HyperPS: A Hypervisor Monitoring Approach Based on Privilege Separation

***Abstract*—Once compromising the hypervisor, remote or local adversaries can easily access other customers' sensitive data of guest virtual machines (VMs). Therefore, it is essential to monitor hypervisor. Privilege separation has long been considered as a fundamental principle in software design to mitigate the potential damage of a security attack. However, previous efforts employ microhypervisor or microkernel which is not convenient for cloud providers to adopt widely. Or they employ a new software relying on higher privilege level than the hypervisor.**

As malware and attacks increase against virtually every level of privileged software including an OS and a hypervisor, we have been motivated to develop a technique, named as HyperPS, to realize hypervisor monitoring based on true privilege separation in hypervisor. HyperPS does not rely on microhypervisor or a higher privileged software. The key of HyperPS is that it decouples the functions of interaction between VM and the hypervisor. As a result, HyperPS monitors the interaction, monitors memory mapping when a page is allocated, and resists system information leakage attack. 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 compromised attacks, and the performance evaluation confirms the efficiency of HyperPS.

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

I. Introduction

In general, the Trusted Computing Base (TCB) of a typical hypervisor has to be big enough to handle resource allocation and hardware peripheral virtualization. Therefore, commodity hypervisors are already big enough to struggle with security problems. Some CVE vulnerabilities can be directly exploited to execute arbitrary code in the hypervisor. Furthermore, successful VM escape attacks, as well as the emerging hypervisor rootkits, greatly exacerbate the current situation. 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. A variety of system software such as an OS and hypervisor has a monolithic design, which integrates its 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). An exploit of any part of the system allows complete access to all memory and resources on the system. Privilege separation stemming from the work of Schroeder [8] has been considered as a fundamental principle in software design that can 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 and system intercepting by creating more privileged software or introducing microkernel, microhypervisor.

Microkernel or Microhypervisor By abandoning the monolithic design and modularizing core services, the microhypervisor and microkernel design both achieve privilege separation. The microkernel is a thin kernel [5] which can prevent damage of a fraction of the OS kernel from spreading across the whole kernel. Trustvisor [6] uses a microhypervisor to provide privilege separation and verify security sensitive workloads. Nova [10] 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 Component In order to mitigate the hazard caused by the hypervisor or kernel, plenty of solutions propose and introduce a higher privilege-level than the original hypervisor or kernel. To isolate kernel, researchers turn their focus onto an additional security layer in the system, such as a hypervisor [12], SMM of Intel [11], or TrustZone of ARM [4], which has a higher privilege over kernel. A clear advantage of this design is that it does not entail substantial modifications in the current software architecture. The drawback is that some of them are customized for hardware features. Nested virtualization [2], [16] is one of the representative approaches to isolate hypervisor, which provides a higher-privileged execution environment to run the monitor securely.

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 extra hardware. It has applicability for multi-platforms (x86, mips and so on), little modifications for original kernel and hypervisor, and practicability for cloud providers. We take x86 architecture as an example.

The key of the design is to create an isolated domain space by separating privileges, in which HyperPS can monitor runtime hypervisor. 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. 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:

- An introduction of a novel technique, HyperPS, which can implement privilege separation in a variety of system software with different processor privilege levels, such as kernel and a hypervisor.
- A practical event-driven hypervisor monitoring approach without any extra hardware or softwares on more privileged level.
- 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. When the processor runs in non-root mode, VM runs on the virtual CPU (vCPU), and when the processor runs in root mode, the hypervisor runs on the vCPU. Switching between the two modes, named context switching, means switching between VM and the hypervisor running on the vCPU.

To better support CPU virtualization, VMCS structure, a data structure based on hardware, is imported to manage transitions into and out of VMX non-root mode from VMX root mode (VM Exit and VM Entry), as well as processor behavior in VMX non-root operation. 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.

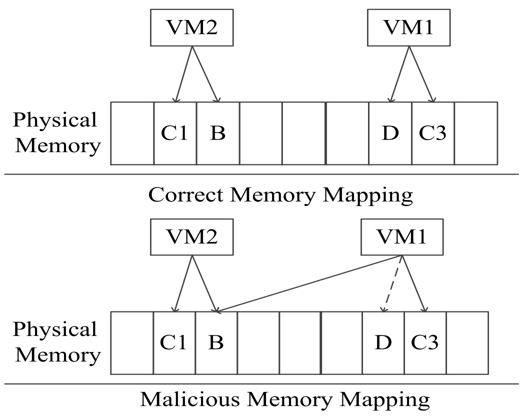

Fig. 1. The execution process of double mapping.

