HypeBIOS: Enforcing VM Isolation with Minimized and Decomposed Cloud TCB

Yulong Zhang
Virginia Commonwealth University
zhangy44@vcu.edu

Wuqiong Pan
Virginia Commonwealth University
wpan@vcu.edu

Qingpei Wang
Coreboot Community
wangqingpei@gmail.com

Kun Bai
IBM T.J. Watson Research Center
kunbai@us.ibm.com

Meng Yu
Virginia Commonwealth University
myu@vcu.edu

Abstract

Virtualization has made cloud computing a popular trend by virtue of its elastic "data anywhere" and "computing anywhere". However, traditional virtualization architectures usually have three drawbacks: 1) being vulnerable to many known attacks targeting at the large software stacks; 2) endowing too much power to cloud providers, who can fully control the Virtual Machine Monitor (VMM) and the management Virtual Machine (VM); and 3) lacking trusted isolation between VMs.

In this paper, we propose HypeBIOS to provide isolation of VMs based on a verifiable thin virtualization Trusted Computing Base (TCB). Unlike the traditional architectures, HypeBIOS excludes unnecessary initialization components in the boot chain and shifts the control VM (management VM) out of the TCB. The reduced TCB is further decomposed into two layers. The master layer works in the System Management Mode (SMM) and contains crucial handlers. The slave layer resides in the legacy virtualization host mode to cooperate with the master layer. We build a prototype of HypeBIOS on the x86 platform. The experiments show that HypeBIOS only introduces moderate overhead.

Categories and Subject Descriptors D.4.6 [*Operating Systems*]: Security and Protection; D.4.7 [*Operating Systems*]: Organization and Design; D.4.8 [*Operating Systems*]: Performance

General Terms Design, Security, Performance

Keywords Virtualization, TCB Minimization, Decomposition, Isolation

1. Introduction

Cloud computing is becoming a major trend due to its inspiring features of elastic "data anywhere" and "computing anywhere". Among the cloud services, Infrastructure-as-a-Service (IaaS) is the most fundamental one, where the cloud user owns a virtual machine (VM) and purchases virtual power to execute as needed, just like running a virtual server. A typical example of commercial IaaS is the Amazon Elastic Compute Cloud [11] based on the Xen architecture [39].

Cloud providers usually prefer the off-the-shelf virtualization solutions to ease deployment and to lower costs. However, these traditional virtualization infrastructures usually have several deficiencies. First, they are vulnerable to many known attacks targeting at the large software stacks. Such as the breaking out of VMWare [22] and Xen [36]. Second, they endow too much power to cloud providers, who can fully control the Virtual Machine Monitor (VMM) and the management VM. Typically a cloud provider with great vendor reputation and viability would not intend to filch users' secret, as long as it has a rational business foresight. However, there exist "insider attacks" from the individual employees of the cloud provider [23]. Finally, the traditional architectures lack trusted isolation between VMs. Because the isolation mechanism is often implemented in the large software stack and at the same time under the direct control of the cloud providers, it is prone to VMM vulnerability exploitations and insider attacks described above.

Consequently, despite of the attractive advantages of cloud computing, many privacy-concerning users still avoid outsourcing their data and computation to cloud platforms. In fact, some laws even restrict a business' freedom to outsource their sensitive computing to cloud providers [13]. To counteract the emerging threats, we propose the Hype-

BIOS architecture. The primary goal is to construct a minimized and decomposed cloud TCB from scratch and thus enable more trusted VM protection, without losing backward-compatibility, efficiency and flexibility. The main contributions of this work include:

