

A METHOD FOR VIRTUAL MACHINE INTROSPECTION AT NATIVE SPEED

Field of the Invention

This disclosure relates to a method and system for reading a guest's Virtual Machine memory at native speed from the monitor Virtual Machine. Particularly, the disclosure relates to a method and system of implementing a new execution environment to access a guest's Virtual Machine memory at native speed.

Summary of the Prior Art

Live Virtual Machine Introspection (VMI) is a technique to extract the state information of a running virtual machine (the guest VM) from the outside, e.g., a trusted monitor VM. This technique is widely used for guest monitoring, forensics, rootkit detection and malicious application detection. Live VMI offers the advantage that the detection cannot be directly tampered with by malicious code running inside the guest VM. Due to this reason, Live VMI has become a mature field of research and consists of a sizeable literature. By and large, a VMI application consists of two components: a high-level semantic reconstruction logic and a memory access engine. The first component is responsible for reconstructing kernel semantic and detection malicious activities while the second component is to read the data from the guest memory based on the first component's demands.

The focus of existing VMI techniques is only upon semantic reconstruction while little effort is made to securely and efficiently access the guest memory.

Existing live VMI systems use a software-based approach to accessing the guest VM. A typical example is the LibVMI where the guest memory is accessed from the monitor VM in the following manner. First, the guest memory is mapped to the monitor VM. Given an object at a virtual address (VA), the access engine walks through the guest page table and extended page tables to find out the corresponding physical address (PA). It then re-maps PA to a new address VA' in its own address space and finally issues an instruction to read the memory word at PA.

Obviously, such a process incurs a much longer latency than the guest's own memory access which only requires execution of one instruction. The consequence of the long delay is not just about the low performance. More importantly, it undermines the

security of introspection because the memory access latency is long enough for the rootkit to maliciously modify a kernel object and then restore it.

Thus, one skilled in the art is striving to improve the system and method of reading guest Virtual Machine's memory.

Summary of the Invention

The above and other problems are solved and an advance in the art is made by a system and method provided by embodiments in accordance with this disclosure. A first advantage of embodiments of a system and method in accordance with this disclosure is that the guest memory can be read at native speed. Inevitably, this increases the introspection performance and provides stronger security. A second advantage of embodiments of a system and method in accordance with this disclosure is that the new execution environment, Immersive Execution Environment (or ImEE), is able to maintain consistent memory view between the guest and the ImEE since both environments physically use the common dynamic VA-to-IPA mappings and share the same constant IPA-to-PA mappings.

The above advantages are provided by embodiments of a system and method for creating an Immersive Execution Environment (or ImEE) for a Virtual Machine Introspection (VMI) in an operating system such that physical memory of a target Virtual Memory (VM) is accessed at native speed.

In accordance with an aspect of the disclosure, a method for creating an Immersive Execution Environment (or ImEE) for a Virtual Machine Introspection (VMI) in an operating system such that physical memory of a target Virtual Memory (VM) is accessed at native speed is provided in the following manner. The method comprises: allocating at least one of a plurality of Computer Processing Unit (CPU) cores and a physical memory region in a memory of the operating system; setting a CR3 register of the allocated CPU core identical to a CR3 register of a CPU core of the target VM such that the allocated CPU core uses a page table of the target VM for translating a virtual address (VA) into an intermediary physical address (IPA) by the CR3 register of the allocated CPU and the CR3 register of the CPU core of the target VM; preparing a first data structure and a second data structure in the allocated physical memory region for completing the IPA to Physical Address (PA) translation, wherein the first and second data structures are directed to different PA.

In accordance with an embodiment of this disclosure, the first data structure is executable in a user space. Further, the first data structure is an extended Page table with privilege settings of code set as read-only and executable and data set as read-writable.

In accordance with an embodiment of this disclosure, the second data structure is an extended Page Table with privilege settings of data set as read-only.