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 assistance to complete sensitive instruction execution operations. After VM Exit is finished, CPU needs to execute VMLAUNCH/VMRESUME instruction to initiate VM Entry and VM running. The execution process of the interaction is as follows.

1) The CPU 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 reads the Host-state filed in VMCS and loads it into CPU. 3) The CPU reads VM Exit reason in VMCS, switches from non-root mode to root mode, and jumps to VM Exit handler function entry. 4) CPU initiates VM Entry and run the VM.

The execution flow of VM Entry is opposite to that of VM Exit.

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 implement attacks based on two attack paths. 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 and double mapping attack.

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 can result in memory information leakage. There are two attack scenes, double mapping, and remapping attack.

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. For the remapping attack, there are VM1 (attacker) and VM2 (victim). 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.

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 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.

A. Architecture

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 hardware.

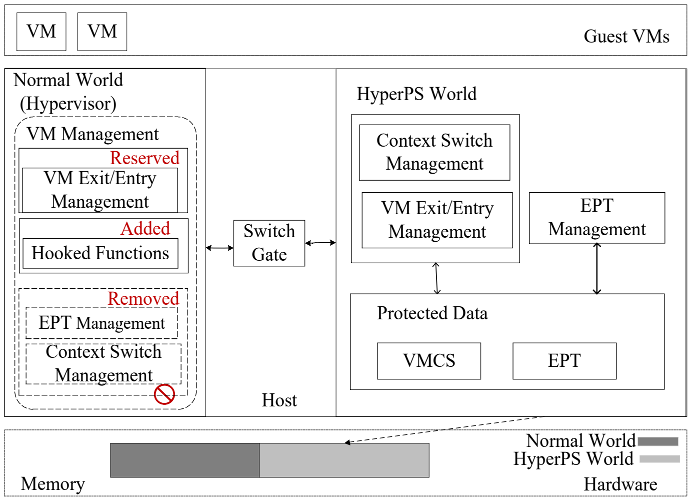

Fig. 2. The architecture of HyperPS.

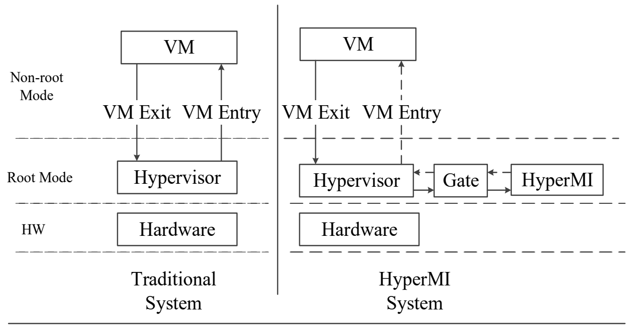

Fig. 3. Interaction comparision.

Figure 2 depicts the architecture of HyperPS. HyperPS creates two sets of different address spaces based on two sets of page tables. It is composed of three parts, Normal World, HyperPS World and Switch Gate. A pivotal condition for the success of this strategy 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, that may be used to invalidate hypervisor monitoring. 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. Operations on VMCS and EPT, such as VMCS access management, EPT management, are hooked and trapped to HyperPS World. Upon receiving such a request, HyperPS World determines whether to accept or reject it.

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. After setting up HyperPS, the whole system is ready to create HyperPS World. With these designs, HyperPS intercepts and monitors VMCS and EPT access. Furthermore, HyperPS guarantees security of interaction and address translation of VMs.

B. Data Access Paths

Some critical data structures' access paths including EPT and VMCS, called interaction flow, are changed based on changed store place since these data structures are hidden in HyperPS World. Figure 3 shows the interaction flow of HyperPS. In order to clarify the critical data access flow, hooks and interaction flow are described as follows.

Hooks We recognize 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 in x86 are unintended, and an instruction is composed of several fields. One or several of these fields may casually match another privileged instruction. 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 an attacker jumps 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 information such as registers in the instruction. 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 hypervisor integrity status.

