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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 December 2017 (14.12.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/213643 A1

(51) International Patent Classification:
G06F 9/54 (2006.01) *G06F 9/44* (2006.01)
G06F 9/445 (2006.01)

(21) International Application Number:
PCT/US2016/036366

(22) International Filing Date:
08 June 2016 (08.06.2016)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(72) Inventors: **EDWARDS, Nigel**; Mailstop T26, Longdown Avenue, Stoke Gifford, Bristol Somerset BS34 8QZ (GB). **DALTON, Chris I**; Mail Stop T21, Longdown Avenue, Bristol Filton B3, Bristol Somerset BS34 8QZ (GB).

(74) Agent: **ZHANG, Shu**; 3404 E. Harmony Road, Mail Stop 79, Fort Collins, Colorado 80528 (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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, 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, SA, 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, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:
— *as to the identity of the inventor (Rule 4.17(i))*

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

(54) Title: EXECUTING SERVICES IN CONTAINERS

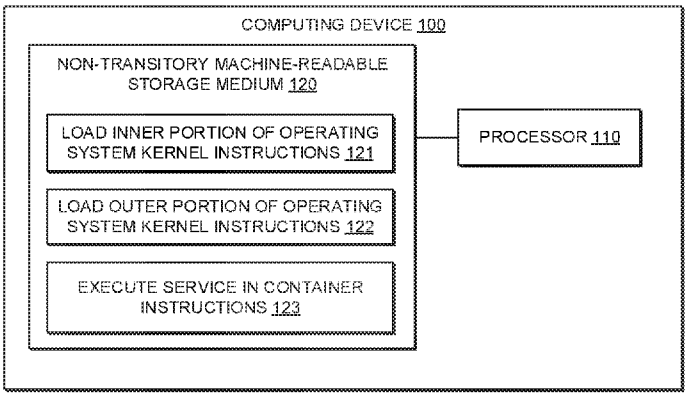

FIG. 1

(57) Abstract: Example embodiments relate to executing services in containers. The examples disclosed herein include a computing device comprising instructions to load an inner portion of an operating system kernel in an inner region of a kernel space and an outer portion of the operating system kernel in an outer region of the kernel space. The example computing device may execute a service in a container in a user space. The container may be communicatively coupled with the outer region of the operating system kernel but divided from the inner portion of the operating system kernel.

WO 2017/213643 A1

EXECUTING SERVICES IN CONTAINERS

BACKGROUND

[0001] Containers are paradigms for developing and deploying computer services. Developers and operators value containers because dependencies for running a service are encapsulated inside a single entity.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] The following detailed description references the drawings, wherein:

[0003] FIG. 1 is a block diagram of an example computing device for executing services in containers;

[0004] FIG. 2 is a block diagram of a non-transitory machine-readable storage medium of an example computing device for executing services in containers;

[0005] FIG. 3 is a block diagram of an example system for executing services in containers;

[0006] FIG. 4 is a block diagram of an example system for executing services in containers illustrating multiple containers in the system;

[0007] FIG. 5 is a flowchart of an example method for executing services in containers.

DETAILED DESCRIPTION

[0008] Containers are paradigms for developing and deploying computer services. Developers and operators value containers because dependencies for running a service are encapsulated inside a single entity. This makes it easy to transport a service from a development environment to an operating environment. Furthermore, multiple containers may share a physical machine without conflict because differences in libraries and configuration may be encapsulated within each container. Containers may efficiently utilize resources and may be easier to manage than virtual machines because there may be a single operating system per physical machine. Thus, a container-based system may involve fewer operating systems for an administrator to manage and fewer redundant processes since there is only a single kernel per machine and a single set of operating system services (daemons).

[0009] Because container-based systems rely on multiple mechanisms spread throughout a kernel of an operating system including a single kernel and multiple processes executing in multiple containers running on the kernel, it may be challenging to verify and maintain the integrity of these mechanisms. For example, a single bug in just one of the mechanisms may compromise some or all of the containers since multiple containers may access the same file systems, memory, storage, and/or other mechanisms. Memory corruption vulnerabilities are typically targeted by attackers to take over a kernel. For example, an attacker may overwrite a significant data structure, typically a function pointer, to exploit a memory corruption bug. The function pointer may then be updated to point to attacker-provided code. When the kernel follows the function pointer, it may be tricked into executing the attacker-provided code, which may provide the attacker unwanted access or control over the kernel. The attacker may then be able to monitor and interfere with processes running on the system and memory on the system, which may compromise other containers running on the system.

[0010] A current solution for protecting against such kernel compromises is to run containers in separate virtual machines. This provides protection because each virtual machine has its own kernel, so a kernel on one virtual machine can be isolated from kernels of other virtual machines. However, each virtual machine may have its own kernel and its

own operating system services. Such an arrangement can create significant overhead because it could require additional processor, memory, and storage resources. Furthermore, system administrators may have to manage multiple operating system instances and various resources in addition to managing the container, the virtual machine, and the interaction therebetween. This defeats the initial rationale behind using containers.

[0011] Examples disclosed herein address these technical challenges by providing an efficient and secure architecture for executing services in containers. An example computing device may load an inner portion of an operating system kernel in an inner region of a kernel space and an outer portion of the operating system kernel in an outer region of the kernel space. A service may be executed in a container in a user space, where the container is communicatively coupled with the outer region of the operating system kernel and is divided from the inner region of the operating system kernel. In this manner, examples herein provide protection against kernel flaws and prevents containers from breaking out of isolation mechanisms and compromising the system.

[0012] Referring now to the drawings, FIG. 1 depicts an example computing device 100 for executing services in containers. Computing device 100 may be, for example, a cloud server, a local area network server, a web server, a mainframe, a mobile computing device, a notebook or desktop computer, a smart TV, a point-of-sale device, a wearable device, any other suitable electronic device, or a combination of devices, such as ones connected by a cloud or internet network, that perform the functions described herein. In the example shown in FIG. 1, computing device 100 includes a processor 110 and a non-transitory machine-readable storage medium 120 encoded with instructions to execute services in containers.

