2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

HyperPS: A Hypervisor Monitoring Approach Based on Privilege Separation

Kun Zhang, Wenqing Liu, Kunli Lin, Bibo Tu
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
{zhangkun, liuwenqing, linkunli, tubibo}@iie.ac.cn

***Abstract*—In monolithic operating system (OS), any error of system software can be exploit to destroy the whole system. The situation becomes much more severe in cloud environment, when the kernel and the hypervisor share the same address space. The security of guest Virtual Machines (VMs), both sensitive data and vital code, can no longer be guaranteed, once the hypervisor is compromised. Therefore, it is essential to deploy some security approaches to secure VMs, regardless of the hypervisor is safe or not. Some approaches propose microhypervisor reducing attack surface, or a new software requiring a higher privilege level than hypervisor.**

In this paper, we propose a novel approach, named HyperPS, which separates the fundamental and crucial privilege into a new trusted environment in order to monitor hypervisor. A pivotal condition for HyperPS is that hypervisor must not be allowed to manipulate any security-sensitive system resources, such as page tables, system control registers, interaction between VM and hypervisor as well as VM memory mapping. Besides, HyperPS proposes a trusted environment which does not rely on any higher privilege than the hypervisor. We have implemented a prototype for KVM hypervisor on x86 platform with multiple VMs running Linux. KVM with HyperPS can be applied to current commercial cloud computing industry with portability. The security analysis shows that this approach can provide effective monitoring against attacks, and the performance evaluation confirms the efficiency of HyperPS.

***Index Terms*—Virtualization, Hypervisor Monitoring, Hypervisor Security, Cloud Computing**

I. Introduction

In general, the larger code base of a typical hypervisor consequently increases the risk of having security vulnerabilities. Furthermore, successful VM escape attacks, as well as the emerging hypervisor rootkits, greatly make threats for VMs. In light of these attacks, there is a pressing need to investigate effective ways to monitor the hypervisor. In addition to larger code base of hypervisor, monolithic system design can also increase the risk of being attacked. Software with monolithic design integrates core services into one huge code base, thereby encompassing them all in a single address space and executing them in the same processor privilege level (i.e., ring 0 and VMX-root modes in Intel x86 or svc and hyp modes in ARM). OS kernels define and store access control policies in main memory, hypervisor controls VM memory allocation and interaction data access between the VM and the hypervisor where any code executing within the hypervisor can modify. An exploit of any part of the hypervisor allows complete access to all memory and critical resource. Privilege separation [2] has been considered as a fundamental principle that can separate the critical capability from the hypervisor and remove such a security concern.

In order to reduce the threat to other components or the whole system caused by attackers, and enforce this security principle in the design, current researchers achieve privilege separation by introducing microhypervisor or creating more privileged software.

Microhypervisor By abandoning the monolithic design and modularizing core services, the microhypervisor design achieves privilege separation. Trustvisor[6] uses a microhypervisor to provide privilege separation and verify security sensitive workloads, eventually prevents damage of a fraction of the hypervisor from spreading across the whole system. Nova[11] uses microhypervisor to provide privilege separation and memory protection for the virtualization layer. Nevertheless, this design has been deemed unrealistic for cloud providers in that it usually necessitates reorganization of the current system software architecture.

A Higher Privilege-level Software In order to mitigate the hazard caused by the hypervisor, plenty of solutions propose and introduce a higher privilege-level software than the original hypervisor. Nested virtualization [15], [9] is one of the representative approaches to isolate hypervisor, which provides a higher privilege-level execution environment to run the monitor securely. CloudAuditor [15] and Nosv [9] are examples of systems that propose nested virtualization idea to achieve isolation for protected resources.

In this paper, we introduce a novel prototype, named HyperPS, to implement hypervisor monitoring based on privilege separation in every level of privileged software without relying on a higher privileged software or microhypervisor. It has applicability for multi-platforms (x86, mips and so on), few modifications for original hypervisor, and practicability for cloud providers. We take x86 architecture as an example.

The key of the design is to create an isolated space by separating privileges, in which HyperPS can monitor runtime hypervisor. HyperPS ensures that the hypervisor never disables protection-mechanism (e.g., DEP, SMEP) by efficiently decoupling updating privilege of critical registers into HyperPS World. Hypervisor monitoring is executed by monitoring key data structures including Extended Page Tables (EPT) and Virtual Machine Control Structure (VMCS), so as to avoid malicious attacks causing the whole system compromised.

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00141

VMCS is used in VMX operation to manage the behavior of VMs as well as transitions between the VM and the hypervisor. Thus, the modification of content of VMCS, especially the Guest-state filed and VM-execution control field, may cause unpredictable consequences. EPT contains the mapping relationship from Guest Physical Address (GPA) to Host Physical Address (HPA). Given the great importance of the data structures mentioned above, HyperPS isolates them beyond access from compromised parts of system.

Our prototype introduces 3K SLOC (Source Lines of Code) to hypervisor monitoring and 270 SLOC modifications of the hypervisor. The experimental results show trivial performance overhead and independence on multi-platforms for runtime hypervisor monitoring.