- We minimize the virtualization TCB by removing redundant procedures from the boot chain. The legacy boot chain contains many repeated initialization processes. By cutting off them and integrating the whole virtualization initialization procedure in BIOS, HypeBIOS reduces the cloud TCB by an order of magnitude (our prototype has ∼4K LOCs).
- We decompose the minimized TCB into two layers. The master layer works in SMM and contains crucial handlers. The slave layer remains in the legacy virtualization host mode to cooperate with the master layer. TCB decomposition further improves the protection of the crucial components and only exposes the thin slave layer (about 1K LOCs) as the attacking surface, which eases verification and attestation.
- We implement a prototype on x86 platform which has enough flexibility and is easy to keep updated. Despite the integration of initialization code into BIOS, we do not program the whole VMM into it. After platform initialization, HypeBIOS will load master/slave layer handlers from hard drive and relocate them in different areas. Thus software upgrade does not need re-flashing BIOS and remains flexible.
- We show that the minimization and decomposition design has good backward compatibility and introduces moderate performance overhead. With the aid of CPU, memory and I/O virtualization extensions, unmodified HVM VMs can be launched on HypeBIOS without significant performance downgrade.

The rest of the paper is organized as follows. The related work and background information is discussed in Section 2 and 3. In Section 4, threat model and assumptions are described. In Section 5 we discuss the design details, and in Section 6 we present the implementation and performance evaluations. We further provide some security analysis in Section 7 and describe our future work in Section 8. The paper is concluded in Section 9.

2. Related Work

2.1 Cloud TCB Minimization

The first type of cloud TCB minimization is through VMM simplification. Designed specially for system execution tracing and malware analysis, MAVMM [31] eliminates unnecessary virtualization features commonly found in general purpose VMMs. Trustvisor [26] is also a specialized VMM minimizing the TCB so that it can be used for isolation purposes only. Although the simplified VMMs can be extremely thin (MAVMM contains only ∼3.2K LOCs and Trustvisor has ∼2K LOCs for its core functions), they can only handle certain application schemes and neither of the two architectures supports multiple VMs. So they are unsuitable to be the cloud VMM serving general purposes.

The second type of approaches makes use of hardware features to minimize TCB size. SICE [6] makes use of x86 System Management Mode (SMM) to enforce strong TCB isolation. The security of the isolated environments is guaranteed by a TCB consists only the hardware, the BIOS, and SMM program of ∼300 LOCs. However, SICE can only provide one isolation zone and supports only one VM at one time, therefore unable to be adopted as a cloud platform solution. Flicker [27] is a technique leveraging the secure-execution features of CPUs to provide application level privacy protection based on a thin and secured TCB. Nevertheless, it only offers app-level protection and cannot be utilized to secure the VMs on cloud.

NoHype [20], as the third type of TCB minimization, dynamically eliminates the VMM layer after booting the guest VMs. The shortage of NoHype is that it has the rigid requirement of one-VM-per-core on multi-core processors, and it requires pre-allocated nested page table. The two requirements restrict the number of VMs that can be simultaneously hosted on the physical platform and decrease the elastic of cloud computing. They increase the cost of cloud providers as well. Moreover, NoHype needs to modify the guest kernel to get rid of memory access outside the region indicated by the pre-allocated nested page table.

Coreboot [9, 10] can be treated as a promising way for TCB minimization. It aims at replacing the proprietary BIOS firmware with a lightweight system designed to perform only the minimum of initializing tasks, and can directly boot an ELF image included in the ROM. However coreboot is just a BIOS design and has no virtualization component.

2.2 Cloud TCB Decomposition

There are some approaches directly dividing VMM into separated components with different privileges. NOVA [35] constructs a microkernel-based VMM that is ∼9K LOCs in size. Despite its thin TCB compared to commodity hypervisors, the complexity of TCB is not markedly decreased since the microhypervisor is still in charge of complex management duties like address space management, interrupt and exception handling, and communication between the running workloads. So the thin TCB is still difficult to be secured and verified dynamically. Although seL4 [21] proposes a technique to formally verify a microkernel with ∼8.7K LOCs, it imposes several restrictions on the microkernel functionality. Thus a verifiable microhypervisor remains impractical so far. Other than microkernel-based systems, VMM Disaggregation [30] also shrinks the TCB size by moving some VMM components out of the privileged domain. However, the TCB size is still too large for dynamic protection.