[0013] Processor 110 may be one or more central processing units (CPUs), semiconductor-based microprocessors, and/or other hardware devices suitable for retrieval and execution of instructions stored in machine-readable storage medium 120. Processor 110 may fetch, decode, and execute instructions 121, 122, 123, and/or other instructions to implement the procedures described herein. As an alternative or in addition to retrieving and executing instructions, processor 110 may include one or more electronic circuits that

include electronic components for performing the functionality of one or more of instructions 121, 122, and 123.

[0014] In an example, the program instructions 121, 122, 123, and/or other instructions can be part of an installation package that can be executed by processor 110 to implement the functionality described herein. In such a case, memory 120 may be a portable medium such as a CD, DVD, or flash drive or a memory maintained by a computing device from which the installation package can be downloaded and installed. In another example, the program instructions may be part of an application or applications already installed on computing device 100.

[0015] Machine-readable storage medium 120 may be any electronic, magnetic, optical, or other physical storage device that contains or stores executable data accessible to computing device 100. Thus, machine-readable storage medium 120 may be, for example, a Random Access Memory (RAM), an Electrically Erasable Programmable Read-Only Memory (EEPROM), a storage device, an optical disc, and the like. Storage medium 120 may be a non-transitory storage medium, where the term "non-transitory" does not encompass transitory propagating signals. Storage medium 120 may be located in computing device 100 and/or in another device in communication with computing device 100. As described in detail below, machine-readable storage medium 120 may be encoded with load inner portion of operating system kernel instructions 121, load outer portion of operating system kernel instructions 122, and execute service in container instructions 123.

[0016] Instructions 121, responsive to being executed by processor 110, may load an inner portion of an operating system kernel in an inner region of a kernel space. Instructions 122, responsive to being executed by processor 110, may load an outer portion of the operating system kernel in an outer region of the kernel space. An operating system kernel may be a computer program that constitutes the central core of the operating system of computing device 100. The operating system kernel may control processes, programs, and functions being executed by computing device 100. The operating system kernel may manage the processing of instructions, memory operations, as well as communications with other components.

[0017] A kernel space may be a part of a virtual memory of computing device 100. The virtual memory may map virtual addresses of a program into physical addresses in computer memory of computing device 100, such as storage medium 120 or other memory device. Processor 110 of computing device 100 may segregate the virtual memory of the computing device into the kernel space and a user space. For example, the kernel space may be reserved for running the operating system kernel, kernel extensions, and device drivers. The user space, in contrast, may be the memory area where applications and services are executed.

[0018] Furthermore, the kernel space may be divided into an inner region and an outer region. The inner portion of the operating system kernel may be loaded in the inner region, and the outer portion of the operating system kernel may be loaded in the outer region. The inner portion may, in some examples, have direct unfettered access to the hardware of computing device 100. In contrast, a virtual memory interface may be presented to the outer portion, which may not have direct access to privileged portions of the hardware, such as a memory management unit. The security goals of the kernel division are integrity guarantees for kernel code and critical data along with kernel control flow integrity, and information flow control enforcement across processes within the operating system kernel.

[0019] For example, the inner portion of the operating system kernel may include a memory management unit, a process management unit, and architecture specific code. The memory management unit may be a hardware unit that manages virtual memory and performs translation of virtual memory addresses to physical memory addresses. The process management unit may manage the data structures for processes running on the operating system. The architecture specific code may be custom instructions that modify an existing operating system kernel to implement an example kernel architecture described herein. The inner kernel may manage communication with the outer portion of the kernel by providing a restricted API which may be accessible to any outer kernel component.

[0020] In some examples, the outer portion of the operating system kernel may include all other components of the operating system kernel not included in the inner portion. For example, the outer portion may include a file systems unit and a device driver unit. The file systems unit may provide an abstraction of files to user space programs. For example, the

file systems unit may communicate with other outer kernel components including the device driver unit. The device driver unit may provide interfaces for hardware devices, which may enable the operating system to access hardware functions.

[0021] In some examples, the kernel space may be divided into the inner region, which loads the inner portion of the operating system kernel, and the outer region, which loads the outer portion of the operating system kernel, by nested page tables. The inner portion of the kernel may be mapped in an inner page table, which may have controlled access from the outer portion of the kernel and any processes running on the outer portion of the kernel. For example, the inner portion may be inaccessible, read-only, or a combination of both. The outer portion, on the other hand, may be mapped in an outer page table, which may map directly to physical memory, but the nested structure of the inner page table and the outer page table controls the access to the inner portion of the kernel. As a result, in some examples, an attempt to write the inner portion of the kernel by the outer portion of the kernel may cause a violation if the access is read-only or inaccessible. Furthermore, the mapping from the outer page table to physical memory may be controlled by the inner portion of the kernel through the inner page table. The mapping of the outer portion of the kernel and its processes' virtual memory to physical memory may thus be under the complete control of the inner portion of the kernel.

[0022] It should be noted that the inner portion of the operating system kernel and the outer portion of the operating system kernel may, in some examples, be loaded initially as a single kernel image. The processes of the kernel may then be dynamically transitioned into their respective portions. The entire kernel may share the same code base but attempts to access privileged functionality, such as those restricted to the inner portion of the kernel, from the outer portion of the kernel may cause a violation.

[0023] Continuing to refer to FIG. 1, instructions 123, responsive to being executed by processor 110, may execute a service in a container in a user space. The container may be an isolated user space instance, in which processes are executed. The user space may be the memory area of the operating system of computing device 100 where application processes are executed. The container may be communicatively coupled to the outer portion of the operating system kernel, and the container may be divided from the inner

portion of the operating kernel. In some examples, the container may include the outer portion of the kernel, and the combined structure is considered the container. The service being executed in the container may be any program, application, or process that can be virtualized in the container.