Our contributions are as follows:

- A practical event-driven hypervisor monitoring approach based on privilege separation in a variety of system software with hypervisor processor privilege level.
- A non-bypassable protection approach for VMCS and EPT which can ensure the security of interaction between VM and hypervisor as well as VM memory mapping.
- A prototype based on KVM and x86 architecture with trivial performance overhead, high security and hardware independence.

The rest of this paper is organized as follows. Section II describes background. Section III discusses our threat model and assumption. Section IV elaborates on the design and implementation of HyperPS on x86 platform. Section V gives the evaluation of security and performance. Section VI compares HyperPS with previous work. At last, Section VII gives the conclusion.

II. Background

A. VMCS

Virtualization technique imports two kinds of operation mode, Root and Non-root mode. To better support CPU virtualization, VMCS structure, a data structure based on hardware, is imported to manage transitions out and into of VMX Non-root mode from VMX Root mode (VM Exit and VM Entry), as well as processor behavior. This structure is divided into six information fields, including Guest-state filed which records VM information and Host-state filed which records host OS information. It causes VM Exit that the vCPU switches from root mode to non-root mode. The reverse operation is called VM Entry that transiting from non-root mode to root mode. During these two operations, VMCS is used primarily for vCPU to query and update key CPU information such as system privileged registers.

B. VM Exit/VM Entry

Intel's VT-X introduces VMX operations, VM Exit and VM Entry. VM Exit is primarily due to that the VM has some sensitive instructions that cannot be performed properly and requires hypervisor's assistance to complete sensitive instruction execution. After VM Exit is finished, CPU needs to execute VMLAUNCH/VMRESUME instruction to initiate VM Entry.

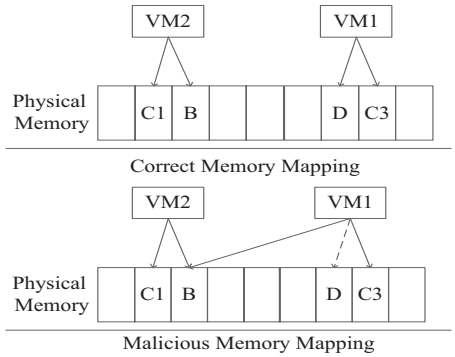

Fig. 1. The execution process of double mapping attack.

The interaction that is triggered by VM Exit process from the VM to hypervisor is as follows. The execution flow of VM Entry is opposite to that of VM Exit.

1) VM makes interrupt request for VM Exit, and the CPU in non-root mode first records VM Exit reason to the corresponding information filed in VMCS and saves the CPU state to the Guest-state filed of VMCS. 2) The CPU accesses the Host-state filed of VMCS and loads it into CPU. 3) The CPU switches from non-root mode to root mode, reads VM Exit reason in VMCS, and jumps to VM Exit handler function entry. 4) CPU initiates VM Entry and run the VM.

C. Address Translation of VM

The address translation of VM requests two page tables, guest page table and EPT. Guest page table can finish the translation from guest virtual address (GVA) to GPA. This translation is finished by VM. The translation from GPA to HPA is executed based on EPT by hypervisor. Intel's VT-x provides EPT, which directly supports address translation from GPA to HPA on hardware, greatly reduces the difficulty of memory virtualization and further improves the performance of virtualization.

III. Threat Model and Assumption

A. Threat Model

We assume that the hypervisor and kernel have been compromised and controlled by the powerful adversary. The adversary can turn off kernel security mechanisms, such as DEP,SMEP, inject code, even tamper page table mappings of memory region, and then implement attacks based on two attack paths, compromising critical interaction data and VM address mapping. First, the adversary can subvert the critical interaction data during the context switching process between VM and the hypervisor. Second, the adversary can tamper values of EPT entry causing memory corruption attacks, such as remapping attack, double mapping attack and disrupting isolation among EPTs.

a) Modifying the Interaction Data: For the modification of the critical interaction data during the context switching process, attackers can obtain the address of VMCS and modify it, such as HOST_RIP, GUEST_CR0, EPTP, et al. For example, modifying the value of privilege register, CR0, can close DEP mechanism, and modifying CR4 can close SMEP mechanism.

b) Modifying the Address Mapping of EPT: Modification of EPT entry can result in memory information leakage. There are three attack scenes, double mapping, remapping attack and disrupting isolation among EPTs of VMs.

Scene 1. For double mapping attack, attackers control and compromise a VM, then obtain the privilege of hypervisor through VM escape attack, and maliciously access the VMCS structure to obtain the value of EPT_Pointer(EPTP). The attack process is as shown in Figure 1. There are two guest VMs, VM1 (attacker) and VM2 (victim). In this way, EPTP of VM1 and EPTP of VM2 are respectively obtained by attackers. Also, for a guest virtual address (GVA) in VM2, named 'A', is mapped to the corresponding real physical address, named 'B'. For VM1, the real physical address corresponding to the guest virtual address 'C' is 'D', then 'D' is modified to be 'B' by modifying the value of the last page item of EPT. At last, 'A' and 'C' are mapped to the real physical address 'B', no address is mapped to 'D'. Then VM1 can access the data of VM2 successfully through accessing 'B', this attack process is called double mapping attack.

