to instantiate a secure execution environment, including an UMHPIC 104. The UMHPIC 104 resides in a protected memory area within user-mode memory 106 or memory 108. Except through specific entry and exit gates provided by the security-enabled processor 102 that allow the code executing on the UMHPIC 104 to transfer execution and data in and out of the UMHPIC 104, the code and data stored in the UMHPIC 104 is inaccessible to code outside of the UMHPIC 104.

[0018] Threads exit the UMHPIC 104 to return to the unprotected memory through exit gates. Exit gates may be called directly, such as through a call instruction, or indirectly, such as through a processor exception. The UMHPIC 104 may be protected by encryption mechanisms; for example, the security-enabled processor 102 is configured to encrypt on write operations to the UMHPIC 104 and decrypt on read operations. Furthermore, the UMHPIC 104 can be tracked with a cryptographic hash (either inline or out of band) so that the security-enabled processor 102 can verify on read operations that the memory has not been altered. In other embodiments, the memory space that the UMHPIC 104 resides in might be on the same chip of the security-enabled processor 102, and the memory space may be segregated so that the security-enabled processor 102 can

access it when execution enters an entry gate. In various embodiments, the security-enabled processor 102 may be implemented within a single chip, such as a system-on-chip (SOC), or it may be implemented across a processor and additional chipsets of system 100.

[0019] An application 110 executes within the UMHPIC 104. That is, the security-enabled processor 102 executes the application 110 in such a way that, except through well-defined entry and exit gates described in more detail below, prevents code executing outside of the UMHPIC 104, including untrusted code 112 or the host operating system (OS) 114, from tampering with or snooping on code or data inside the UMHPIC 104. For example, a thread executing code stored outside the UMHPIC 104 cannot access data stored in the UMHPIC 104. The security-enabled processor 102 might protect code and data within the UMHPIC for snooping or tampering through various techniques including, but not limited to, encrypting data as it leaves the processor, signing data to detect external tampering, blocking direct memory access (DMA) operations from I/O devices, separating caches, or using memory stored directly on the processor. This provides protection against untrusted code - that is, any code that runs outside of the UMHPIC 104 - snooping on or tampering with contents of the UMHPIC 104.

[0020] Untrusted code 112 executes in user mode outside of the UMHPIC. The security-enabled processor 102 executes the untrusted code 112 in a way that prevents it from accessing code and data within the UMHPIC 104. Multiple UMHPICs are possible, and the security-enabled processor 102 may execute the untrusted code 112 in another UMHPIC. Even if executed in another UMHPIC, the untrusted code 112 is "untrusted" from the perspective of the UMHPIC 104. In some security-enabled processors, UMHPICs may be nested: an inner UMPHIC is protected from code in outer UMHPIC and all code outside the outer UMHPIC. In the case of nested UMHPICs, the code of the outer UMHPIC is untrusted by the inner UMHPIC; code and data of the inner UMHPIC cannot be tampered with or snooped on by code of the outer UMHPIC.

[0021] An initialization process - such as is described in related U.S. Pat. App. No. XXXXXX, filed concurrently, and having attorney reference number MS1-5267US - may be used to initialize the UMHPIC 104 in a way that provides a client system with confidence that the UMHPIC 104 is established with the client's trusted code and data, and with no untrusted code or data, thereby providing the client system with confidence that the application 110 properly executes on system 100.

[0022] The UMHPIC 104 may be instantiated in the context of a hosted computing service, such as a "cloud computing" service, in which a client system initializes the UMHPIC 104 in order to execute the application 110 securely on the cloud computing service. Thus, from the perspective of the client system, all portions of system 100 outside of the UMHPIC 104 and the security-enabled processor 102 are "untrusted." Embodiments of the present application enable the application 110 to execute within the UMHPIC and still access operating system functions from an "untrusted" host operating system, such as the host OS 114.

[0023] In various embodiments, the application 110 includes one or more subcomponents. In common embodiments, one or more of the subcomponents of the application include a library operating system (OS) 116, which may be the same as or similar to the isolated OS subsystems described in U.S. Pat. App. No. 12/834,895, filed July 13, 2010 and entitled "ULTRA-LOW COST SANDBOXING FOR APPLICATION APPLIANCES. U.S. Pat. App. No. 12/834,895 describes "application sandboxes" configured to run various operating system elements (sometimes referred to as a "library OS") within an isolated application process separate from other elements of the operating system. The application sandboxes described in U.S. Pat. App. No. 12/834,895, protect the host operating system from tampering or snooping by the application, but do not necessarily protect the application from tampering or snooping by the host operating system. The isolated OS subsystem described in U.S. Pat. App. No. 12/834,895 provides an application with a rich set of APIs in order to directly provide operating system functions that the application is designed to use. The isolated OS subsystem also utilizes a small subset of application programming interfaces (APIs) to communicate with a host operating system, via an operating system platform adaptation layer (PAL), in order to provide the application with basic computation services. The Host PAL 118 may be the same as, or similar to, the host operating system PAL described in U.S. Pat. App. No. 12/834,895. The host PAL 118 may in embodiments include a trap instruction. In alternative embodiments, the host PAL 118 includes libraries that create high-level abstractions from low-level host OS 114 abstractions. Collectively, the host PAL 118 (if present) and host OS 114 are included in a host platform 130.

[0024] The library OS 116 may be configured to provide the application 110 with a first subset of operating system services (such as, in one non-limiting example, the "rich" set of services described above), and to call the host OS 114 to provide the application 110 with a second subset of operating system services (such as, in one non-limiting example,

the basic computational services described above). Embodiments of the present disclosure are not limited to executing applications, such as the application 110, that are packaged with a library OS such as library OS 116. The host PAL 118 and/or the library OS 116 are omitted in various embodiments. In at least one embodiment, the library OS 116 is omitted, and the application 110 consists of a user-mode processor emulator executing a virtual machine (which in turn may consist of a "guest" operating system and applications).

[0025] An UMHPIC-aware PAL (U-PAL) 120 resides partly within the UMHPIC 104, and partly in the untrusted portion of user-mode memory 106. In various embodiments, the application 110 may be unmodified for execution inside of the UMHPIC 104, and the application 110, in its unmodified version, is configured to interface directly with the host platform 130 via a service interface provided by the host platform 130. The U-PAL 120 therefore facilitates system resource request interactions between the application 110 executing in the UMHPIC and one or more components of a host platform 130 executing outside of the UMHPIC. And, a secure services component 122 of the U-PAL 120 performs one or more security-relevant actions. The U-PAL 120 provides the application 110 with a service interface, within the UMHPIC 104, that emulates the interface provided by the host platform 130. The portion of the U-PAL 120 that resides outside of the UMHPIC 104 is configured to interface with the host OS 114, such as through the host PAL 118 via a platform interface. In alternative embodiments, the host PAL 118 may be a sub-component of the U-PAL 120. In still other embodiments, such as those that omit the library OS 116 and the host PAL 118, the U-PAL 120 emulates all or part of the service interface provided by the host OS 114. For example, the U-PAL 120 may provide a partial or full set of rich APIs that the application 110 is configured to use.