[0024] As described in further detail in relation to FIG. 2 below, a single operating system kernel may include an inner portion and a plurality of outer portions sharing the same base code. Each of the outer portions may be able to facilitate a container in which a service is executed. By loading a plurality of outer portions, computing device 100 running one operating system kernel may facilitate multiple container instances, where each container is isolated from each other.

[0025] FIG. 3 provides an illustrative example of the architecture described herein. FIG. 3 is a block diagram of a system 300 for executing services in containers. For example, system 300 may be implemented as computing device 100, but may also be a system of multiple devices connected through a network, such as the cloud. System 300 may include a kernel space 310 and a user space 320. An inner portion 330 of an operating system kernel may be loaded in an inner region 312 of the kernel space 310. An outer portion 335 of the operating system kernel may be loaded in an outer region 314 of kernel space 310. As described above, the inner region 312 may be divided from the outer region 314 by nested page tables. As an example, inner portion 330 of the operating system kernel may include a memory management unit 330A, a process management unit 330B, and architecture specific code 330C. Similarly, outer portion 335 may include a file systems unit 335A and a device driver unit 335B.

[0026] In the user space 320, a service 345C may be executed in a container 340C. As illustrated, container 340C may be communicatively coupled to outer portion 335 of the kernel. It may also be understood that the combination of service 345C and outer portion 335 as a whole may represent container 340C. Container 340C is divided from inner portion 330 of the operating system kernel.

[0027] Referring back now to FIG. 2, FIG. 2 depicts a non-transitory machine-readable storage medium 200 of an example computing device for executing services in containers. The example computing device may be computing device 100 or FIG. 1 or a second

computing device. The example device may be, for example, a cloud server, a local area network server, a web server, a mainframe, a mobile computing device, a notebook or desktop computer, a smart TV, a point-of-sale device, a wearable device, any other suitable electronic device, or a combination of devices, such as ones connected by a cloud or internet network, that perform the functions described herein. Non-transitory machine-readable storage medium 200 may be encoded with instructions to execute a service in a container.

[0028] The example computing device may also include a processor, which may be one or more central processing units (CPUs), semiconductor-based microprocessors, and/or other hardware devices suitable for retrieval and execution of instructions stored in machine-readable storage medium 200. The processor may fetch, decode, and execute instructions 210, 220, 230, 240, and/or other instructions to implement the procedures described herein.

[0029] Load inner portion of operating system kernel instructions 210 may be analogous to instructions 121 of computing device 100, and may load an inner portion of an operating system kernel in an inner region of a kernel space. Outer portion of operating system kernel instructions 220 may be analogous to instructions 122 of computing device 100. Furthermore, instructions 220 may include load outer portion instructions 221 and load second outer portion instructions 222. Instructions 221 and instructions 222 may respectively load an outer portion and a second outer portion of the operating system kernel in an outer region of the kernel space.

[0030] As mentioned above, multiple outer portions of the operating system kernel may be loaded with a single inner portion of the kernel. In the example of FIG. 2, the outer portion of the kernel and the second outer portion of the kernel may each have its own outer page tables. For example, the outer portion and the second outer portion may share memory, and may be marked copy on write. If one of the outer portions attempts to update a section of memory marked copy on write, the operating system kernel, via a memory management unit loaded in the inner portion of the kernel, may allocate a new piece of physical memory for that outer portion of the kernel so that it is writing to its own private copy and has no effect on the other outer portions of the kernel.

[0031] Continuing to refer to FIG. 2, container instructions 230 may be analogous to instructions 123 of computing device 100. Furthermore, instructions 230 may include execute service in container instructions 231 and execute second service in second container instructions 232. Instructions 231 may execute a service in a container in a user space, where the container may be communicatively coupled to the outer portion of the operating system kernel, and the container may be divided from the inner portion of the operating kernel. Instructions 232 may execute a second service in a second container in the user space, where the second container may be communicatively coupled to the second outer portion of the operating system kernel, and the second container may be divided from the inner portion of the operating kernel.

[0032] FIG. 3 provides an illustrative example of the architecture implemented by the example computing device of FIG. 2. For example, instructions 210 may load inner portion 330 of an operating system kernel in an inner region 312 of a kernel space 310. Instructions 221 may load an outer portion 335 of the operating system kernel in an outer region 314 of the kernel space 310. Instructions 231 may execute a service 345C in a container 340C communicatively coupled to the outer portion 335 of the operating system kernel. Furthermore, instructions 222 may load a second outer portion of the operating system kernel in the outer region 314 of the kernel space 310. The second outer portion of the operating system kernel may include the same physical memory as the outer portion 335 and may include the same units and processes. A second container, such as container 340B may allow execution of a second service 345B. Second container 340B may be communicatively coupled with the second outer portion of the operating system kernel and may be divided from the inner portion 330.

[0033] In some examples, container 340C and second container 340B may be isolated from each other. In the example illustrated in FIG. 3, system 300 may also include a third container 340A, which may also be isolated from the other containers. The containers may be isolated by the use of address spaces. For example, each container may comprise an isolated address space. For example, multiple containers may map to the same base code, but are marked copy on write. If a process of a container attempt to update a section of memory marked copy on write, the operating system kernel, via a memory management

unit loaded in the inner portion of the kernel, may allocate a new piece of physical memory for a corresponding outer portion of the kernel so that the process is writing to its own private copy and has no effect on the other containers.

[0034] Referring back to FIG. 2, attack instructions 240 may include detect attack instructions 241 and remedial action instructions 242. Detect attack instructions 241, responsive to being executed by a processor, may detect an attack on a container, a service, or an outer portion of an operating system kernel. An attack may be any attempt to destroy, expose, alter, disable, steal, compromise, or otherwise gain unauthorized access or use of a computing asset. Detect attack instructions 241 may, for example, detect attacks on the container architecture itself, on the processes running in the containers, or on the outer portion of the operating system kernel.