Scene 2. In a remapping attack scenario, we assume there are two VMs, VM1, a VM controlled by attackers to attack other VMs, VM2, a victimized VM. A physical page (named 'A') used by VM2 is released after it is used and it is released with VM2's content. After 'A' is released, VM1 remaps a GVA (named 'E') to 'A'. By this way, VM1 can access the content on 'A' used by VM2 through 'E', causing information leakage. Through the analysis of these two kinds of attack models, it is necessary to achieve attack prevention.

Scene 3. For disrupting the isolation among EPTs, there is no isolation among current EPTs, and attackers can compromise a VM and hypervisor first, then access memory of VM2 (victim) through VM1 (attacker) through making VM1 load VM2's EPT.

B. Assumption

We propose some assumptions. First, we assume hardware resources are trusted including processor, buses, BIOS, UEFI and so on, the trusted boot based on hardware can ensure the security and integrity of bootloaders. The Trusted Computing Base (TCB) contains created HyperPS and hardware resources. Second, this paper does not consider denial of service attack (DOS), side channel attack and hardware-based attack, such as cold-boot attack and RowHammer. Third, we assume that there exists no software that runs at a higher privilege level than the hypervisor.

IV. Design and Implementation

In this section, we present the design and implementation details of HyperPS, focusing on the two core mechanisms, VMCS and EPT monitoring.

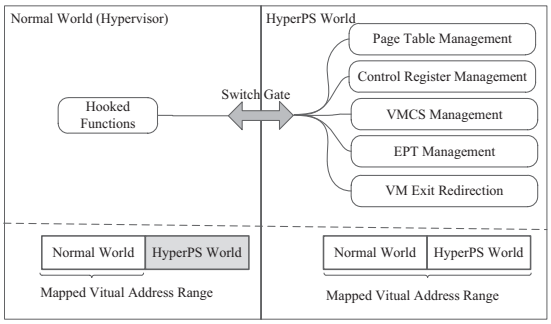

Fig. 2. The overview of HyperPS.

A. Overview

HyperPS is designed to provide a secure domain to provide hypervisor monitoring against attackers without depending on a higher privilege-level software than the hypervisor or microhypervisor. Figure 2 depicts the overview of HyperPS which creates other secure address spaces based on a set of new page table. It is composed of three parts, Normal World, HyperPS World and Switch Gate. A pivotal condition for HyperPS is that Normal World (origin hypervisor) must not be allowed to manipulate any security-sensitive system resources, such as EPT, VMCS, page tables and system control registers. To meet this condition, Normal World is deprived of controlling authority over security-sensitive system resources. Therefore, VMCS and EPT are hidden in HyperPS World from Normal World accessing. Instead, Normal World is only allowed to send requests through a specified interface (Switch Gate) to HyperPS World for controlling these resources. To assess the security of interaction between hypervisor and VM, VM memory mapping, management on VMCS and EPT, are hooked and trapped to HyperPS World. Page table and control register management is used to guarantee the security of HyperPS World. VM Exit redirection can balance VM Exit and VMCS management. Upon receiving such a request, HyperPS World determines whether to accept or reject it. More importantly, only virtual address of Normal World is mapped in page table of Normal World while all virtual address of two worlds is mapped in page table of HyperPS World.

While the hypervisor together with guest VMs runs in Normal World, the hypervisor is forced to request HyperPS to perform two operations on its behalf: 1) switching context (VMCS monitoring) between the hypervisor and VM, 2) EPT monitoring. With these designs, HyperPS intercepts and monitors VMCS and EPT access. Furthermore, HyperPS guarantees security of interaction and address translation of VMs.

B. Security-sensitive Resource Access

For the implement, the hypervisor corresponding to Normal World should be modified to be deprived of control capabilities for security-sensitive system resources, including control registers, page table, VMCS and EPT. In order to clarify the resource access in HyperPS World, hooks, unintended

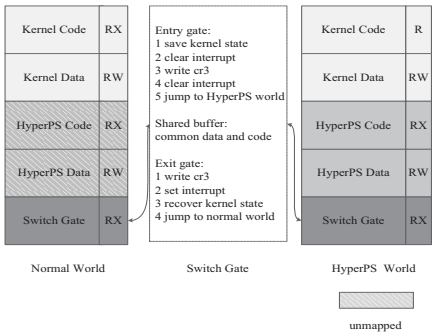

Fig. 3. An overview of address space layout.

instruction elimination and interaction flow are described as follows.

Hooks. We recognize control register access, page table updating, VMCS and EPT operations in Normal World as designated hypervisor events. To tackle the threats from compromised hypervisor, hypervisor monitoring invokes the integrity assessment routines upon the occurrences of designated hypervisor events. A common approach to capture designated events is to place hooks into the origin hypervisor and eliminate related privilege instructions. These hooks can be code hooks (jump instructions) inserted at arbitrary locations in hypervisor code, or any other technique that can transfer the control flow.