[0026] The U-PAL 120 includes the secure services component 122 and a thunking component 124. The secure services component 122 resides entirely within the UMHPIC 104 and performs the security-relevant actions. The secure services component performs security-relevant actions such as validating handles returned by the host platform, validating memory allocations, ensuring that thread scheduling adheres to expected semantics (e.g. mutual exclusion primitives indeed provide mutual exclusion), validating the results of various calls made to the host platform 130, updating data structures necessary for such validations, encrypting data, protecting the integrity of data by means of digital signatures or message authentication codes, protecting data from replay attacks, securely providing random numbers. The application 110 may be unmodified, and not

designed to execute within the UMHPIC 104 on an untrusted computing system. Thus, the application 110 may have been developed with the assumption that the computing system on which it runs is trusted. Thus, the application 110 is not necessarily configured to validate that the host OS 114 behaves properly. But, because the computing system 100 may be untrusted from the perspective of the UMHPIC 104, the host platform 130 is not assumed to behave properly. Thus, the secure services component 122 performs security-relevant actions that result in a virtual platform that behaves in expected and proper ways.

[0027] In one non-limiting example, the secure services component 122 validates that interface handles returned by the host platform 130 in response to API calls for device access made by the application 110 and/or the library OS 116 are proper. For example, it may determine whether the interface handles are duplicates of interface handles previously provided to the application 110 and/or the library OS 116. Such interface handles may be used to access devices, such as I/O devices, networking devices, or other types of devices. In another non-limiting example, the secure services component 122 validates that memory allocations returned by the host platform 130 are proper. For example, it may ensure that a new memory allocation does not overlap with memory previously allocated to the application 110. Such overlapping memory allocation could cause the application 110 to unexpectedly overwrite its own data, thereby potentially causing it to behave in unpredictable ways.

[0028] In another non-limiting example, the secure services component 122 is configured to protect data to be written to a file on a file system of the host platform 130. This protection may include encryption, adding digital signatures or messages authentication codes and/or measures to protect against replay attacks, such as those described in U.S. Pat. No. 7,421,579, issued to England et al. on September 2, 2008 and entitled "Multiplexing a secure counter to implement second level secure counters"; U.S. Pat. No. 7,065,607, issued to England et al. on June 20, 2006 and entitled "System and method for implementing a counter"; and as described in "Memoir: Practical State Continuity for Protected Modules", by Bryan Parno, Jacob R. Lorch, John R. Douceur, James Mickens, and Jonathan M. McCune, and published in Proceedings of the IEEE Symposium on Security and Privacy, IEEE, May 201 1. The application 110 itself may be configured to write data to a file system in an unprotected form because the application 110, as noted above, may be developed with the assumption that the computing system on which it runs is trusted. Thus, when the application 110 and/or the library OS 116 issue an API call to write data to the file system, the secure services component 122 protects the

data on behalf of the application 110 before passing the protected data in a call to the host platform 130. The secure services component is also configured to unprotect and/or verify data read from the host platform, since the application 110 may not be configured to receive protected data from the file system and since it may not verify digital signatures or message authentication codes and may not check for replay attacks. In another embodiment, the secure services component 122 may implement all file system accesses using a protected virtual file system, possibly using encryption, digital signatures, message authentication codes and or measures to prevent replay attacks; avoiding the exposure of trusted file names and metadata to the host platform 130.

[0029] The application 110 may be configured to request thread scheduling services from the host platform 130. It would be possible, however, for an untrusted misbehaving host platform 130 to schedule threads in such a way as to cause the application 110 to execute incorrectly. For example, if one thread of the application 110 acquires a lock, a misbehaving operating system could allow another thread to concurrently acquire the same lock, thereby causing the threads to operate concurrently and leading to unpredictable results. Thus the secure services component 122 may include a user-mode thread scheduler configured to handle thread scheduling on behalf of the application 110 and/or the library OS 116. Such a user mode thread scheduler may not be required, for example, if the application 110 and the library OS 116 have only one thread of execution. If the application 110 and/or the library OS 116 invoke a call for host functionality related to thread scheduling, the secure services component 122 may determine to handle the thread scheduling call itself, rather than pass it to the host platform 130.

[0030] The thunking component 124 of the U-PAL 120 provides the ability to pass API calls from the secure services component 122 to the host platform 130. A trusted portion of the thunking component, the trusted thunking component 126, executes inside the UMHPIC 104 and is configured to marshal parameters associated with calls to the host platform 130, invoke an exit gate provided by the security-enabled processor 102, and to pass the marshaled parameters out to an untrusted thunking component 128. Thus, the trusted thunking component 126 acts as a transport for calls from the secure services component 122 to the host platform 130.

[0031] The trusted thunking component 126 also receives execution requests from the host platform 130 and results from the calls made to the host platform 130, via an entry gate invoked by the untrusted thunking component 128. The trusted thunking component

126 unmarshals the various parameters associated with those execution requests and results, and passes them to the secure services component 122 for validation.

[0032] The untrusted thunking component 128, also part of the thunking component 124, executes outside of the UMHPIC 104. It is configured to unmarshal the parameters passed to it by trusted thunking component 126 via the exit gate, and to make corresponding calls to the host platform 130 (such as invoking operating system calls to the host OS 114 through the host PAL 118) on behalf of the UMHPIC 104. The untrusted thunking component 128 receives from the host platform 130 execution requests and results from system calls, marshals the corresponding parameters for the execution requests and results, invokes an entry gate, and passes such marshaled components to the trusted thunking component 126. Thus, the untrusted thunking component 128 acts as an entry point for results of calls and execution requests from the host platform 130.

[0033] In a common embodiment, the untrusted thunking component 128 executes in user-mode outside of the UMHPIC 104, and may be invoked by the trusted thunking component 126 through an explicit exit gate. In an alternative embodiment, the untrusted thunking component 128 executes as a subcomponent of the host platform 130, and may be invoked from a system call or trap instruction issued within the trusted thunking component 126.