[0035] Remedial action instructions 242, responsive to being executed by a processor, may take remedial action on the attacked container, service, or outer portion of the operating system kernel. For example, the remedial action may only be taken on the unit of the system that is targeted and affected by the attack. A remedial action on the attack may include identifying malicious activity, logging information about this activity, attempting to stop it, reporting it, and/or action to respond to the attack. In such a manner, the second container, the second service, and the second outer portion of the operating system kernel are unaffected.

[0036] FIG. 4 provides an illustrative example of the isolation mechanisms of the architecture described herein and the operations of attack instructions 240 of storage medium 200. FIG. 4 depicts a system 400 for executing services in containers illustrating multiple containers in the system. For example, system 400 may be implemented as computing device 200, but may also be a system of multiple devices connected through a network, such as the cloud. System 400 may include a kernel space 410 and a user space 420. An inner portion 430 of an operating system kernel may be loaded in an inner region of the kernel space 410. A plurality of outer portions 435 of the operating system kernel may be loaded in an outer region of kernel space 410.

[0037] A plurality of containers 440 may facilitate the executing of a plurality of services. For example, a service may be executed in each container 440 in the user space 420. Each

container 440 may include an outer portion of the operating system kernel and a service in the user space. The container 440 may also include user code and data. Each container 440 may include the same base code, which may be marked copy on write. As a result, an attempt to change the kernel memory may result in a new copy of the physical memory being created to account for the change.

[0038] In some examples such as those described herein, any attempt 460 by a container 440 to access physical memory not in its page table or with the wrong permission or to inject unauthorized code from the user space 420 into the kernel space 410 may trap into the inner portion of the operating system kernel 430 because the user space 420 and outer region of the kernel space may have separate upper level page tables for each container 440. So the processor may enforce isolation between user space processes and data from separate containers as it is processed by the operating system kernel. Memory for outer regions of the kernel space may be immutable within a container, and only changeable within the inner region of the kernel space.

[0039] Furthermore, in some examples where the memory is marked copy on write, an update to a function pointer during an attack will be made within the outer portion 435 of the operating system kernel in which the attacker is running, but not to any other outer portions of the operating system kernel. Therefore the integrity of other outer portions may be unaffected. So far as is possible, any sensitive kernel data structures, such as those controlling process privilege, may be read-only in the outer portions of the operating system kernel.

[0040] The above means that an attacker may not be able to monitor or change the memory of any other process or container on the system that is using a different outer portion of the operating system kernel. The physical memory may be inaccessible to the outer portions of the operating system kernel because of nested page tables. Attempts to access this memory may cause a trap as described above, so an attack will be detected and can be remediated. Similarly any attempt to update memory which is read-only to the outer region of the kernel space to escalate privilege or similar will be detected via a trap and the attack can be remediated.

[0041] Referring now to FIG. 5, example method 500 illustrates executing services in containers. Although execution of method 500 is described below with reference to the examples illustrated in FIG. 2, 3, and 4, other suitable devices for execution of this method should be apparent, including the examples of FIG. 1. Method 500 may be implemented in the form of executable instructions stored on a machine-readable storage medium and/or in the form of electronic circuitry.

[0042] In an operation 510, an inner portion of an operating system kernel may be loaded in an inner region of a kernel space. For example, instructions 210 of the computing device of FIG. 2 may load an inner portion 330 of an operating system kernel in an inner region 312 of a kernel space 310 in the system 300 of FIG. 3. As described previously herein, the inner portion may, in some examples, have direct unfettered access to the hardware of computing device 100. In some examples, the inner portion of the operating system kernel may include a memory management unit, a process management unit, and architecture specific code.

[0043] In an operating 520, a plurality of outer portions of the operating system kernel may be loaded in an outer region of the kernel space. For example, instructions 221 of the computing device of FIG. 2 may load an outer portion 335 of an operating system kernel in an outer region 314 of the kernel space 310 in the system 300 of FIG. 3. Instructions 222 of the computing device of FIG. 2 may also load a second outer portion of an operating system kernel in the inner region 312 of the kernel space 310. Each outer portion of the operating system kernel may include all other components of the operating system kernel not included in the inner portion, and each outer portion may facilitate a separate container for executing services.

[0044] In an operation 530, a service may be executed in a container of a plurality containers in a user space. For example, instructions 231 of the computing device of FIG. 2 may execute a service 345C in a container 340C in a user space 320 in the system 300 of FIG. 3. The container may be communicatively coupled to the outer portion of the operating system kernel, and the container may be divided from the inner portion of the operating kernel. In some examples, the container may include the outer portion of the kernel. The service being executed in the container may be any program, application, or process that can be virtualized in the container.

[0045] In an operation 540, an attack on the container, the service, or the outer portion of the operating system kernel may be detected. For example, instructions 241 of the computing device of FIG. 2 may detect an attack on service 345C, container 340C, or outer portion 335 of the operating system kernel in the system 300 of FIG. 3. As explained previously, an attack may be any attempt to destroy, expose, alter, disable, steal, compromise, or otherwise gain unauthorized access or use of a computing asset. For example, an attack may be detected on the container architecture itself, on the processes running in the containers, or on the outer portion of the operating system kernel.

[0046] In an operation 550, take remedial action may be taken on the attacked container, service, or outer portion of the operating system kernel in response to detecting an attack in operation 540. For example, instructions 242 of the computing device of FIG. 2 may take remedial action on service 345C, container 340C, or outer portion 335 of the operating system kernel in the system 300 of FIG. 3. A remedial action on the attack may include identifying malicious activity, logging information about this activity, attempting to stop it, and/or reporting it. As illustrated in FIG. 4, an attack 460 on container 440, a service running in container 440, or outer portion 435 of the kernel associated with container 440 may be detected. An remedial action may be taken on the attacked component, which is isolated within container 440 and does not affect inner portion 430 of the kernel or other containers in the system.