Unintended Instruction Elimination. Instructions composed of several fields in x86 are unintended. One or several of these fields may casually match another privileged instruction causing privilege instructions expected to be eliminated retained and hook failure. The general situation is as follows: 1)One instruction contains another privilege instruction. 2)A privilege instruction can be formed across two instructions. There is a threat that attackers jump into the middle of the instructions to execute a privileged instruction which has been eliminated and hooked by us. For the first case, the attack is blocked by replacing the same kind of resource, such as replacing the register in the instruction with another register. The solution to prevent the second attack is that we insert NOP instruction in the middle of these two instructions.

Interaction Flow. When the hypervisor's execution reaches a hook, it will transfer the control flow to HyperPS World. By implanting hooks at the code positions where hypervisor events occur, HyperPS can capture the occurrences of designated events to assess the sensitive resource integrity.

C. HyperPS World

The creation of HyperPS World has two purposes: 1) creating a address space which can provide monitoring for VMCS and EPT. 2) creating a software system that does not depend on hardware devices and adapts to multi-system platforms. The key point of its design is that it creates address space used for privilege separation for all privilege levels at the same privilege-level with hypervisor or kernel.

Creating HyperPS World. We create another page table to achieve an isolated address space, named HyperPS World, from compromised hypervisor. Figure 3 describes the address space layout of two worlds through two sets of page tables, the normal page table and HyperPS page table. On the left of Figure 3, the normal page table contains code and data of Normal World except for that of HyperPS World. This can prevent compromised hypervisor from breaking the integrity of HyperPS World. Programs running in Normal World cannot access data in HyperPS World. On the right of Figure 3, all code and data segments are mapped in HyperPS page table. HyperPS code remains executable and HyperPS data remains writable. What's the most important, kernel code is forbid to execute when HyperPS World is active, so that it cannot attack HyperPS World.

Creating Switch Gate. HyperPS designs and implements Switch Gate to perform worlds switching by executing a series of ordinary instructions. Switch Gate performs the control switching operation between Normal World and HyperPS World, acting as a wrapper function for a handler which processes incoming requests in HyperPS World. It provides Normal World with a unique way to enter HyperPS World. In addition, Switch Gate is implanted across Normal World and is invoked with specific parameters in order to handle sensitive resources including control registers, page tables, VMCS and EPT, by sending relevant requests to HyperPS World.

Figure 3 describes the details of Switch Gate and it is divided into the entry/exit gates and shared buffer. Entry gate provides the only entrance to HyperPS World and the exit gate provides the address for returning to Normal World. The shared buffer contains common data and code in Normal World and HyperPS World. Common code is switch code and common data includes the entry address of two sets of page tables. Of course, the entrance address must be protected after switching to HyperPS World in case that attackers access HyperPS World causally after trusted boot. This will be introduced in section IV-E.

D. Hypervisor Monitoring

VMCS and EPT are the most two important data structures that the hypervisor can utilize to interact with VM. VMCS, the architectural in-memory structure, is used to manage the mode transitions as well as specify the restricted operations causing the VM Exit/Entry. EPT contains the mapping relationship from GPA to HPA. The VMCS can be also used to specify extended page tables (EPTs) through EPTP that control the memory access in non-root mode. The big point of vulnerability is that the content of the VMCS and EPT would be synchronized to associated in-memory region (named VMCS region and EPT region in x86), the hypervisor could bypass the interception via directly accessing the region. If the hypervisor is compromised by attackers, during the interval between VM Exit/Entry, some attacks can be conducted to subvert VM as follows.

1) Attackers can illegally get the address of VMCS in memory region and modify the content of VMCS directly. For example, it can falsify the HOST_RIP field causing control flow hijack attack or tampering the EPTP filed with a dedicated illegal EPT. 2) Attackers can illegally modify the content of EPT entries and conduct remapping or double mapping attack to the VM. 3) Malicious VM can load EPT of any VM and access this VM's normal memory illegally.

VMCS Management and EPT Management. Obviously, the key node of attack path is to get the addresses of these two data structures. Hence, we straightforwardly provide the protection for these data structures by using HyperPS World. These two data are hidden in HyperPS World in case of malicious access from attackers with hypervisor privilege. So hypervisor requests HyperPS to handle VMCS and EPT management operations, intercept writing operations and return the corresponding result for the legal request.

Hardware virtualization includes a set of privilege instructions (vmptrld, vmptrst, vmclear, vmread, vmwrite) to load, store, clear, read and write the current VMCS. HyperPS intercepts the execution of these instructions by eliminating them from hypervisor's code, so that any manipulation on the VMCS can be validated by HyperPS. This is what VMCS Management in Figure 2 does. HyperPS places hooks on EPT functions, including EPT creating, loading, walking and destroying, then dispatches them to HyperPS World and handles appropriately. We also store EPT in HyperPS World so that we can block attacks on malicious writing EPT in memory. Eventually, HyperPS avoids double mapping attack and remapping attack during EPT updating.