[0034] The thunking component 124 is configured to invoke the entry and exit functions of the security-enabled processor 102, thereby allowing it to pass parameters and execution between the UMHPIC 104 and the outside execution environment and vice versa. As noted elsewhere, the application 110 may be unmodified and may be unaware that it executes within the UMHPIC 104, and therefore may not be configured to invoke the exit gates of the security-enabled processor 102 in order to pass calls out to the host platform 130. Likewise, the host PAL 118 may also be unmodified to provide support for an application executing in the UMHPIC 104, and therefore may not be configured to invoke the entry gates of the security-enabled processor 102 in order to pass execution requests and the results from calls back to the application 110. Thus, the thunking component 124 provides the ability to invoke the entry and exit gates of the security-enabled processor 102 on behalf of the application 110 and the host PAL 118. The secure services component 122 provides the ability to handle thread execution, protect application data written to untrusted storage and validate host platform 130 behavior on behalf of the application 110 and takes the corresponding security-relevant actions. Together, the thunking component 124 and the secure services component 122 allow the

application 110 to execute within the UMHPIC 104 while still making use of system resources provided by the "untrusted" host OS 114.

Example Computing Device for Providing a Secure Execution Environment

[0035] FIG. 2 is a block diagram of an example computing system usable to provide a secure execution environment according to embodiments. The computing system 200 may be configured as any suitable computing device capable of implementing a secure execution environment. According to various non-limiting examples, suitable computing devices may include personal computers (PCs), servers, server farms, datacenters, special purpose computers, tablet computers, game consoles, smartphones, combinations of these, or any other computing device(s) capable of storing and executing all or part of secure execution environment.

[0036] In one example configuration, the computing system 200 comprises one or more processors 202 and memory 204. The processors 202 include one or more security-enabled processors that are the same as or similar to security-enabled processor 102. The processors 202 may include one or more general-purpose or special-purpose processors other than a security-enabled processor. The computing system 200 may also contain communication connection(s) 206 that allow communications with various other systems. The computing system 200 may also include one or more input devices 208, such as a keyboard, mouse, pen, voice input device, touch input device, etc., and one or more output devices 210, such as a display, speakers, printer, etc. coupled communicatively to the processor(s) 202 and memory 204.

[0037] Memory 204 may store program instructions that are loadable and executable on the processor(s) 202, as well as data generated during execution of, and/or usable in conjunction with, these programs. In the illustrated example, memory 204 stores an operating system 212, which provides basic system functionality of the computing system 200 and, among other things, provides for operation of the other programs and modules of the computing system 200. The operating system 212 may be the same as or similar to the host OS 114.

[0038] Portions of memory 204 may be included within an UMHPIC as is described elsewhere within this Detailed Description. Memory 204 may be divided between memory on the same physical chip as the processor and memory on other chips. Memory 204 includes a U-PAL 214, which may be the same as or similar to the U-PAL 120. Memory 204 includes an application 216, which may be the same as or similar to the application 110. The application 216 may include a library OS as described elsewhere

within this Detailed Description configured to interface with a host PAL 218, which may be the same as or similar to the host PAL 118. Memory 204 includes untrusted code 220, which may be the same as or similar to untrusted code 112.

Example Operations for Execution of an Application in an UMHPIC

[0039] FIG. 3 is a flow diagram showing an example process 300 for providing security to an application, such as application 110, executing in an UMHPIC, such as UMHPIC 104. At 302, an application executes inside the UMHPIC, as is described elsewhere within this Detailed Description.

[0040] At 304, a U-PAL, such as U-PAL 120, facilitates a resource allocation request interaction between the application executing inside the UMHPIC and a host platform executing outside of the UMHPIC. The facilitating may include passing requests from the application, and results from the host platform, out of and into the UMHPIC, such as is described elsewhere within this Detailed Description. The facilitating may include invoking the entry and exit gates and marshaling parameters as are described elsewhere within this Detailed Description. The facilitating may include passing system resource requests to, and receiving associated results from, a host platform.

[0041] At 306, a secure services component of the U-PAL, such as the secure services component 122, performs, as part of the facilitating, a security-relevant action. The security-relevant action may include, in various embodiments, encrypting data from the application associated with the system service request interaction; decrypting data associated with the system service request interaction for the application; maintaining a bookkeeping data structure to store information regarding the system service request interaction; verifying the correctness of a resource allocation result that is provided by the host platform as part of the system service request interaction; checking for overlapping memory allocations; determining whether the device handle is a duplicate device handle; performing thread scheduling services, and so forth as described elsewhere within this Detailed Description. Other security-relevant actions may be performed without departing from the scope of the present disclosure.

[0042] FIG. 4 is a flow diagram showing an example process 400 for facilitating a resource request interaction between an application executing within an UMHPIC and a host platform, such as the host platform 130, executing outside of the UMHPIC. At 401, an application executes in the UMHPIC.

[0043] At 402, an application executing in an UMHPIC issues a call for a system service. The UMHPIC is provided by a security-enabled processor, such as the security-

enabled processor 102. The security-enabled processor provides at least one exit gate for passing execution out of the UMHPIC and at least one entry gate for passing execution into the UMHPIC. The call may come from any subcomponent of the application including a library OS, such as the library OS 116, and the application and any subcomponents, including the library OS, may be unmodified to execute within the UMHPIC and may have no native capability to transfer execution outside of the UMHPIC.

[0044] At 404, a secure services component of a U-PAL, such as the secure services component 122 of the U-PAL 120, receives the call from the application. The secure services component executes within the UMHPIC.

[0045] At 405, the U-PAL may perform operations to cloak data in call parameters before they are sent to the host platform. For example, the U-PAL may encrypt data that will be written to persistent storage by the host platform.

[0046] At 406, a trusted thunking component of a U-PAL, such as the trusted thunking component 126 of the U-PAL 120, marshals the parameters of the call. For some parameters, such as for scalar values, the marshaling operation may be a null operation. For other parameters, the marshaling operation may be more complex. For example, if the call is one to write a buffer to disk, the trusted thunking component may allocate a temporary buffer in unprotected memory outside of the protected memory area, copy the contents from the buffer in the protected memory area to the temporary buffer in unprotected memory, and update the buffer pointer in the call parameters to point to the temporary buffer. In another example, a parameter may include a pointer-rich data structure, and the trusted thunking component may serialize it into a temporary buffer.

[0047] At 408, the trusted thunking component invokes an exit gate of the UMHPIC provided by the security-enabled processor to transfer execution to code of an untrusted thunking component, such as the untrusted thunking component 128 of the U-PAL 120, executing outside the UMHPIC.

[0048] At 410, the untrusted thunking component unmarshals the call parameters. For some parameters, such as for scalar values, the unmarshaling operation may be a null operation. For other parameters, the unmarshaling operation may be more complex. For example, the unmarshaling operation may deserialize a pointer-rich data structure. Various embodiments may include optimizations and/or extra coordination between an untrusted thunking component and a trusted thunking component to improve performance. Various optimizations for marshaling and unmarshaling may be used.

[0049] At 412, the untrusted thunking component issues the call to a host platform, such as the host platform 130. At 414, the host platform fulfills the system service requested.