[0047] The foregoing disclosure describes a number of example embodiments for executing services in containers. The disclosed examples may include systems, devices, computer-readable storage media, and methods for execution of services in containers. For purposes of explanation, certain examples are described with reference to the components illustrated in FIGS. 1-5. The functionality of the illustrated components may overlap, however, and may be present in a fewer or greater number of elements and components. All or part of the functionality of illustrated elements may co-exist or be distributed among several geographically dispersed locations. Moreover, the disclosed examples may be implemented in various environments and are not limited to the illustrated implementations.

[0048] Further, the sequence of operations described in connection with FIGS. 1-5 are examples and are not intended to be limiting. Additional or fewer operations or combinations

of operations may be used or may vary without departing from the scope of the disclosed examples. Furthermore, implementations consistent with the disclosed examples need not perform the sequence of operations in any particular order. Thus, the present disclosure merely sets forth possible examples of implementations, and many variations and modifications may be made to the described examples. All such modifications and variations are intended to be included within the scope of this disclosure and protected by the following claims.

CLAIMS

What is claimed is:

1. A computing device, comprising a processor and a non-transitory machine-readable storage medium encoded with instructions executable by the processor, the non-transitory storage medium comprising instructions to:

load an inner portion of an operating system kernel in an inner region of a kernel space;

load an outer portion of the operating system kernel in an outer region of the kernel space; and

execute a service in a container in a user space, wherein the container is communicatively coupled with the outer portion of the operating system kernel, and wherein the container is divided from the inner portion of the operating system kernel.

2. The computing device of claim 1, wherein the non-transitory storage medium further comprises instructions to:

load a second outer portion of the operating system kernel in the outer region of the kernel space; and

execute a second service in a second container in the user space, wherein the second container is communicatively coupled with the second outer portion of the operating system kernel, and wherein the second container is divided from the inner portion of the operating system kernel.

3. The computing device of claim 2, wherein the container is isolated from the second container.

4. The computing device of claim 1, wherein the inner portion of the operating system kernel comprises a memory management unit, a process management unit, and architecture specific code.

5. The computing device of claim 1, wherein the container and the second container each comprises an isolated address space.

6 The computing device of claim 1, wherein the outer portion of the operating system kernel and the second outer portion of the operating system kernel each comprises a file systems unit and a device driver unit.

7. The computing device of claim 1, wherein the outer region of the kernel space and the inner region of the kernel space are divided by nested page tables.

8. The computing device of claim 1, wherein the non-transitory storage medium further comprises instructions to: in response to detecting an attack on the container, the service, or the outer portion of the operating system kernel, take remedial action on the attacked container, service, or outer portion of the operating system kernel, wherein the second container, second service, and second outer portion of the operating system kernel are unaffected.

9. A system, comprising a processor to:

load an inner portion of an operating system kernel in an inner region of a kernel space;

load a set of outer portions of the operating system kernel in an outer region of the kernel space;

execute a service in a container of a set of containers in a user space, wherein each container is communicatively coupled with an outer portion of the set of outer portions of the operating system kernel, and wherein each container is divided from the inner portion of the operating system kernel and each container is isolated from all other containers in the set of containers.

10. The system of claim 9, wherein the outer region of the kernel space and the inner region of the kernel space are divided by nested page tables.

11. The system of claim 9, wherein:

the inner portion of the operating system kernel comprises a memory management unit, a process management unit, and architecture specific code; and

each outer portion of the set of outer portions of the operating system kernel comprises a file systems unit and a device driver unit.

12. The system of claim 9, wherein processor is to: in response to detecting an attack on a container, a corresponding service, or a corresponding outer portion of the operating system kernel, take remedial action on the attacked container, service, or outer portion of the operating system kernel, wherein the other containers, services, and outer portions of the operating system kernel are unaffected.

13. A method, comprising:

loading, by a computing device, an inner portion of an operating system kernel in an inner region of a kernel space;

loading, by the computing device, a set of outer portions of the operating system kernel in an outer region of the kernel space;

executing, by the computing device, a service in a container of a set of containers in a user space, wherein each container is communicatively coupled with an outer portion of the set of outer portions of the operating system kernel, and wherein each container is divided from the inner portion of the operating system kernel and each container is isolated from all other containers in the set of containers; and

in response to detecting an attack on a container, a corresponding service, or a corresponding outer portion of the operating system kernel, taking, by the computing device, remedial action on the attacked container, service, or outer portion of the operating system kernel, wherein the other containers, services, and outer portions of the operating system kernel are unaffected.

14. The method of claim 13, wherein the outer region of the kernel space and the inner region of the kernel space are divided by nested page tables.

15. The method of claim 13, wherein:

the inner portion of the operating system kernel comprises a memory management unit, a process management unit, and architecture specific code; and

each outer portion of the set of outer portions of the operating system kernel comprises a file systems unit and a device driver unit.