VM Exit Redirection. During VM Exit, hypervisor needs to access VM Exit reason data of VMCS, and deal with the VM exit event. To avoid hooking and trap all VM Exit handlers to HyperPS World, VM exit redirection is designed to finish VM Exit process. After accessing VM exit reason from VMCS in HyperPS World, the control flow switches to hypervisor and hypervisor executes VM exit event handler functions.

TABLE I
VM-MARK TABLE.

VM-Mark Table			
Label	VMID	EPTID	EPT_Address
Description	The VM Identifier	The EPT Identifier	The Entry Address of EPT

EPT Isolation. In addition to directly accessing EPT in memory to tamper VM mapping, attackers can disrupt the isolation among EPTs by loading the EPT of another VM, indirectly access memory data on the VM. It is important to ensure EPT isolation and one VM only access own corresponding EPT. To ensure one EPT for one VM, HyperPS creates the VM-Mark structure stored in HyperPS World as Table I described. It records VMID, EPTID, EPT_Address and binds them together. VMID is created when the VM is created. EPTID and EPT_Address is recorded as long as the EPT of current VM is created. This table is destroy once the VM is shutdown or destroyed.

E. Security Guarantee for HyperPS World

The security of HyperPS World guarantees the security of HyperPS, because it relies on HyperPS World to provide a trusted environment for VMCS and EPT monitoring. Nevertheless, without any protection measures, kernel page table in Normal World is not secure for four reasons: 1) Attackers can control page table with the highest privilege after hypervisor is compromised. 2) Attackers can bypass Switch Gate to break the security of HyperPS World. 3) Attackers with the highest privilege can free to execute privilege instructions to modify the value of privilege registers, such as CR0, CR3 and so on. 4) Attackers can carry out DMA attack to access address space of HyperPS World casually. We detail the protection measures to resist these four types of attacks below.

Protecting Page Table. There are three reasons for controlling the two sets of page tables: 1) To access casually or bypass HyperPS World, attackers can tamper normal page table to map address of HyperPS World or load malicious page table to CR3. 2) Attackers can cover the hooked functions, redirect the functions to malicious code and bypass HyperPS World. 3) To break HyperPS World, malicious kernel code in Normal World with execution permission can be executed to subvert HyperPS.

For the first attack, we remove all entries that map to HyperPS World from the page table in Normal World and deprive the ability to access CR3 of the kernel. For the second attack, we intercept the accessing operation to CR0 and maintain the WP bit as 1. We stick to W$\oplus$X and maintain the code segment of hooked functions unwritable. For the third attack, we set the kernel code segment as NX (non-executable) when HyperPS World is running. For more security, we modify the kernel to configure these two sets of page table as read-only by setting the page tables page unwritable. This is necessary to prevent the page tables from being modified by attackers. Privilege instructions about page fault and setting pte are hooked and trapped to HyperPS World. Any write permission modification of two sets of page table must cause the kernel to page fault, then we dispatch page fault to HyperPS World to verify the correctness of address mapping.

Worlds Switching Securely. HyperPS creates Switch Gate between Normal World and HyperPS World by loading entry address of page table into CR3. In order to ensure switch security, we design the switch process as follows.

The switching process described in Figure 3 is as follows: 1) Save the kernel state to the stack including general registers and interrupt enable/disable status. 2) Clear the interrupt with the CLI instruction. 3) Load the page table to the register CR3. 4) Interrupt again. 5) Jump to the HyperPS World. For the exit process, the control flow returns to Normal World by performing the operations in the reverse order.

Accessing Control Registers Securely. The hypervisor without HyperPS is privileged and it can free to execute privilege instructions, so that it can write any value to the related control registers. 1) Malicious attackers can close DEP mechanism by modifying CR0, close SMEP mechanism by

modifying CR4. 2) Kernel code can load a crafted page table to bypass HyperPS World by converting a meticulously constructed address of one page table to CR3. To prevent these, HyperPS deprives sensitive privilege instructions executed by the hypervisor, and dispatches captured events to HyperPS World. HyperPS World can choose how to handle this event, such as executing, issuing alerts or terminating the process.

Resisting DMA Attack. DMA operation is used by hardware devices to access physical address directly. Malicious attackers can read or write arbitrary memory regions including HyperPS World by DMA. Therefore, it is a crucial focus of intercepting direct access to physical pages belonging to HyperPS World by DMA operation. Fortunately, HyperPS employs IOMMU mechanism to avoid DMA attack, which can carry out access control for DMA access. Our approach adopts two policies: 1) We remove the corresponding mapping of the critical data from the page table which IOMMU uses. These critical unmapped data includes the entrance address of HyperPS, data recording VM-Mark structure used in EPT management and so on. 2) HyperPS intercepts the address mapping functions about I/O, verifies whether the address belongs to the address space of HyperPS World, then chooses to map or unmap.