[0050] At 416, when execution returns from the host platform to the untrusted thunking component, the untrusted component marshals the results of the call. For some results, such as for scalar values, the marshaling operation may be a null operation. For other results, the marshaling operation may be more complex. For example, the marshaling operation may serialize a pointer-rich data structure.

[0051] At 418, the untrusted thunking component invokes an entry gate of the UMHPIC provided by the security-enabled processor to transfer execution to code of the trusted thunking component executing inside the UMHPIC.

[0052] At 420, the trusted thunking component unmarshals the results of the call. For some results, such as for scalar values, the unmarshaling operation may be a null operation. For other results, the unmarshaling operation may be more complex. For example, if the call is one to read a buffer from disk, the trusted thunking component may copy the data from a temporary buffer in unprotected memory to a buffer in the protected memory area, and update the buffer pointer in the result parameters to point to the final buffer. In another example, a parameter may include a pointer-rich data structure, and the trusted thunking component may deserialize it.

[0053] At 421, the secure services component may decrypt data read from storage provided by the host platform and may also verify digital signatures or message authentication codes to verify the integrity of data. It may also perform verifications designed to protect against replay attacks. These steps may also be applied, for example, to virtual memory pages as they are read from page files on a hard disk. As noted elsewhere within this Detailed Description, the secure services component protects data written to the host storage system by the application executing in the UMHPIC. This is because the host platform and its storage services, including any file system provided by the host platform, are untrusted by applications executing in the UMHPIC. But the application itself may be unmodified to execute in the UMHPIC, and may therefore be configured to write data to a file system without protecting it. Thus, data read from the file system is decrypted and/or verified by the secure services component because the application is not configured to receive protected data, and may be unequipped to decrypt and verify it. In embodiments, the decryption step 421 may be combined with the unmarshaling step 420 as an optimization.

[0054] At 422, the secure services component verifies the returned results against bookkeeping data structures to verify that the results are consistent and trustworthy. In embodiments, the call might have been a request for a system resource or service, and the secure services component verifies that the result returned in response to the system request is valid for the requested resource or service. The system resource may be, in various embodiments, an allocated memory resource, a networking resource, an input/output resource, or other system resource.

[0055] The secure services component may utilize a bookkeeping data structure to verify output of any untrusted services from the host platform. For example, the bookkeeping data structure may enable the secure services component to validate that interface handles and memory allocations from the host platform are proper. In embodiments where the result is a resource or interface handle, the secure services component verifies that the returned resource or interface handle is not a duplicate of a previously provided resource handle in order to avoid handle replay attacks. In embodiments where the result is an indication of memory allocated to the application, the secure services component verifies that the allocated memory is not previously allocated to the application (which may cause the application to overwrite its own data and behave in unexpected ways), is accessible to the application, and does not lie within a reserved or invalid address range. In common embodiments, the bookkeeping data structures include structures for storing data related to one or more of memory allocations, resource handles, and thread identifiers.

[0056] At 424, if verification against bookkeeping data structures of the results of the call determines that the results are valid and therefore may be trusted, then execution proceeds.

[0057] At 426, if the result of the verification at 422 includes that one or more of the results are invalid, then the secure services component returns a failure result to the application. In some embodiments, the failure result is an error code relevant to that service request; for example, a well-known "disk read error" result on a disk read service request. In some embodiments, the failure may be delivered to the application as a catastrophic failure that causes the application to terminate immediately.

[0058] At 428, if the result of the verification at 422 includes that the results are valid, the secure services component updates the bookkeeping data structures as appropriate to the type of service request. For example, if the service request was to allocate a new region of memory, the secure services component may update a table of memory

allocations; the secure services component may also issues instructions to the security-enabled processor to add the new region of memory to the protected memory area of the UMPHIC.

[0059] At 430, the secure services component provides the successful results of the call to the application, which continues its execution.

[0060] FIG. 5 is a flow diagram showing an example process 500 for handling requests for execution and/or notifications of asynchronous events from a host platform, such as the host platform 130. At 502, an untrusted thunking component, such as the untrusted thunking component 128, receives an execution request or asynchronous notification from a host platform, such as the host OS 114 or host PAL 118. Various well-defined parameters accompany the requests and asynchronous notifications.

[0061] At 504, the untrusted thunking component marshals the parameters that accompany the requests and notifications. That is, the untrusted thunking component transforms the parameters to make them suitable for passing into an UMHPIC, such as the UMHPIC 104, via an entry gate to a process executing inside of the UMHPIC.

[0062] At 506, the untrusted thunking component invokes an entry gate provided by the security-enabled processor to pass the marshaled parameters associated with the requests and the notifications to the UMHPIC. And at 508, a trusted thunking component executing inside the UMHPIC, such as the trusted thunking component 126, receives the marshaled parameters via the entry gate.

[0063] At 510, the trusted thunking component unmarshals the parameters received from the untrusted thunking layer. Unmarshaling the parameters involves transforming the marshaled parameters to re-create the execution request or the asynchronous notification. At 512, the trusted thunking component passes the unmarshaled parameters to a secure services component executing inside the UMHPIC, such as the secure services component 122.

[0064] At 514, the secure services component performs security-relevant actions. These actions may include verifying that the execution request or asynchronous notification provided by the host platform is valid.

[0065] At 516, the secure services component passes the execution request or asynchronous notification to the application and/or a library OS executing in the UMHPIC.

[0066] FIGS. 3-5 depict flow graphs that show example processes in accordance with various embodiments. The operations of these processes are illustrated in individual

blocks and summarized with reference to those blocks. These processes are illustrated as logical flow graphs, each operation of which may represent a set of operations that can be implemented in hardware, software, or a combination thereof. In the context of software, the operations represent computer-executable instructions stored on one or more computer storage media that, when executed by one or more processors, enable the one or more processors to perform the recited operations. Generally, computer-executable instructions include routines, programs, objects, modules, components, data structures, and the like that perform particular functions or implement particular abstract data types. The order in which the operations are described is not intended to be construed as a limitation, and any number of the described operations can be combined in any order, separated into sub-operations, and/or performed in parallel to implement the process. Processes according to various embodiments of the present disclosure may include only some or all of the operations depicted in the logical flow graph.

Computer-Readable Media

[0067] Depending on the configuration and type of computing device used, memory 204 of the computing system 200 in FIG. 2 may include volatile memory (such as random access memory (RAM)) and/or non-volatile memory (such as read-only memory (ROM), flash memory, etc.). Memory 204 may also include additional removable storage and/or non-removable storage including, but not limited to, flash memory, magnetic storage, optical storage, and/or tape storage that may provide non-volatile storage of computer-readable instructions, data structures, program modules, and other data for computing system 200.

[0068] Memory 204 is an example of computer-readable media. Computer-readable media includes at least two types of computer-readable media, namely computer storage media and communications media.