There are also some approaches realizing decomposition with the help of nested virtualization. SplitVisor [32] splits the traditional VMM into two components according to their functions: a smaller one as the minimized TCB to enforce isolation, and a larger one to provide service functionalities. The problem with SplitVisor is that it requires specialized guest VMMs, which should be uploaded by cloud users. Similar to SplitVisor, CloudVisor [40] utilizes nested virtualization to separate TCB from the computing software stacks. As the host VMM, CloudVisor simply forwards (and verifies) all the operation and data flows between the guest VMM and the VMs. But in practice, the operation and data flows between the guest VMM and its VMs could be very complicated thus CoudVisor has to allow guest VMM to directly operate with VM's sensitive components (e.g. Instruction Pointer (IP) relocation after VMEXIT handling) . In this sense, the guest VMM should be considered as a part of the TCB.

3. Background

3.1 Root of Trust Measurement

The Root of Trust Measurement (RTM) mechanism is commonly used to ensure the integrity of TCB, which mainly relies on the Trusted Platform Module (TPM) chip. Specified by the Trusted Computing Group (TCG), the TPM chip can be used to authenticate hardware devices [25]. It can be found on almost all the motherboards of servers and high-end PCs. A unique and secret RSA Endorsement Key (EK) is generated for each TPM at the time of manufacture and will be permanently sealed inside the chip, and other sensitive data will be stored into shielded memory. A Privacy CA (Certificate Agency) can authenticate a TPM according to its public Endorsement Key. The main role of TPM chips in trusted computing is to act as the Core Root of Trust for Measurement (CRTM), which measures the integrity metrics of modules, holds them in Platform Configuration Registers (PCRs) and reports them in an authenticated way in remote attestation. For privacy concerns, EK is not allowed to be used as platform identity directly. Instead, Application Identity Keys (AIKs) are created to sign these PCR values. A detailed example to establish TCB with TPM can be found in Terra model [12]. Note that RTM can be either Static or Dynamic (SRTM and DRTM, respectively) [18].

3.2 System Management Mode

Both Intel and AMD CPUs support the System Management Mode (SMM) as one of its operating modes, to handle power management and other duties. The application of SMM in TCB protection has gradually become popular [5, 6, 24]. Upon an System Management Interrupt (SMI), the processor saves its state to a dedicated state save map and switches to the SMM. As an alternative of sending SMI by software interrupt, a southbridge timer [2] can trigger SMM automatically and periodically. To return from the SMM, the special instruction RSM restores the saved processor state and resumes normal execution.

SMM code is loaded by BIOS and is stored in a designated memory called SMRAM. To provide protection of the SMM code and data, both AMD and Intel provide the capability of locking the SMRAM. When the SMRAM is locked, all accesses to it, except from within the SMM, are prohibited. All interrupts, including non-maskable ones, are disabled upon entering the SMM. Thus, no other code running on the system can interfere with the SMI handler. Current hardware can support up to 4 GB of SMRAM.

Note that in Intel's manual for system developers [16], a **Dual-Monitor Mode** is described, which has an "Executive Monitor" and an "SMM Monitor". This mode is seemingly similar to the double-layer architecture in this paper. However, Dual-Monitor Mode is no more than a complement of Intel VTx (by serving SMI for VTx VMs). The design purpose and triggering mechanism are actually different from our work.

4. Threat Model and Assumptions

4.1 Threat Model

HypeBIOS aims at defending against all malicious activities trying to (locally or remotely) compromise the VMM and break into users' privacy.

We assume that once compromised, the root software stack could be fully controlled by the adversaries. Thus the adversaries are able to do the following types of attacks: 1) taking over the control of management/control VM of the cloud provider and then looking into VMs' memory and disk; 2) breaking into the management/control VM of the cloud provider and injecting malicious code into the VMs; 3) directly exploiting the vulnerabilities of the VMM (the legacy root software stack with bulk size) and dumping secrets from VMs; 4) as an internal employee, launching insider-attacks (not physically) to leverage the powerful management interfaces to steal information.

We do not take internal physical attacks (for example [4]) into consideration, because defending hardware attacks is out of the scope of our work. Some physical attacks require special tools like microprobing needles [4]. Even if special tools are not needed, physical attacks may leave evidences physically (camera videos or scratches). We assume that the cloud providers have no motivation to launch such attacks.