In accordance with another aspect of the disclosure, a system for creating an Immersive Execution Environment (or ImEE) for a Virtual Machine Introspection (VMI) in an operating system such that physical memory of a target Virtual Memory (VM) is accessed at native speed is provided in the following manner. The system comprises: a processor; and a non-transitory media readable by the processor, the non-transitory media storing instructions that when executed by the processor, cause the processor to: allocate at least one of a plurality of Computer Processing Unit (CPU) cores and a physical memory region in a memory of the operating system; set a CR3 register of the allocated CPU core identical to a CR3 register of a CPU core of the target VM such that the allocated CPU core uses a page table of the target VM for translating a virtual address (VA) into an intermediary physical address (IPA) by the CR3 register of the allocated CPU and the CR3 register of the CPU core of the target VM; prepare a first data structure and a second data structure in the allocated physical memory region for completing the IPA to Physical Address (PA) translation, wherein the first and second data structures are directed to different PA.

Brief Description of the Drawings

The above advantages and features in accordance with this invention are described in the following detailed description and are shown in the following drawings:

Figure 1 illustrating a VMI application running in random with an Immersive Execution Environment (ImEE) in accordance with an embodiment this disclosure;

Figure 2 illustrating a process executable by the hypervisor to set up the ImEE in accordance with an embodiment of this disclosure;

Figure 3 illustrating the address translation settings inside the ImEE after the hypervisor executes process in figure 2 in accordance with an embodiment of this disclosure; and

Figure 4 illustrating the information flow after the hypervisor executes process in figure 2 in accordance with an embodiment of this disclosure.

Detailed Description

This disclosure relates to a method and system for reading a guest's Virtual Machine memory at native speed from the monitor Virtual Machine. Particularly, the disclosure relates to a method and system of implementing a new execution environment to access a guest's Virtual Machine memory at native speed.

The technical problem that is solved by the system and method according to this disclosure is to securely and effectively read the guest Virtual Machine's memory at native speed from the monitor Virtual Machine (VM). The security challenges come from the fact that the guest operation system is not trusted. Particularly, the guest operation system could be malicious with intend to evade or sabotage introspection. When the Virtual Machine monitor reads the guest memory at virtual address (VA), the Virtual Machine monitor should read the same content as a guest Operation System reads the memory at VA. In other words, the introspection must reliably be consistent from the guest's actual view. This property is also challenging to achieve because the guest Operation System can modify the memory at any time in arbitrary ways. Lastly, the requirement of native speed access means that the VM monitor accesses the guest memory in the same speed as memory accesses within the guest Operation System.

To solve this problem, we introduce a hardware based method for the VM monitor to access the guest memory. Specifically, we propose a new execution environment called Immersive Execution Environment (or ImEE). The ImEE has a special setting for the address translation which allows the code inside the ImEE to access the guest memory at native speed and ensures that the output is identical to the guest at the same time.

Figure 1 illustrates a VMI application 150 running in tandem with the ImEE 110. Basically, the high-level introspection logic stays inside the VM monitor while the memory access engine 115 (called the ImEE agent) is located inside the ImEE 110.

Figure 2 illustrates a process 200 executable by the hypervisor to set up the ImEE 110. Process 200 begins with step 205 where the hypervisor allocates a dedicated CPU core 116 as well as a physical memory region 117 that stores the ImEE code and data. This memory region 117 is referred to as the ImEE pages.

In step 210, the hypervisor sets the CR3 register of the ImEE's CPU core 116 with the same value as the CR3 register of the guest CPU. As a result, the ImEE CPU core 116

will use the target guest page table 352 for translating a virtual address (VA) into an intermediary physical address (IPA).

In step 215, the hypervisor prepares two sets of data structures configured for second level address translation from IPA to Physical Address (PA) such as Extended Page Table (EPT) for Intel platform and Nested Page Table (NPT) for AMD platform. The first set of data structure is denoted as EPT_u 310 and is for translating ImEE virtual addresses in the user space. The second set of data structure is denoted as EPT_k 320 and is for translating the ImEE kernel virtual addresses. Using a 32-bit architecture with 4GB virtual address space having a 3GB/1GB split between user and kernel space, the first data structure EPT_u 310 is for translating 0-3GB of user space while the second data structure is for translating the 3GB-4GB of kernel space.