[0069] Computer storage media includes volatile and non-volatile, removable and non-removable media implemented in any process or technology for storage of information such as computer-readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, phase change memory (PRAM), static random-access memory (SRAM), dynamic random-access memory (DRAM), other types of random-access memory (RAM), read-only memory (ROM), electrically erasable programmable read-only memory (EEPROM), flash memory or other memory technology, compact disk read-only memory (CD-ROM), digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage

or other magnetic storage devices, or any other non-transmission medium that can be used to store information for access by a computing device.

[0070] In contrast, communication media may embody computer-readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave, or other transmission mechanism. As defined herein, computer storage media does not include communication media.

Conclusion

[0071] Although the disclosure uses language that is specific to structural features and/or methodological acts, the invention is not limited to the specific features or acts described. Rather, the specific features and acts are disclosed as illustrative forms of implementing the invention.

What is claimed is:

1. A method comprising:

executing an application in a user-mode hardware-protected isolation container (UMHPIC), the UMHPIC protected by a security-enabled processor that provides at least an exit gate for passing execution out of the UMHPIC and an entry gate for returning execution into the UMHPIC;

facilitating, by an UMHPIC-aware platform adaptation layer executing in the UMHPIC, a system service request interaction between the application and one or more components of a host platform executing outside of the UMHPIC; and

performing, by a secure services component of the UMHPIC-aware platform adaptation layer, a security-relevant action as part of the facilitating.

2. The method of claim 1, wherein the security-relevant action includes encrypting data that is from the application and that is associated with the system service request interaction.

3. The method of claim 1, wherein the security-relevant action includes maintaining a bookkeeping data structure to store information regarding the system service request interaction.

4. A system, comprising:

one or more processors including a security-enabled processor configured to provide at least an entry gate and an exit gate for passing execution to and from, respectively, a user-mode hardware-protected isolation container (UMHPIC);

memory including the UMHPIC; and

an UMHPIC-aware platform adaptation layer executable by the security-enabled processor, at least partially inside the UMHPIC, to facilitate a system service request interaction between an application executing in the UMHPIC and one or more components of a host platform executing outside of the UMHPIC, the UMHPIC-aware platform adaptation layer including:

a secure services component executable by the security-enabled processor inside the UMHPIC to perform a security-relevant action as part of the facilitation of the system service request interaction.

5. The system of claim 4, wherein the system service request interaction is a request for allocation of resources, and wherein the security-relevant action includes verification of the correctness of a resource allocation result that is provided by the host platform as part of the system service request interaction.

6. The system of claim 4, wherein the resource allocation result includes an indication of memory resources, and wherein the verification of the correctness of the resource allocation result includes a check for overlapping memory allocations.

7. The system of claim 4, wherein the security-relevant action includes performance of thread scheduling services.

8. Computer-readable storage media comprising a plurality of programming instructions executable by one or more processors of a computing device to cause the computing device to:

facilitate a system resource request interaction between an application that executes within a user-mode hardware-protected isolation container (UMHPIC) and a host platform that executes outside of the UMHPIC, the UMHPIC provided by a security-enabled processor that is configured to provide at least an exit gate for passing execution outside of the UMHPIC and an entry gate for passing execution into the UMHPIC;

perform, on behalf of the application as part of the facilitation of the system resource request interaction, a security-relevant action.

9. The computer-readable storage media of claim 8, wherein the system service request interaction includes a request for allocation of resources, and wherein the security-relevant action includes verification of correctness of a resource allocation result that is provided by the host platform as part of the system service request interaction.

10. The computer-readable storage media of claim 9, wherein the resource allocation result includes a device handle, and wherein the verification of the correctness of the resource allocation result includes determining whether the device handle is a duplicate device handle.

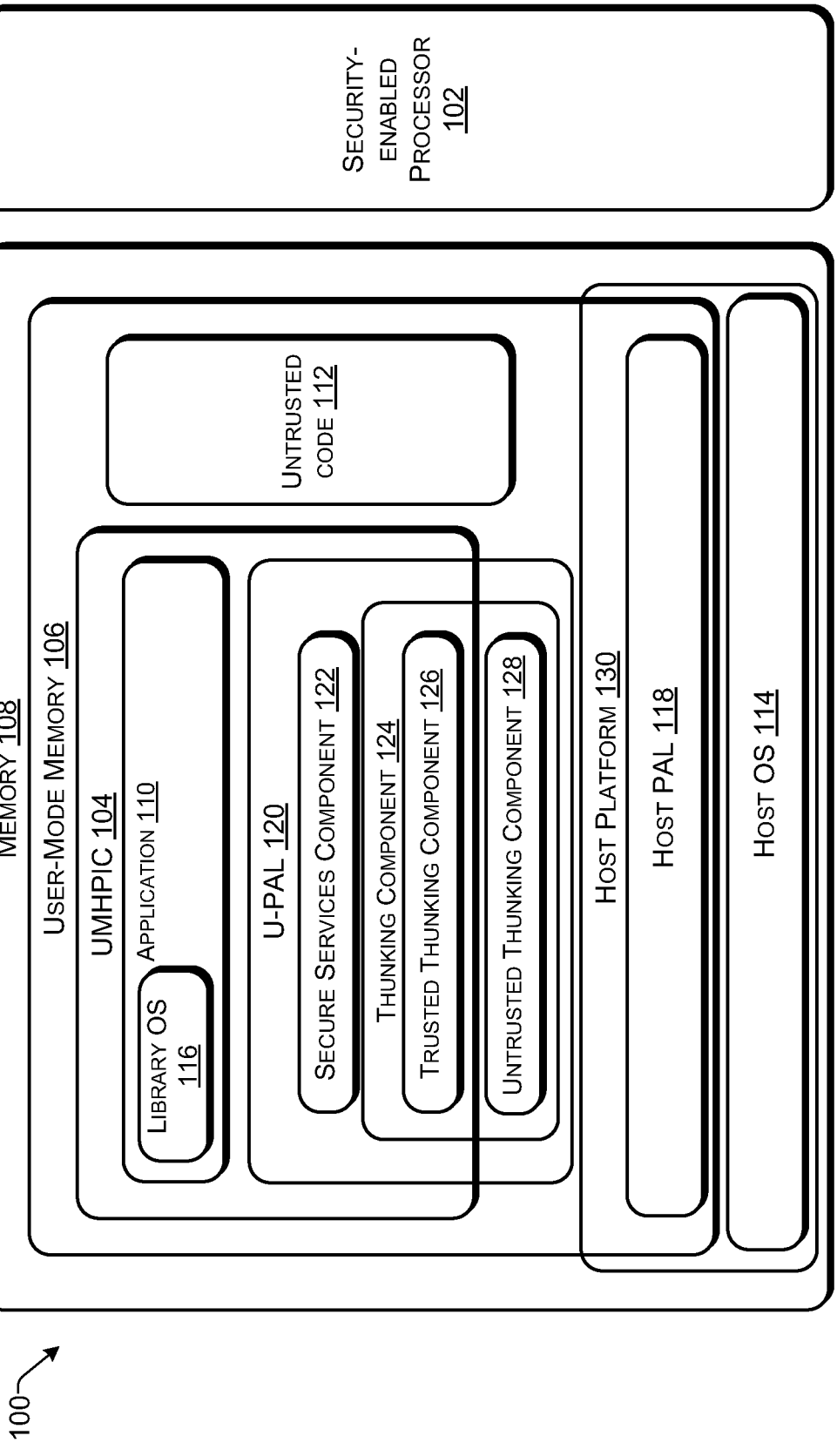
100
MEMORY 108
USER-MODE MEMORY 106
UMHPIC 104
APPLICATION 110
LIBRARY OS 116
U-PAL 120
SECURE SERVICES COMPONENT 122
THUNKING COMPONENT 124
TRUSTED THUNKING COMPONENT 126
UNTRUSTED THUNKING COMPONENT 128
UNTRUSTED CODE 112
HOST PLATFORM 130
HOST PAL 118
HOST OS 114
SECURITY-ENABLED PROCESSOR 102