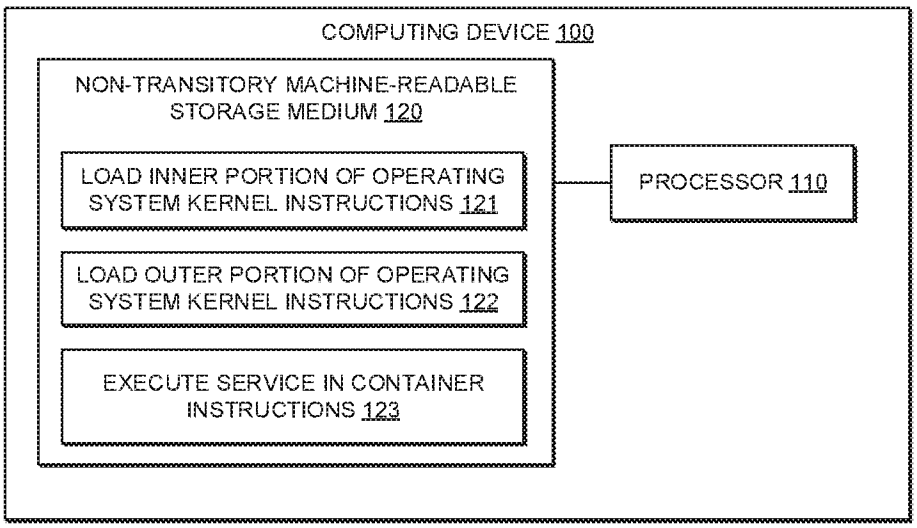
COMPUTING DEVICE 100
NON-TRANSITORY MACHINE-READABLE STORAGE MEDIUM 120
LOAD INNER PORTION OF OPERATING SYSTEM KERNEL INSTRUCTIONS 121
LOAD OUTER PORTION OF OPERATING SYSTEM KERNEL INSTRUCTIONS 122
EXECUTE SERVICE IN CONTAINER INSTRUCTIONS 123
PROCESSOR 110

FIG. 1

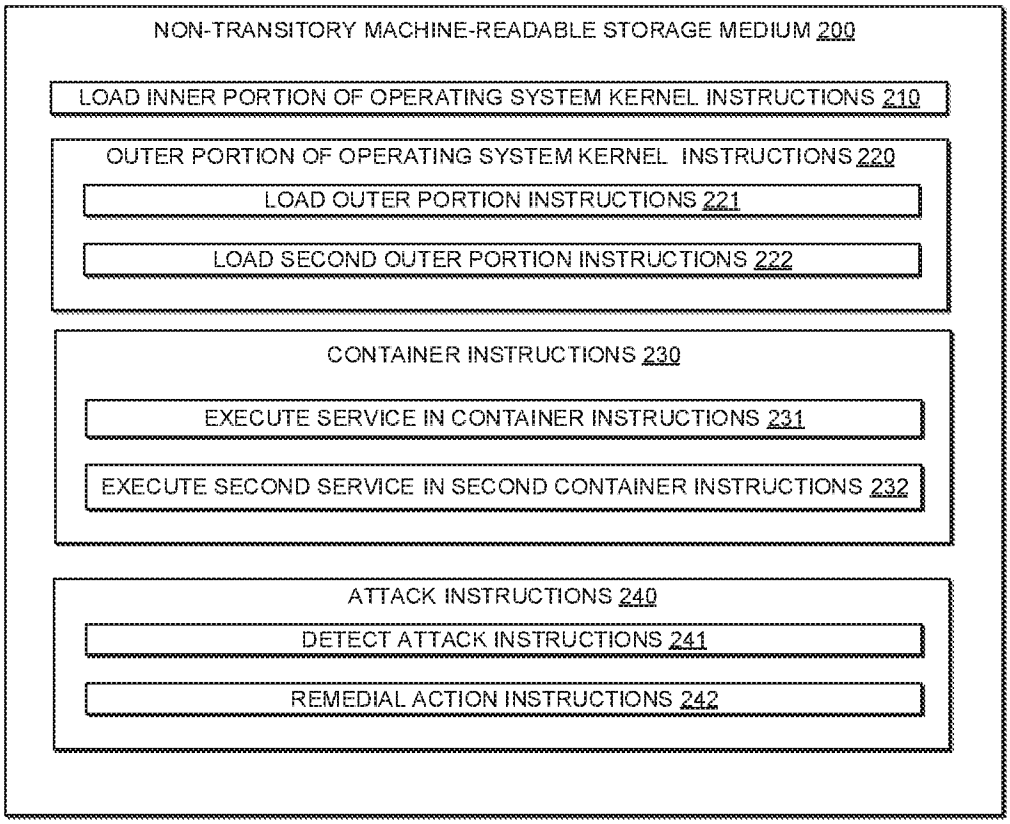
NON-TRANSITORY MACHINE-READABLE STORAGE MEDIUM 200
LOAD INNER PORTION OF OPERATING SYSTEM KERNEL INSTRUCTIONS 210
OUTER PORTION OF OPERATING SYSTEM KERNEL INSTRUCTIONS 220
LOAD OUTER PORTION INSTRUCTIONS 221
LOAD SECOND OUTER PORTION INSTRUCTIONS 222
CONTAINER INSTRUCTIONS 230
EXECUTE SERVICE IN CONTAINER INSTRUCTIONS 231
EXECUTE SECOND SERVICE IN SECOND CONTAINER INSTRUCTIONS 232
ATTACK INSTRUCTIONS 240
DETECT ATTACK INSTRUCTIONS 241
REMEDIAL ACTION INSTRUCTIONS 242