The privilege setting of the EPT_u is set as normal such that code pages are set as read-only and executable and the data pages are set as read-writable. The code running in the ImEE user space uses the target guest VM's page table 352 to complete the VA-to-IPA translation. Thereafter, EPT_u is used to complete the IPA-to-PA translation. As a result, given a virtual address VA (e.g., the address of a kernel symbol), the code in the ImEE reads the memory in exactly the same way as reading it from within the target guest VM.

The privilege setting of the EPT_k is set differently compared to EPT_u. In particular, the address mappings of the EPT_k are set in the same way as the EPTs used for the guest's kernel pages, and all those pages are set as read-only without execution or write permission. This ensures that no kernel page is executable and writable inside the ImEE to protect the security of introspection.

Figures 3 and 4 illustrate the address translation settings inside the ImEE after the hypervisor executes process 200. With the CR3 register of the ImEE's CPU core 116 being set to the same value as the CR3 register of the guest CPU, the memory management unit (MMU) 380 ensures that the target guest page table 352 maps the ImEE kernel IPA to the EPT_k 320 and ImEE user IPA to the EPT_u 310.

The solid line 350 describes how a kernel virtual address 390 is translated inside the ImEE 110. The ImEE kernel virtual addresses 390 are the addresses of the introspection target. In particular, ImEE kernel virtual addresses 390 are the locations in the guest VM which is the data the VMI application intends to read. The kernel VA 390 inside ImEE is firstly mapped to the kernel IPA using the same page table 352 in the guest VM. Whenever the guest kernel makes any change on its page table 352, the changes would be applied to the kernel IPA immediately. In other words, any changes in the page table 352 in the guest

kernel would be replicated on the kernel IPA. The kernel IPA is then mapped to the physical address via EPT_k 320. Although EPT_k 320 is a separated page table as the EPT 375 in the guest VM, both have the same mapping which do not change over the time.

The dash line 360 describes how a user virtual address 390 is translated inside the ImEE 110. For purposes of this disclosure, reference number 390 contains both user and kernel virtual addresses. The ImEE user virtual addresses 390 are addresses used by the ImEE code and data. The ImEE user virtual addresses 390 use the target VM's page table 352 to complete the VA-to-IPA translation, and uses the EPT_u 320 to complete the IPA-to-PA translation. As a result, given a virtual address (VA), the code inside the ImEE 110 reads the memory in exactly the same way as reading it within the target VM. Although the VA-to-IPA mapping also goes through the guest page table 352, the IPA is mapped into a separated physical page segment 361 because they use the EPT_u 310.

The dotted line 370 describes how an arbitrary virtual address is translated in the guest VM. The guest virtual addresses 377 are mapped to the physical memory as in regular VM. Due to the EPT restriction, none of guest virtual addresses is mapped out of the guest's pre-assigned physical memory pages 371 in the host physical memory.

In a system call, a checking program in the monitor VM can dumps all 349 entries of the guest's system call table. It starts with the table's base address indicated by the global symbol `sys_call_table`. The checking program passes the virtual address of `sys_call_table` and the length of the table to the ImEE. It is noted that the ImEE agent performs a continuous memory reading in the target page of the guest VM and returns all entries to the program.

In kernel module listing, the program lists all kernel modules currently loaded in the guest VM. The program traverses the module list pointed to by the global structure `modules`. For each module, the program reads the module's name and the next pointer in order to move forward. The program sends two requests to the ImEE agent for each available module with one for the module the name and the other for the next pointer. The program stops when the traversal reaches the end of the list.

The beneficial results of the ImEE based guest memory accesses can be summarized as high efficiency, high effectiveness and strong security. The ImEE achieves high efficiency because the ImEE agent reads the guest memory at native speed. In other words, the ImEE has the same speed as running inside the guest. The native speed access not only boosts the introspection performance, but also provides stronger security. It is much more difficult for the malwares in the guest kernel to evade detection, because the malwares

no longer have the speed advantage over the introspection. The high effectiveness is due to the fact that the consistency between the guest's memory view and the ImEE's memory view is always maintained. Both environments physically use the common dynamic VA-to-IPA mappings and share the same constant IPA-to-PA mappings. It is noted that the IPA-to-PA mapping is managed by the hypervisor. Therefore, it is trivial to maintain the consistency between the EPT_k and the EPT in the guest.