FIG. 1

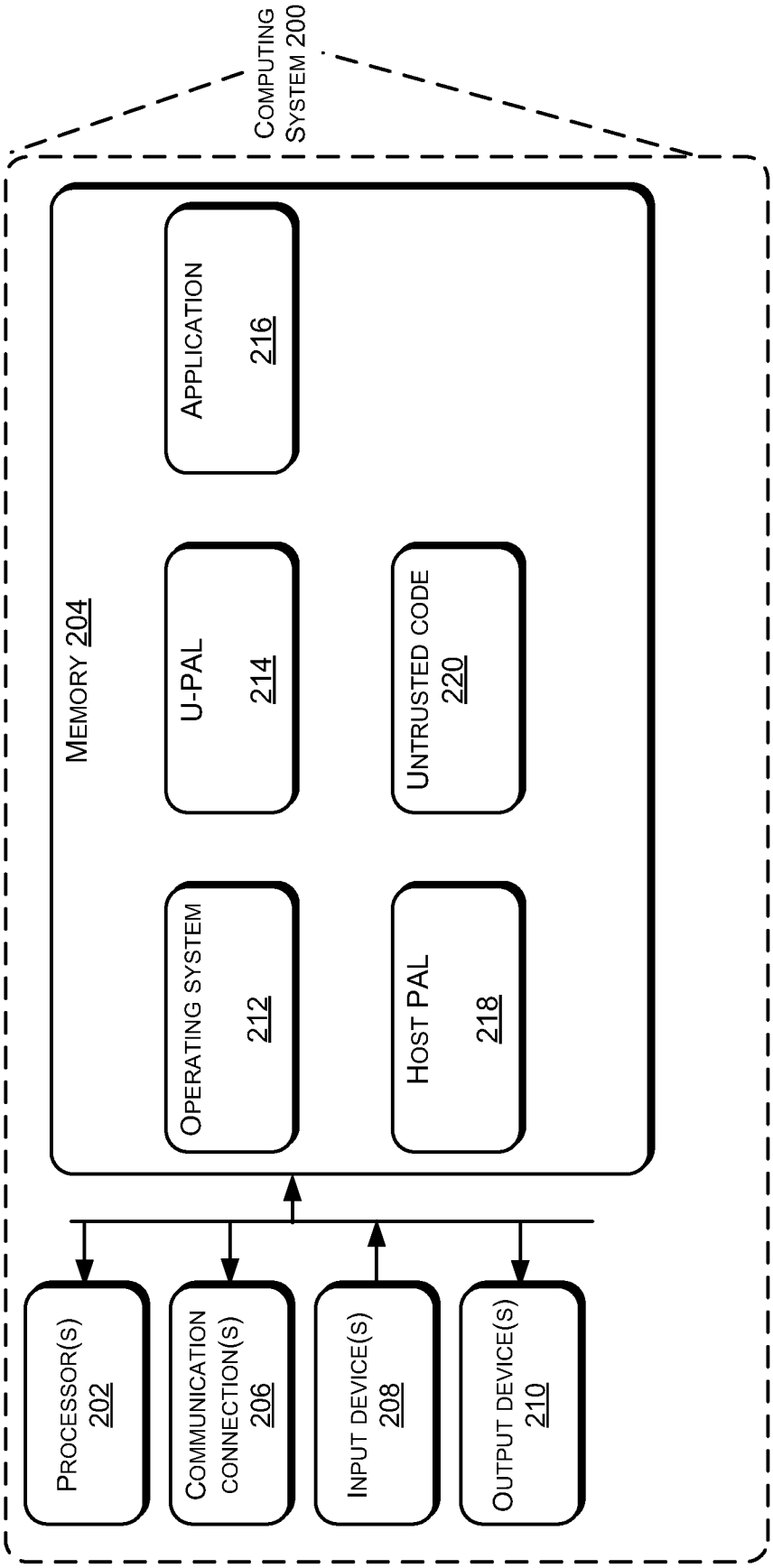
COMPUTING SYSTEM 200
MEMORY 204
APPLICATION 216
U-PAL 214
UNTRUSTED CODE 220
OPERATING SYSTEM 212
HOST PAL 218
PROCESSOR(S) 202
COMMUNICATION CONNECTION(S) 206
INPUT DEVICE(S) 208
OUTPUT DEVICE(S) 210

FIG. 2

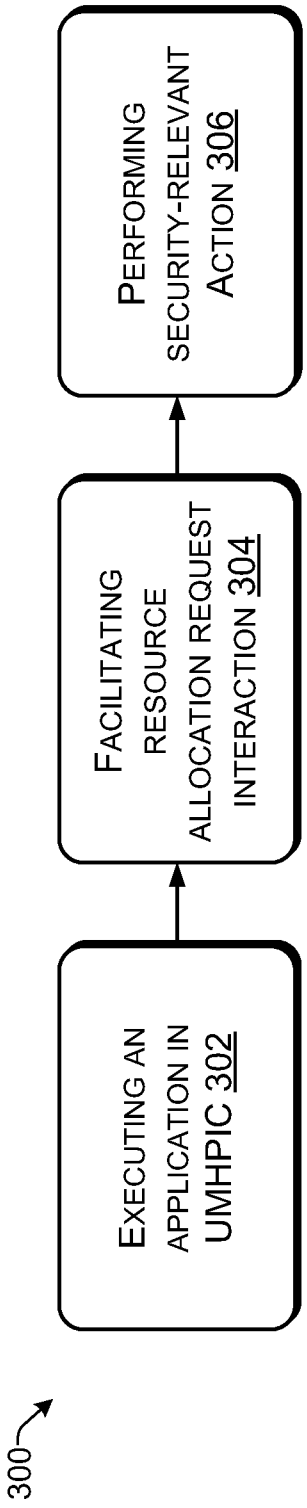
300
EXECUTING AN APPLICATION IN UMHPIC 302
FACILITATING RESOURCE ALLOCATION REQUEST INTERACTION 304
PERFORMING SECURITY-RELEVANT ACTION 306