4.2 Assumptions

First of all, we take the attacks from insiders into consideration but we assume the overall cloud provider entity is benign and credible. Second, the platform is physically secure (e.g. monitored by cameras, locked in a safe room, etc.) so that attacks via physical accesses are not possible. Third, the platform is equipped with trusted computing hardware, including the Core Root of Trust Measurement (CRTM) and Trusted Platform Module (TPM), which allows the attesta-

tion to the integrity of the crucial components in TCB. And finally, SMRAM cache should not be poisoned. On AMD platforms, SMRAM cache has already been protected so that cache-poisoning attacks [37] are immunized. On Intel platforms, however, a proper configuration of the System Management Range Register (SMRR) [16] is required to ensure this assumption.

5. HypeBIOS Design

5.1 Design Goals

The primary goal of HypeBIOS is to improve the traditional cloud architectures by 1) reducing TCB size and decomposing it to provide special protection to the crucial components, 2) enforcing VM isolation based on the minimized and decomposed TCB, and 3) without losing backward-compatibility, efficiency and flexibility. The detailed design considerations are listed below.

TCB Minimization and Decomposition The large size and high complexity of security-sensitive applications and systems software is a primary cause of poor testability and high vulnerability [34]. Hence the TCB size of the cloud architecture with HypeBIOS should be as small as possible. However, a small/simple TCB is not sufficient. There should be a strong protection mechanism to enforce the security of the TCB. Formal analysis [21] is usually used to verify the TCB correctness and security properties, and software model checking [19] can be utilized to verify the implementation. There are also approaches utilizing hardware based dynamic measurement to secure TCB, like TrustVisor [26] and Flicker [27] . All the above protection mechanisms, however, have restrictions on the TCB size. For example the recent successful report of formal verification shows the capability of a general-purpose kernel with ∼8.7K LOCs [21]. Therefore, we should further decompose the minimized TCB and only apply protection on the crucial components.

Weakening the Cloud Provider's Power In order to alleviate the concerns of privacy leakage to cloud provider's internal employees, the over-powerfulness of cloud providers should be dealt with. In the current cloud architecture, Xen [39] for example, the cloud provider occupies the most privileged domain and handles all the operations with the authority to look into users' data and computation. In HypeBIOS based architecture, the power of cloud provider should be limited, as long as it can normally perform cloud resources management (allocation, revoking and migration).

VM Level Isolation We choose the isolation granularity at the VM level with the same reasons listed by CloudVisor [40]. First, most of the current commercial public clouds provide the service in the IaaS fashion (e.g. Amazon EC2 [11]). Second, VM is a native and simple abstraction/encapsulation of privacy for each cloud user. Unlike protecting processes, protecting VMs does not require handling the complex and subtle semantic gaps. And third, protection at the VM level is more likely to preserve backward-compatibility, without the need of modifying OS kernels and applications.

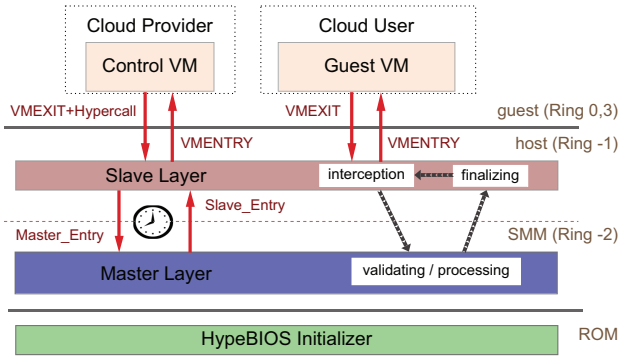


Figure 1: HypeBIOS contains an initializer and a double-layer VMM. Traditional initialization process done by BIOS, bootloader and VMM is retrenched and merged into the initializer. After initializing the virtualization environment, the initializer will load the VMM handlers either from ROM or disk. The master layer will be put in SMRAM while the slave layer will be located in normal RAM area. The transition between the double layers is enforced by a southbridge timer.