C. HyperPS World

The creation of HyperPS World has two purposes: 1) creating a address space which can provide isolation 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 use two isolated address spaces based on two sets of page tables to achieve isolation of HyperPS World. Figure 4 describes the address space layout of two worlds through two sets of page table, the normal page table and HyperPS page table. On the left of Figure 4, the normal page table contains code and data of

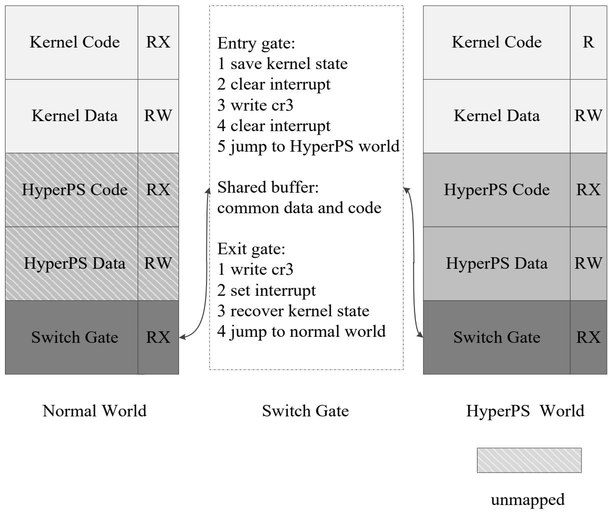

Fig. 4. An overview of address space layout.

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 can not access data in HyperPS World. On the right of Figure 4, all address is 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 can not 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 4 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 is 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) The compromised hypervisor 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) The compromised hypervisor can illegally modify the content of EPT entries and conduct remapping or double mapping attack to the VM. 3) Attackers can load EPT of any VM and access this VM's normal memory illegally.

Obviously, the direct purpose of attackers 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 hypervisor. So hypervisor requests HyperPS to handle operations and return the corresponding result for the legal request.

VMCS Monitoring 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 access management in Figure 2 does.

During VM Exit, hypervisor needs to access VM Exit reason data of VMCS, and deals with the VM exit event. Because hypervisor absents the ability to access VMCS directly, 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. The control flow is shown as Figure 3.

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 Monitoring Some functions, EPT creating, loading, walking and destroying, need access address of EPT. It can cause system suspend if they can not access the address of EPT. In order to ensure these functions can execute normally, HyperPS places hooks on these functions, then dispatches them to HyperPS World and handles appropriately. In the meantime, HyperPS avoids double mapping attack to ensure that there is only one virtual address mapping to one physical memory page during EPT updating, and handles remapping problems to ensure the content of page cleaned after page is swapped out.

If EPT isolation among VMs can not be guaranteed, a malicious VM can load other VM's EPT and access the memory data. 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 HyperPS relies on HyperPS World to provide a secure execution environment. 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 access the value of privilege registers, such as CR0, CR3 and so on. 4) Attackers can carry out DMA attack to access HyperPS World casually. We detail the protection measures for these four types of attacks below.

Protecting Page Table There are three reasons for controlling the two sets of kernel 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 interaction monitoring of HyperPS. 3) To break HyperPS World, malicious kernel code 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 memory regions of the page tables unwritable. This is necessary to prevent the page tables from being modified by attackers. 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 a 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 4 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 Privilege 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 privileged registers. 1) Malicious attackers can close DEP mechanism by writing CR0, close SMEP mechanism by writing 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 the attack, 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 an 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 breaking, 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.

TABLE II
Hypervisor Attacks Against HyperPS.

Attack	*Description*
Interaction-data 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

Modifying VMCS Attack We clarify interaction data including VMCS in Section IV-D. To prevent interaction-data leakage, we protect VMCS from attackers. Firstly, VMCS is hidden in HyperPS World and can not 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 can not 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. Table II lists all attack cases we used. The attack, named Interaction-data attack, tries to tamper the Guest_CR3 field in VMCS. This attack fails because it can not access VMCS. According to the above analysis, attackers can not access VMCS, and can not conduct further attacks. Therefore, the VM interaction data can be protected by HyperPS.

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.

We implement a real attack, 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 can not 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 is created by relying on page tables. We analyze the protection of HyperPS World from four aspects, page table modification attack, hooks redirection attack, reg 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 World 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

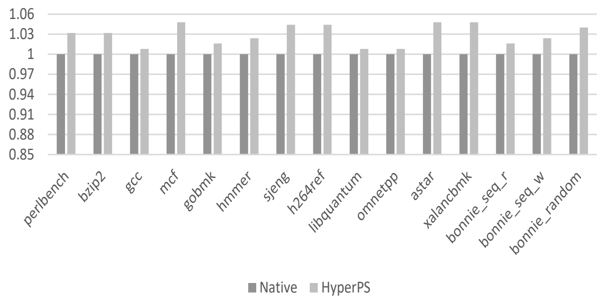

Fig. 5. Performance overhead.

executable permissions in case of attacking running processes 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 VMCS operations, EPT operations and control register access operations is 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 can not redirect hooked functions and bypass monitoring.

3) Reg Modification Attack: Some registers access operations including CR0, CR3, CR4, are controlled and hooked to HyperPS World. CR0 register can control the W$\oplus$X privilege of code, CR3 can control the loading of the page table and CR4 can decide SMEP mechanism. Protection for page table, hooked functions and regs, 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. 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 host OS 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.