Through the above security measures, HyperPS can be protected from being bypassed and being broken, thus providing a secure execution environment for hypervisor monitoring.

V. Evaluation

In this section, we first analyze the security guarantees provided by HyperPS. Then, we evaluate the performance overhead by running a set of benchmarks on both standard KVM and HyperPS.

A. Security Analysis

In this section, we elaborate the security evaluation on how HyperPS achieves hypervisor monitoring through monitoring VMCS and EPT manipulations. In addition, we analyze the security of HyperPS itself. Table II shows the real attack instances in line with the above attack model.

Modifying VMCS Attack. We clarify the importance of protecting interaction data including VMCS in Section IV-D. Firstly, VMCS is hidden in HyperPS World and cannot be accessed by hypervisor. Secondly, functions that can access VMCS are hooked into HyperPS World, therefore, no functions outside HyperPS World can access VMCS. Attackers cannot get location of VMCS and access it. This prevents attackers from tampering interaction data. We examine protection for VMCS by conducting several attack cases which are widely adopted in real world. The modifying VMCS attack, one of attack cases listed in Table II, which tries to tamper the Guest_CR3 field of VMCS, fails because it loses the access privilege to VMCS. According to the above analysis, attackers cannot conduct further attacks which rely on VMCS access.

Subverting Memory Across VMs Attack. The main attacks that attackers can execute on subverting memory are double mapping attack and remapping attack by modifying EPT entry. Firstly, double mapping attack succeeds by allocating memory pages that have already been owned by a hostile VM to a victim VM. Secondly, another challenge is page remapping attack by a compromised hypervisor from a victim VM to a conspiratorial VM. This attack involves remapping a private page to another virtual address. The absence of EPT in Normal World can prevent modification from attackers because they cannot access EPT in memory region directly.

TABLE II
Hypervisor Attacks Against HyperPS.

Attack	*Description*
Modifying VMCS Attack	Load a crafted GUEST_CR3 value
CVE-2017-8106	Load a crafted EPT value
DMA Attack	Access HyperPS World by DMA
Code Injection Attack	Inject code and cover hooked functions to bypass HyperPS World

We implement a real attack by exploiting CVE-2017-8106 in kernel version 3.12. A privileged KVM guest OS user accesses EPT, conducts attacks via a single-context INVEPT instruction with a NULL EPT Pointer. Attackers cannot implement successfully and incur EPT access fault because HyperPS hides the address of EPT in HyperPS World and hijacks the loading of EPT. HyperPS verifies the value of EPT to avoid loading NULL value. Therefore, HyperPS can avoid subverting memory across VMs including double mapping attack, remapping attack as well as malicious EPT access.

Destroying HyperPS World. HyperPS World is created by relying on new page table. We analyze the protection for HyperPS World from four aspects, page table modification attack, hooks redirection attack, register modification attack and DMA attack.

1) Page Table Modification Attack: Page table protection has been introduced in section IV-E. The entry address mapping of the HyperPS page table is deleted from the old page table to prevent the kernel from accessing HyperPS World directly through the page table mapping. When HyperPS World is active, the kernel code does not have any executable permissions to prevent attacking a process running in HyperPS world. Attackers may attack in two ways. First, attackers may try to directly access the new page table address on the kernel page table by virtual address mapping. When he accesses it, there is page fault due to the absence of virtual to physical address mapping. Second, attackers may run kernel code while HyperPS World is active to attack programs running in HyperPS World. This can be prevented because of the absence of executable privilege of kernel code.

2) Hooks Redirection Attack: Due to the code of hooked functions including page table updating, control register access operations and VMCS operations, EPT operations, are writable-protection. Accessing CR0 register operation used to set W$\oplus$X is controlled and page table updating used to change code execution privilege is limited, attackers cannot redirect hooked functions and bypass monitoring.

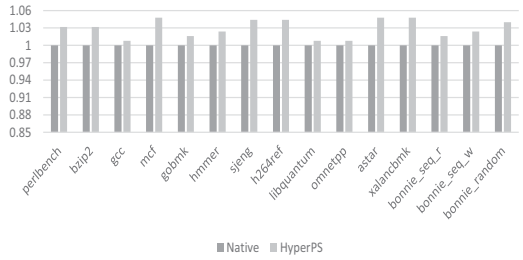

Fig. 4. Performance evaluation.

3) Register Modification Attack: Some register access operations including CR0, CR3, CR4, are controlled and hooked to HyperPS World. HyperPS completes the update and monitoring of registers to ensure that the security mechanism is enabled while the system is running. Protection for page table, hooked functions and registers, plays a role mutually in protection for HyperPS.

4) DMA Attack: DMA attack is described in detail in section IV-E. Attackers can use this feature to read or corrupt arbitrary memory regions. DMA attack is not a threat to HyperPS, because HyperPS is inherently secure against DMA using IOMMU. We remove the corresponding mapping of the critical data from the page table which IOMMU uses. These critical unmapped data includes the entrance address of HyperPS, VMCS, EPT and VM-Mark structure. HyperPS intercepts the related functions of the DMA mapping and verifies that the physical address to be mapped is not within the scope of the critical data address during the DMA mapping process. DMA attack that aims at modifying the VM memory or the page tables can also be defeated.