Backward-compatibility, Efficiency and Flexibility A good cloud system solution should require no modification from cloud users and introduce as less performance overhead as possible. Besides, if the functionality of the platform needs to be changed, the software components should be easy to be updated accordingly. No cloud provider would accept a solution that cannot be flexibly upgraded. That is why we should not seal the whole TCB into the substrate or the BIOS, which will be discussed in Section 5.3.

5.2 Overview of HypeBIOS Architecture

The architecture of HypeBIOS is shown in Figure 1. Once the processor virtualization extension (SVM [3] for AMD and VMX [16] for Intel) is enabled, CPU will switch between two modes - guest and host. All the cloud users' VMs are executed in the guest mode. The VM kernels reside in Ring 0 and applications run in Ring 3. All sensitive instructions will trigger VMEXITs, which should be taken care of by VMM handlers in the host mode (Ring -1). From a guest VM's view, the execution environment has no significant difference compared with a physical platform.

HypeBIOS differs from the traditional virtualization architectures in two aspects. First, HypeBIOS integrates all the platform initialization code in the BIOS ROM, just like LinuxBIOS [9]. Traditionally, after power-on, BIOS has to create interrupt tables, prepare the e820 tables, set up PIC/-timer/MTRR, initialize PCI devices, probe SMP, and configure the VGA display, etc. This cumbersome process has to be gone through later by the bootloader and the VMM again. Although the data structures and the content inside may be different (for example, BIOS uses Interrupt Vec-

tor Table (IVT) while bootloader and VMM use Interrupt Descriptor Table (IDT)), the initialization code is quite the same. Thus HypeBIOS choose to avoid the repetitive operations. On power-on, following some necessary hardware initialization, the BIOS ROM directly transforms processors into protected mode with paging, and enables virtualization extensions. Afterwards the HypeBIOS initializer loads the VMM handlers either from ROM or from hard drives (for easy-to-upgrade purpose) into RAM and passes the control to those handlers.

The second difference is that HypeBIOS relocates VMM handlers in different memory areas. A double-layer VMM, consisting of the slave layer in host mode and the Master Layer in SMM, replaces the legacy VMM. The master layer will be put in SMRAM while the slave layer will be located in normal RAM area.

5.3 HypeBIOS Initializer

Some existing approaches adopt Dynamic Root of Trust Measurement (DRTM) to late launch the VMM (e.g. Cloudvisor [40]). In this way, platform initialization code can be removed from TCB. However, DRTM can only ensure the integrity of VMM during the launching process, and cannot protect the VMM handlers in a real-time fashion. While formal verification and model checking methods [19, 21] can help to dynamically verify the TCB, but they have to consume computing power and impose several restrictions on the target TCB.

HypeBIOS makes use of the native protection mechanism provided by the hardware, SMRAM, to secure its crucial handlers. The initialization of SMRAM requires the trust of BIOS, thus the Static Root of Trust Measurement (SRTM) is needed. To our knowledge, most of the other SRTM-based approaches assume that the attacker is unable to violate the integrity of the booting process (e.g. NOVA [35] explicitly claims so). However, the boot chain nowadays can be complex and repetitive. For example, GRUB 2 [14] nowadays contains $\sim$200K LOCs in total and even GRUB Legacy contains $\sim$10K LOCs as its core components. Some of the initialization duties of the bootloader have already be carried out by the BIOS and will later be covered again by the VMM. It can largely decrease the TCB size if we merge them together into the BIOS and remove unnecessary boot procedures.

The boot process of HypeBIOS is as follows:

ROM stage: On power-on or reset, CPU fetches the instruction at 0xFFFFFFF0. At this time, RAM has not been initialized so HypeBIOS has to execute using cache-as-RAM. This part is the same with the coreboot [10] boot block. After some basic CPU initialization operations, HypeBIOS initializes the Front End Bus (FSB) and brings the RAM online. Then the initializer probes the PCI bridges and buses and enables all possible devices.