FIG. 2

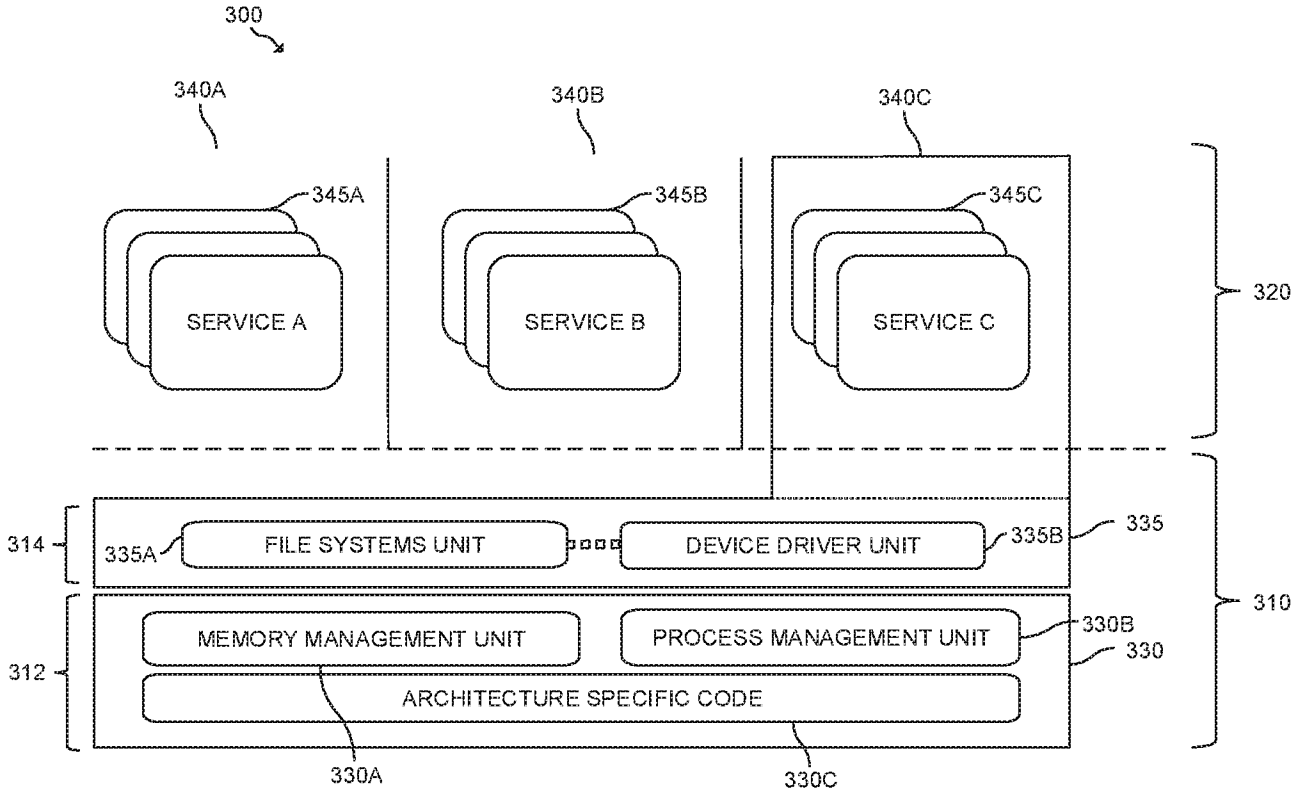
300
340A
340B
340C
345A
345B
345C
SERVICE A
SERVICE B
SERVICE C
320
314
335A
FILE SYSTEMS UNIT
DEVICE DRIVER UNIT
335B
335
310
312
MEMORY MANAGEMENT UNIT
PROCESS MANAGEMENT UNIT
330B
330
ARCHITECTURE SPECIFIC CODE
330A
330C