B. Performance Evaluation

In HyperPS, the hypervisor is modified so that HyperPS World is initialized during the boot up sequence. This includes creating a new memory page table for HyperPS, allocating memory pages. VM startup can cause memory allocation and I/O access. These processes introduce security accessing and verification for VMCS in HyperPS World during VM Exit/Entry sequence. The hypervisor is modified to place hooks upon some sensitive system resources, including control registers, page table, VMCS and EPT, introducing worlds switching overhead using Switch Gate.

In order to assess the effectiveness of all aspects of HyperPS, we conduct a set of experiments to evaluate the performance impact imposed by HyperPS against an original KVM system (the baseline). We run two groups of experiments, compare the performance overhead including benchmarks performance overhead and VM load time. For simplicity, we only present the performance evaluation on a server with 64 cores and 32 GB memory, running at 2.0 GHz and guest VM with 2 virtual cores. The version of the hypervisor and guest VM is 3.10.1. Different experiments are based on different numbers of guest VMs with different memory size. This is described later in all kinds of tests. Both the original hypervisor and HyperPS systems have the same configuration except the protection supported by HyperPS. The deviation of these experiments is insignificant. All the experiments are replicated fifty times and the average results are reported here.

TABLE III
EXECUTION TIME OF VM OPERATION(S).

Test Case	*VM Create*	*VM Destroy*
No_HyperPS	10.54 s	2.34 s
With_HyperPS	11.49 s	2.50 s
Efficiency	1.09	1.07

Benchmarks Performance. In order to obtain the impact of HyperPS on the whole system, we measure HyperPS with microbenchmarks and application benchmarks. We use guest VMs with 1 GB memory size.

To better understand the factor causing the performance overhead, we experiment with compute-bound benchmark (SPEC CPU2006 suite) and one I/O-bound benchmark (Bonnie++) running upon original KVM and HyperPS in Linux VMs. Figure 4 describes the results of SPEC CPU 2006 and Bonnie++ about the original KVM and HyperPS. The first 12 columns describe the results of SPEC CPU 2006 suite about Integer test and floating point test, and the last 3 columns describe the results of Bonnie++ about sequential read, write and random access. Most of the SPEC CPU2006 benchmarks (the first twelve groups) show less than 4.8% performance overhead. It's not surprising as there are few OS interactions and these tests are compute-bound. Mcf, astar, and xalancbmk with the highest performance loss allocate lots of memory. HyperPS verifies the legality of EPT when EPT updates. This can incur worlds switching which involves controlling register access and incur VM Exit which involves EPT updating. For Bonnie++, we choose a 1000 MB file to perform the sequential read, write and random access. The performance loss of sequential read, write and random access described in Figure 4 is 1.6%, 2.4% and 4%, not high, the main reason is that HyperPS has no extra memory operations for I/O data. The performance result shows that HyperPS introduces trivial switch overhead of two worlds and trivial overhead of monitoring hypervisor.

VM Loading. VM load time is a critical aspect of performance to be considered because it influences user experience. Experiments are done with 3 VMs, each guest VM is with different memory sizes from 512MB to 2GB. As expected, the VM booting time in HyperPS increases as the memory sizes increase, and the growth amplitude is larger due to the worlds switch and VMCS, EPT access. In addition, we measure the impact of completely booting and shutting down a VM (configured with 2 VCPU and 512MB memory). As Table III shown, the booting time is suffered a 1.09 times slowdown under HyperPS, shutdown time is suffered a 1.07 times slowdown, due to the extra overhead of worlds switching and VM-Mark table accessing in HyperPS World.

VI. Related work

We describe the related work from these three aspects, privilege separation, TCB reduction, and the same privilege-level isolation.

A. Privilege Separation

Some works[10], [11], [13], [14] pay attention to privilege separation. Pre-allocating physical resource and providing privilege separation for every VM can avoid VM cross-domain attack, and data leakage attack. Some efforts[8] and KI-Mon [4] provide monitoring for access operations to critical data by introducing extra hardware. principle to it. NOVA[11] divides hypervisor into microhypervisor and user hypervisor running in root mode, adopts an idea which is similar to fault domain isolation to guarantee privilege separation for every VM. Some of them redesign hypervisor greatly. In contrary, HyperPS adopts a feasible way to separate privilege without lots of modification of hypervisor.

B. TCB Reduction

Reducing TCB can reduce the possibility of the whole system being attacked by reducing the amount of hypervisor code, changing the organizational structure of virtualization, even reducing the interaction between VM and hypervisor. HyperLock[13] removes the KVM module from the TCB using the combination of address space isolation and software-based fault isolation (SFI). DeHype[14] further demotes the KVM module to user mode and applies the least privilege. Mycloud[5] and MycloudSEP[7], on the other hand, deprivilege the execution of the host VM in non-root mode and remove it from TCB.