FIG. 3

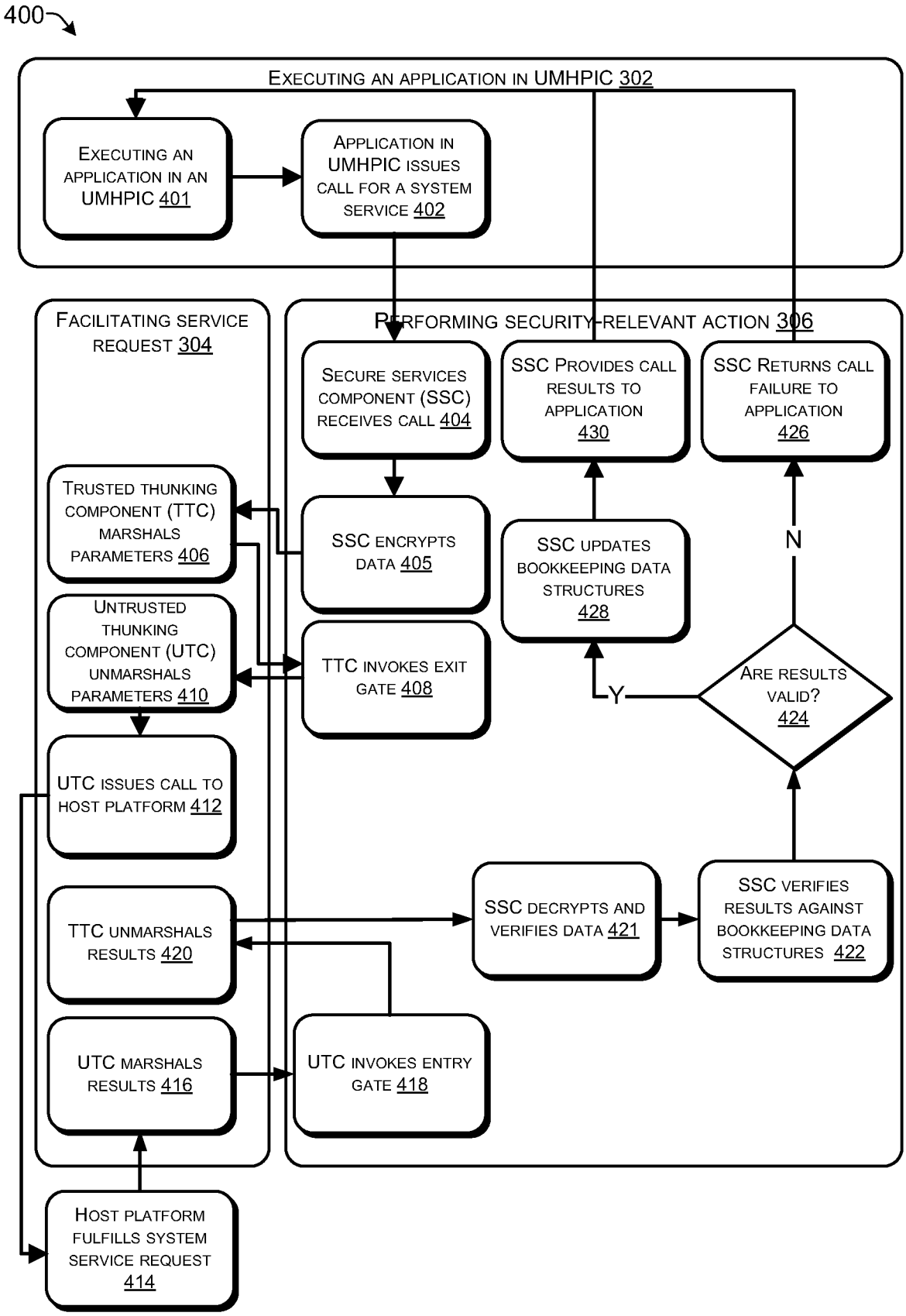
400
EXECUTING AN APPLICATION IN UMHPIC 302
EXECUTING AN APPLICATION IN AN UMHPIC 401
APPLICATION IN UMHPIC ISSUES CALL FOR A SYSTEM SERVICE 402
FACILITATING SERVICE REQUEST 304
PERFORMING SECURITY-RELEVANT ACTION 306
SECURE SERVICES COMPONENT (SSC) RECEIVES CALL 404
SSC PROVIDES CALL RESULTS TO APPLICATION 430
SSC RETURNS CALL FAILURE TO APPLICATION 426
TRUSTED THUNKING COMPONENT (TTC) MARSHALS PARAMETERS 406
SSC ENCRYPTS DATA 405
SSC UPDATES BOOKKEEPING DATA STRUCTURES 428
N
UNTRUSTED THUNKING COMPONENT (UTC) UNMARSHALS PARAMETERS 410
TTC INVOKES EXIT GATE 408
Y
ARE RESULTS VALID? 424
UTC ISSUES CALL TO HOST PLATFORM 412
TTC UNMARSHALS RESULTS 420
SSC DECRYPTS AND VERIFIES DATA 421
SSC VERIFIES RESULTS AGAINST BOOKKEEPING DATA STRUCTURES 422
UTC MARSHALS RESULTS 416
UTC INVOKES ENTRY GATE 418
HOST PLATFORM FULFILLS SYSTEM SERVICE REQUEST 414