TABLE III
EXECUTION TIME OF VM OPERATION(S).

Test Case	*VM Create*	*VM Destroy*
No_HyperPS	11.79 s	1.75 s
With_HyperPS	12.97 s	1.89 s
Efficiency	1.1	1.08

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. 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.

Benchmarks Performance In order to obtain the impact of HyperPS on the whole system, we measure HyperPS with microbenchmarks and application benchmarks. We use one guest VM 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 a Linux VM. The experiment result described in Figure 5 (the last three groups) shows a relatively low cost. 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 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 memory isolation of VMs.

VM Load Time The load time of a VM is a critical aspect of performance to be considered because it influences user experience. We design experiments to evaluate the performance impact of HyperPS for VM loading. We measure the impact of completely booting and shutdowning a VM (configured with 2 VCPU and 512MB memory). As Table III shown, the booting time is suffered a 1.1 times slowdown under HyperPS, shutdown time is suffered a 1.08 times slowdown, due to the extra overhead of worlds switching and VM-Mark table accessing in HyperPS World. Such overhead is worth for HyperPS.

VI. Related work

We describe the related work from these three aspects, reconstructed hypervisor, customized hardware, and the same privilege-level isolation.

A. Privilege Separation

Some works [9], [10], [14], [15] pay attention to privilege separation. Pre-allocating physical resource and privilege separation for every VM can avoid VM cross-domain attack, and data leakage attack. DeHype [15] further demotes the KVM module to user mode and applies the least privilege principle to it. NOVA [10] divides hypervisor into micro-hypervisor 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. Integrity Verification for Hypervisor

In order to ensure the security of the hypervisor during trusted boot and runtime, an effective and commonly used method is to verify the integrity of the hypervisor, and reduce the attack surface. For the security of the hypervisor during trusted boot, paper [7] proposes control flow integrity protection policy, by verifying regularly control flow integrity behavior to detect rootkit attacks. However, attacker can detect the regular and bypass the detection. For runtime security of the hypervisor, HyperSafe [13] and HyperCheck [11] choose pooling-query method based on SMM to finish integrity verification of hypervisor. And attackers can hide trace during polling-query intervals when comparing to event-driven monitoring.

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] can only be applied to limited levels of system software in comparison with HyperPS. It is one of the most notable work to realize privilege separation on ARM's commodity processors today. SecPod [12], an extensible approach for virtualization-based security systems that can provide both strong isolation and the compatibility with modern hardware. The biggest difference between SecPod and HyperPS is that SecPod creates the secure isolation environment for every VM. SecPod solves the problem of VM address mapping with the assistance of shadow page table (SPT) technology.

We neither adopt software at a higher level than the hypervisor, nor use micro-system. 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 a secure isolated execution environment at the same privilege-level with hypervisor. The environment is designed to provide memory isolation protection for VMs, and event-driven runtime monitoring for interaction between hypervisor and VM. This approach, which does not rely on additional hardware devices 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 protection for VM and HyperPS itself, the performance evaluation shows its efficiency by introducing negligible performance overhead. It can be implemented widely in real-world for cloud providers.

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] Ben-Yehuda, M., Day, M., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., Yassour, B.A., Ben-Yehuda, M.: The turtles project: Design and implementation of nested virtualization. Yehuda pp. 1–6 (2007)

[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] Ge, X., Vijayakumar, H., Jaeger, T.: Sprobes: Enforcing kernel code integrity on the trustzone architecture. Computer Science **25**(6), 1793–1795 (2014)

[5] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Kai, E., Kolanski, R., Norrish, M.: Sel4: Formal verification of an os kernel. In: Acm Sigops Symposium on Operating Systems Principles (2009)

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

[7] Petroni, N.L., Hicks, M.: Automated detection of persistent kernel control-flow attacks. In: ACM Conference on Computer and Communications Security. pp. 103–115 (2007)

[8] Saltzer J, S.M.: The protection of information in computer systems. Proceedings of the IEEE **63**(9), 1278–1308 (1975)

[9] 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)

[10] 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)

[11] Wang, J., Stavrou, A., Ghosh, A.: Hypercheck: a hardware-assisted integrity monitor. In: International Conference on Recent Advances in Intrusion Detection. pp. 158–177 (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., Jiang, X.: Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: Security and Privacy. pp. 380–395 (2010)

[14] 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)

[15] 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)

[16] Zhang, F., Chen, J., Chen, H., Zang, B.: Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: ACM Symposium on Operating Systems Principles. pp. 203–216 (2011)