C. The Same Privilege Level Isolation

Some efforts, [1], [12], [3], adopt the same privilege-level idea to achieve privilege separation and avoid performance overhead of inter-level translation. SKEE[1] is one of the most notable work to realize privilege separation on ARM's commodity processors today. However, SKEE is not commonly applicable to different levels of system software in comparison with HyperPS, such as hypervisors because it is designed for different hardware feature which is only defined in the kernel privilege level on ARM's 32-bit or 64-bit architecture. SecPod[12] creates the secure isolation environment for every VM and with the assistance of shadow page table (SPT) technology.

We neither adopt software at a higher level than the hypervisor, nor use microhypervisor. Inspired by the same privilege-level, we propose HyperPS World placed at the same privilege-level with hypervisor. HyperPS is independent on multi-platforms and practical for cloud providers.

VII. Conclusion

We introduce HyperPS, an approach that enables x86 platforms to support hypervisor monitoring based on privilege separation in every privileged software. HyperPS World, an isolated world, is designed to provide event-driven runtime monitoring for interaction between hypervisor and VM. This approach, which does not rely on microhypervisor or a higher privilege level software, has fewer changes to system and fewer requirements for types of CPU hardware device. It reflects good practicality, portability, and independence on multi-platforms. And security analysis describes monitoring for hypervisor and protection for HyperPS itself, the performance evaluation shows its efficiency by introducing negligible performance overhead. It can be implemented widely in real-world for cloud providers.

ACKMOWLEDGMENT

We thank the anonymous reviewers for their valuable comments. This work is supported by the Guangdong Province Key Area R&D Program (Grant No.2019B010137002).

References

[1] Azab, A., Swidowski, K., Bhutkar, R., Ma, J., Shen, W., Wang, R., Ning, P.: Skee: A lightweight secure kernel-level execution environment for arm. In: Network and Distributed System Security Symposium (2016)

[2] Cho, Y., Kwon, D., Yi, H., Paek, Y.: Dynamic virtual address range adjustment for intra-level privilege separation on ARM. In: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017 (2017)

[3] Deng, L., Liu, P., Xu, J., Chen, P., Zeng, Q.: Dancing with wolves: Towards practical event-driven vmm monitoring. Acm Sigplan Notices **52**(7), 83–96 (2017)

[4] Lee, H., Moon, H., Jang, D., Kim, K., Lee, J., Paek, Y., Kang, B.H.: Ki-mon: a hardware-assisted event-triggered monitoring platform for mutable kernel object. In: Usenix Conference on Security. pp. 511–526 (2013)

[5] Li, M., Zang, W., Bai, K., Yu, M., Liu, P.: Mycloud:supporting user-configured privacy protection in cloud computing. In: Computer Security Applications Conference (2013)

[6] McCune, J.M., Li, Y., Qu, N., Zhou, Z.: Trustvisor: Efficient tcb reduction and attestation. Cylab **41**(3), 143–158 (2010)

[7] Min, L., Zha, Z., Zang, W., Meng, Y., Peng, L., Bai, K.: Detangling resource management functions from the tcb in privacy-preserving virtualization (2014)

[8] Moon, H., Lee, H., Lee, J., Kim, K., Paek, Y., Kang, B.B.: Vigilare: toward snoop-based kernel integrity monitor. In: ACM Conference on Computer and Communications Security. pp. 28–37 (2012)

[9] Ren, J., Qi, Y., Dai, Y., Yu, X., Shi, Y.: Nosv: A lightweight nested-virtualization vmm for hosting high performance computing on cloud. Journal of Systems & Software **124**, 137–152 (2017)

[10] Shi, L., Wu, Y., Xia, Y., Dautenhahn, N., Chen, H., Zang, B., Guan, H., Li, J.: Deconstructing xen. In: Network and Distributed System Security Symposium (2017)

[11] Steinberg, U., Kauer, B.: Nova:a microhypervisor-based secure virtualization architecture. In: European Conference on Computer Systems, Proceedings of the European Conference on Computer Systems, EUROSYS 2010, Paris, France, April. pp. 209–222 (2010)

[12] Wang, X., Chen, Y., Wang, Z., Qi, Y., Zhou, Y.: Secpod: a framework for virtualization-based security systems. In: Usenix Conference on Usenix Technical Conference. pp. 347–360 (2015)

[13] Wang, Z., Wu, C., Grace, M., Jiang, X.: Isolating commodity hosted hypervisors with hyperlock. In: Proceedings of the 7th ACM european conference on Computer Systems. pp. 127–140 (2012)

[14] Wu, C., Wang, Z., Jiang, X.: Taming hosted hypervisors with (mostly) deprivileged execution. In: 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013 (2013)

[15] Zhe, W., Jin, Z., Tao, L., Shi, B., Bo, L.: Cloudauditor: A cloud auditing framework based on nested virtualization. In: IEEE International Conference on Cyber Security & Cloud Computing (2016)