FIG. 3

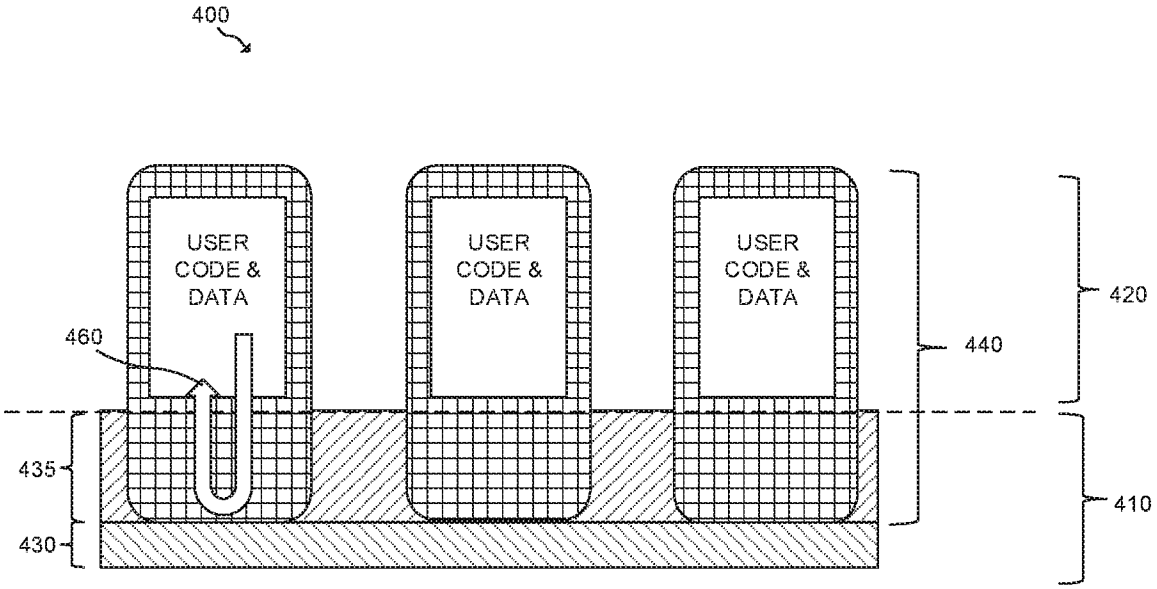
400
USER CODE & DATA
USER CODE & DATA
USER CODE & DATA
460
440
420
435
430
410

FIG. 4

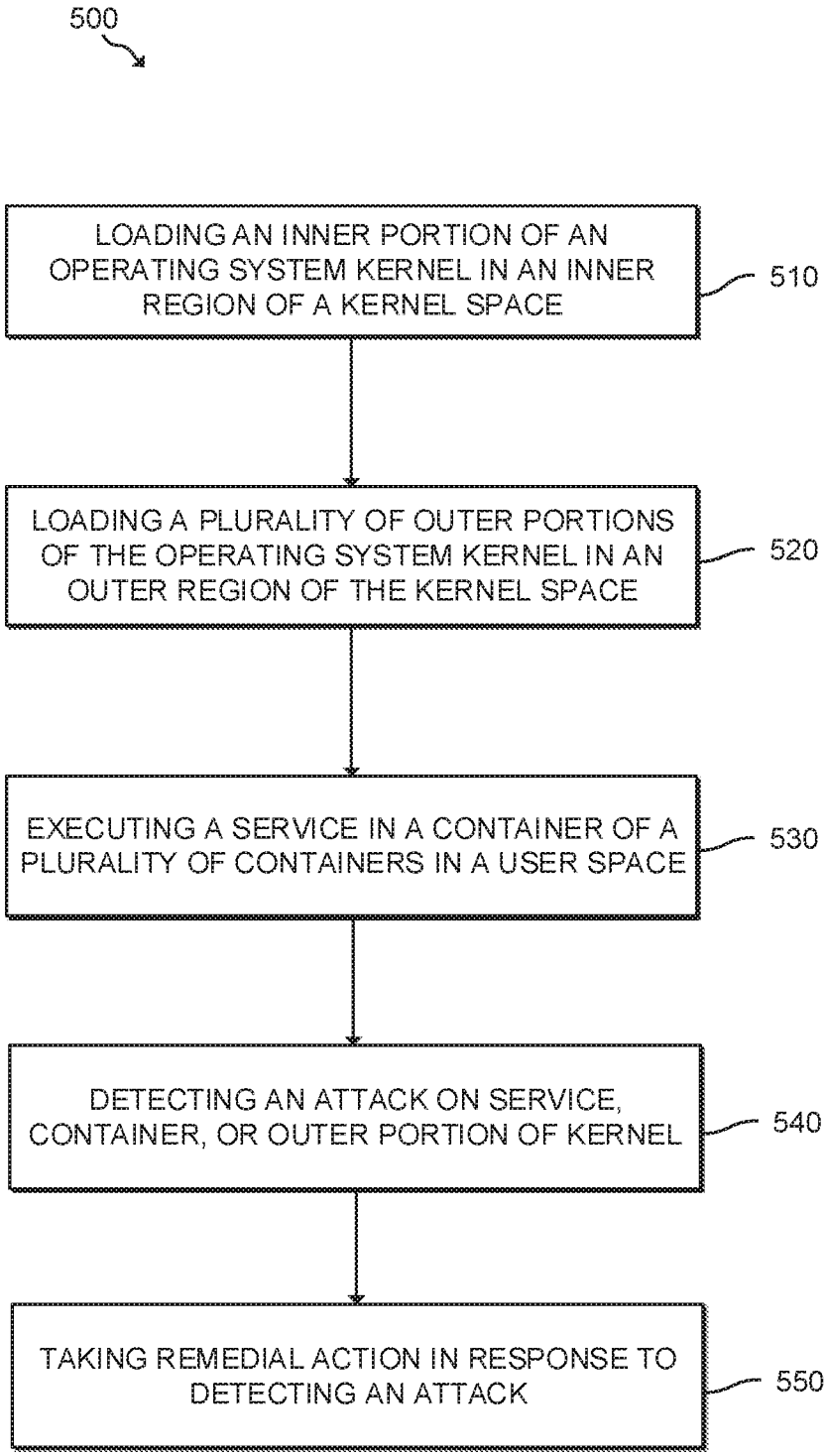
500
LOADING AN INNER PORTION OF AN OPERATING SYSTEM KERNEL IN AN INNER REGION OF A KERNEL SPACE
510
LOADING A PLURALITY OF OUTER PORTIONS OF THE OPERATING SYSTEM KERNEL IN AN OUTER REGION OF THE KERNEL SPACE
520
EXECUTING A SERVICE IN A CONTAINER OF A PLURALITY OF CONTAINERS IN A USER SPACE
530
DETECTING AN ATTACK ON SERVICE, CONTAINER, OR OUTER PORTION OF KERNEL
540
TAKING REMEDIAL ACTION IN RESPONSE TO DETECTING AN ATTACK
550

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/036366

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/54(2006.01)i, G06F 9/445(2006.01)i, G06F 9/44(2006.01)i

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 9/54; G06F 21/56; G06F 9/46; G06F 12/00; G06F 9/455; G06F 9/52; G06F 9/445

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: container, operating system, kernel, inner region, outer region, user space, attack, kernel space, isolation

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2015-0178497 A1 (BITDEFENDER IPR MANAGEMENT LTD.) 25 June 2015 See paragraphs [0027]-[0063]; and figures 1-A, 4.	1-15
A	US 2015-0150003 A1 (PAVEL EMELYANOV et al.) 28 May 2015 See paragraphs [0051]-[0053]; and figures 3-4.	1-15
A	US 2014-0075555 A1 (SUNIL NAMDEO SHILIMKAR) 13 March 2014 See paragraphs [0020]-[0047]; and figure 1.	1-15
A	US 2005-0251637 A1 (ADRIAN MARINESCU et al.) 10 November 2005 See paragraphs [0047]-[0048]; and figures 3-4.	1-15
A	US 8209704 B1 (PETER J. MCCANN et al.) 26 June 2012 See column 9, line 65 - column 11, line 27; and figure 2.	1-15

☐ 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 another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
17 February 2017 (17.02.2017)	**02 March 2017 (02.03.2017)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
CHIN, Sang Bum
Telephone No. +82-42-481-8398

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2016/036366

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2015-0178497 A1	25/06/2015	US 9117081 B2	25/08/2015
		WO 2015-174874 A1	19/11/2015
US 2015-0150003 A1	28/05/2015	EA 201301283 A1	29/05/2015
		US 9348622 B2	24/05/2016
US 2014-0075555 A1	13/03/2014	US 8973136 B2	03/03/2015
US 2005-0251637 A1	10/11/2005	US 2004-0193819 A1	30/09/2004
		US 6963960 B2	08/11/2005
		US 7127582 B2	24/10/2006
US 8209704 B1	26/06/2012	None	