For purposes of this disclosure, a hypervisor also known as a virtual machine monitor is a program, firmware or hardware that allows multiple operating systems (also known as virtual machines) to operate via a host system. The hypervisor controls the host system's processor, memory and other resources, and allocates each virtual machine with the host system's processor, memory, and other resources where required. As hypervisor is well known in the art, a detailed description of a hypervisor is omitted for brevity.

The host system is essentially a processing system such as a computer, a server or an electronic mobile device. A typical host system includes Central Processing Unit (CPU) such as a processor, microprocessor, or any combination of processors and microprocessors that execute instructions to perform the processes; non-volatile memory, such as a Read Only Memory (ROM), that stores instructions and data to operate various sub-systems of processing system; volatile memory such as Random Access Memory (RAM), that stores the instructions and data needed by CPU to perform software instructions; and other I/O device, keyboard, display, network device 250 and any number of other peripheral devices. All various components of the host system are connected via busses to exchange data. Such typical host system is well known in the art. Hence, a detailed description of a host system is omitted for brevity.

The above is a description of embodiments of a method and system of implementing a new execution environment called Immersive Execution Environment in order to read the guest memory at native speed. It is foreseeable that those skilled in the art can and will design alternative method and system based on this disclosure that infringe upon this invention as set forth in the following claims.

CLAIMS:

1. A method for creating an Immersive Execution Environment (or ImEE) for a Virtual Machine Introspection (VMI) in an operating system such that physical memory of a target Virtual Memory (VM) is accessed at native speed, the method comprising:
 - allocating at least one of a plurality of Computer Processing Unit (CPU) cores and a physical memory region in a memory of the operating system;
 - setting a CR3 register of the allocated CPU core identical to a CR3 register of a CPU core of the target VM such that the allocated CPU core uses a page table of the target VM for translating a virtual address (VA) into an intermediary physical address (IPA) by the CR3 register of the allocated CPU and the CR3 register of the CPU core of the target VM; and
 - preparing a first data structure and a second data structure in the allocated physical memory region for completing the IPA to Physical Address (PA) translation, wherein the first and second data structures are directed to different PA.
2. The method according to claim 1 wherein the first data structure is executable in a user space.
3. The method according to claim 1 or 2 wherein the first data structure is an extended Page table with privilege settings of code set as read-only and executable and data set as read-writable.
4. The method according to anyone of claims 1-3 wherein the second data structure is an extended Page Table with privilege settings of data set as read-only.
5. A system for creating an Immersive Execution Environment (or ImEE) for a Virtual Machine Introspection (VMI) in an operating system such that physical memory of a target Virtual Memory (VM) is accessed at native speed, the system comprising:
 - a processor; and
 - a non-transitory media readable by the processor, the non-transitory media storing instructions that when executed by the processor, cause the processor to:
 - allocate at least one of a plurality of Computer Processing Unit (CPU) cores and a physical memory region in a memory of the operating system;
 - set a CR3 register of the allocated CPU core identical to a CR3 register of a CPU core of the target VM such that the allocated CPU core uses a page table of the target VM for translating a virtual address (VA) into an intermediary physical address (IPA) by the CR3 register of the allocated CPU and the CR3 register of the CPU core

of the target VM; and

prepare a first data structure and a second data structure in the allocated physical memory region for completing the IPA to Physical Address (PA) translation, wherein the first and second data structures are directed to different PA.

6. The system according to claim 5 wherein the first data structure is executable in a user space.

7. The system according to claim 5 or 6 wherein the first data structure is an extended Page table with privilege settings of code set as read-only and executable and data set as read-writable.

8. The system according to anyone of claims 5-7 wherein the second data structure is an extended Page Table with privilege settings of data set as read-only.