FIG. 4

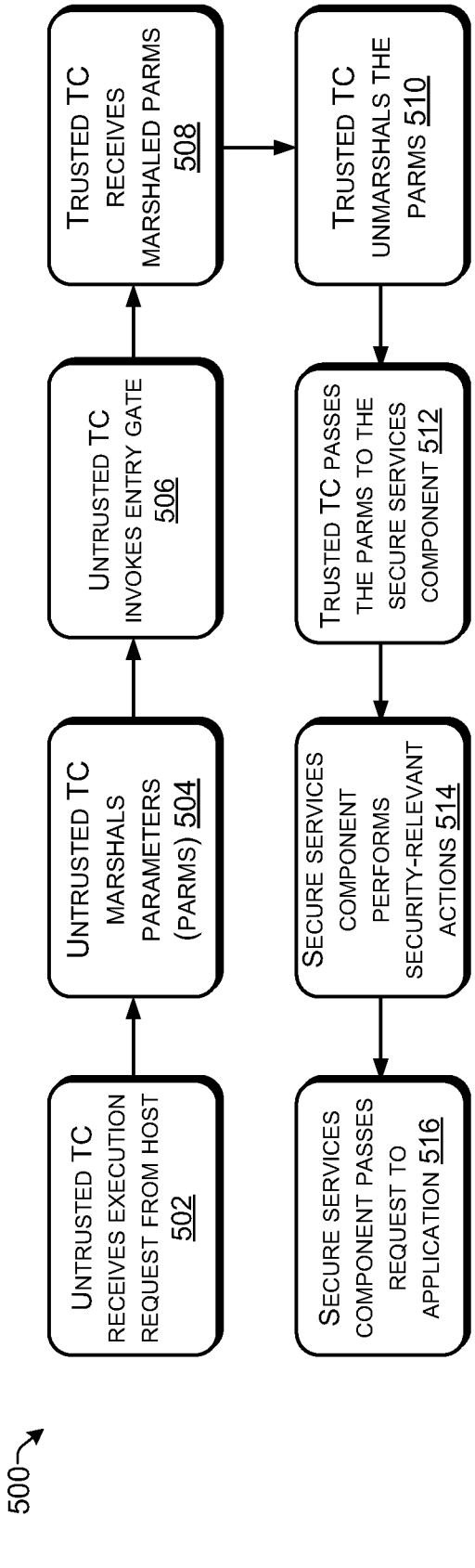
500
UNTRUSTED TC RECEIVES EXECUTION REQUEST FROM HOST 502
UNTRUSTED TC MARSHALS PARAMETERS (PARMS) 504
UNTRUSTED TC INVOKES ENTRY GATE 506
TRUSTED TC RECEIVES MARSHALED PARMS 508
TRUSTED TC UNMARSHALS THE PARMS 510
TRUSTED TC PASSES THE PARMS TO THE SECURE SERVICES COMPONENT 512
SECURE SERVICES COMPONENT PERFORMS SECURITY-RELEVANT ACTIONS 514
SECURE SERVICES COMPONENT PASSES REQUEST TO APPLICATION 516

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2012/067660

A. CLASSIFICATION OF SUBJECT MATTER

G06F 21/60(2013.01)i, G06F 15/16(2006.01)1, G06F 9/44(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 11/00, G06F 12/00, G06F 21/00, G06F 9/00

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: protected isolation container, secure, application, host and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2005-0033980 Al (BRYAN MARK WILLMAN et al.) 10 February 2005 See paragraphs 45-112 ; and figures 3-4 .	1-10
A	US 2005-0060722 Al (D0NN R0CHETTE et al.) 17 March 2005 See paragraphs 92-95 , 107 , 116 ; and figures 2 , 6 , 10 .	1-10
A	US 2007-0174910 Al (FREDERICK JOHN ZACHMAN et al.) 26 July 2007 See paragraphs 69-72 ; figure 2 ; and claim 1.	1-10
A	US 2011-0202739 Al (RICHARD ROY GRISENTHWAITE) 18 August 2011 See paragraphs 50-53 ; and figure 3 .	1-10
A	US 2010-0306848 Al (WOLFGANG GELLERICH) 02 December 2010 See paragraphs 56-74 ; and figure 4 .	1-10

☐ 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
20 MARCH 2013 (20.03.2013)	**20 MARCH 2013 (20.03.2013)**
Name and mailing address of the ISA/KR Korean Intellectual Property Office 189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-70 1, Republic of Korea Facsimile No. 82-42-472-7140	Authorized officer BYUN, Sung Cheal Telephone No. 82-42-481-8262

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

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2012/067660

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005-0033980 A1	10.02.2005	AT 530991 T	15.11.2011
		AT 539411 T	15.01.2012
		AU 2004-203528 A1	24.02.2005
		AU 2004-203528 B2	28.01.2010
		AU 2004-203528 B9	04.03.2010
		BR PI0403261 A	31.05.2005
		CA 2473487 A1	07.02.2005
		CN 1581073 A	16.02.2005
		CN 1581073 B	21.03.2012
		CO 5600214 A1	31.01.2006
		EP 1505475 A2	09.02.2005
		EP 1505475 A3	09.03.2005
		EP 1505475 B1	28.12.2011
		EP 1916614 A1	30.04.2008
		EP 1916614 B1	26.10.2011
		ES 2372989 T3	30.01.2012
		ES 2376433 T3	13.03.2012
		IL 162997 A	17.05.2010
		JP 04726449 B2	20.07.2011
		JP 2005-056429 A	03.03.2005
		KR 10-0949022 B1	23.03.2010
		KR 10-2005-0016202 A	21.02.2005
		MX PA04007483 A	08.06.2005
		MY 140973 A	12.02.2010
		NO 20043285 A	08.02.2005
		NZ 534064 A	26.08.2005
		RU 2004124072 A	27.01.2006
		RU 2390836 C2	27.05.2010
		SG 108962 A1	28.02.2005
		US 7530103 B2	05.05.2009
		ZA 200405646 A	24.03.2005
US 2005-0060722 A1	17.03.2005	CA 2481613 A1	15.03.2005
		CA 2482248 A1	22.03.2005
		CA 2544943 A1	02.11.2006
		CA 2545047 A1	06.11.2006
		CA 2546720 A1	13.11.2006
		EP 1515229 A2	16.03.2005
		EP 1515229 A3	19.10.2005
		EP 1720098 A1	08.11.2006
		EP 1722312 A2	15.11.2006
		EP 1722312 A3	05.10.2011
		US 2005-0066303 A1	24.03.2005
		US 2006-0184931 A1	17.08.2006
		US 2006-0253858 A1	09.11.2006
		US 2006-0265761 A1	23.11.2006
		US 2008-0222160 A1	11.09.2008
		US 7519814 B2	14.04.2009
		US 7757291 B2	13.07.2010

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2012/067660

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
		US 7774762 B2	10.08.2010
		US 7784058 B2	24.08.2010
		WO 2009-114758 A1	17.09.2009
US 2007-0174910 A1	26.07.2007	None	
US 2011-0202739 A1	18.08.2011	CN 102763092 A	31.10.2012
		GB 1210574 DO	01.08.2012
		GB 2488938 A	12.09.2012
		US 8301856 B2	30.10.2012
		WO 2011-101609 A1	25.08.2011
US 2010-0306848 A1	02.12.2010	CN 101632083 A	20.01.2010
		EP 2143031 A1	13.01.2010
		JP 05079084 B2	07.09.2012
		JP 2010-527060 A	05.08.2010
		US 8239959 B2	07.08.2012
		WO 2008-138653 A1	20.11.2008