Protected mode and paging enabling: After the ROM stage HypeBIOS configures the control registers (CR0, CR3 and CR4) to enable protected mode and paging. The reason of enabling protected mode is that virtualization related instructions are only valid in this mode, and the reason to enabling paging is that nested paging requires host paging. The tricky part is that legacy BIOS interrupt handlers work in real mode, so mode transition is needed to call software interrupts before GDT and IDT are initialized.

Basic platform configuration: Now it is time to do peripheral configurations, including PIC, timer, MTRR and PCI devices. The southbridge should be carefully configured so that SMM can work properly later with timer triggering. The details will be introduced in Section 5.4. TPM chip should also be initialized in this stage.

APIC and virtualization enabling: This is the most import step for HypeBIOS. The processor virtualization extension (SVM/VMX) is enabled. Specially, HypeBIOS creates a nested page table and a page-access-control table. The page-access-control table marks the owner (VM) and access attributes (R/W) for every page. IOAPIC as well as LAPIC are also configured in this step.

SMP preparation: The above procedures are handled by the Boot Strap Processor. To enable the full function of the cloud platform, the BSP needs to further bring other cores/processors online and enable protected mode/paging/virtualization for them.

Other initializing work: This part contains VGA/serial console initialization and other hardware preparation work. For conciseness the details are omitted.

Loading double-layer handlers: Finally, the platform is ready for the VMM handlers. HypeBIOS loads them either from the ROM or the disk and measures the integrity of them. The measurement value is extended to the TPM Platform Configuration Register (PCR). After that, the master layer is relocated into SMRAM and the slave layer is relocated in normal memory area. In the end, SMRAM should be set as locked to avoid tampering, and the platform is handed over to the double-layer VMM handlers.

Because the initializing code in the ROM is executed only once during the boot, and service-oriented changes only impact the VMM handlers, which can be loaded from disk, the integration of BIOS, bootloader and VMM initializer as above would not hurt the flexibility to upgrade HypeBIOS.

After the integration, the boot chain is significantly retrenched. As shown in Figure 2, the TCB size of HypeBIOS is extremely thin compared to other approaches. We note that integration is not the only reason of HypeBIOS's extremely thin TCB. The utilization of hardware I/O virtualization support also contribute to TCB minimization, which will be discussed in Section 5.6.

5.4 The Decomposed Double-layer VMM

Although the integration of HypeBIOS initializer has significantly reduced the TCB size, an effective and efficient mechanism is still needed to dynamically ensure the security of the TCB. Thus we further decompose the TCB into two lay-

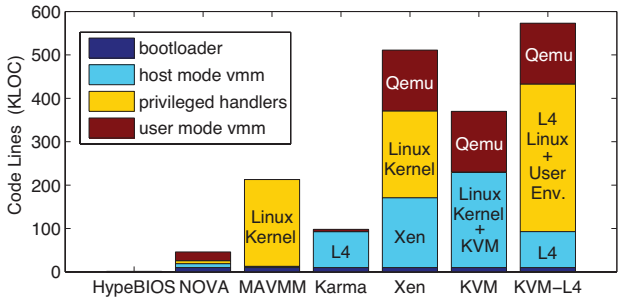


Figure 2: TCB size comparison of some virtualization architectures relying on trusted boot chain. Since all of them contain the BIOS ROM in TCB, the code lines of BIOS are omitted in the figure. Taking advantages of initializer integration and using only virtualized I/O devices, HypeBIOS has only ∼4K LOCs in memory (∼3K LOCs are protected by SMRAM).

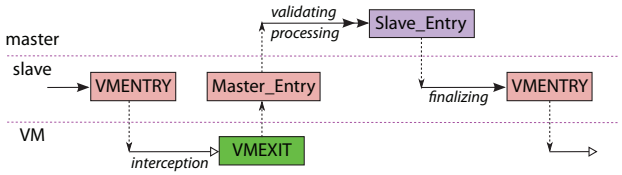


Figure 3: CPU execution and transition flow of HypeBIOS.

ers, a master layer and a slave layer. The slave layer replaces the legacy VMM to intercept the execution of the VMs, and passes the crucial VMEXIT handling (such as nested page fault) to the master layer, as shown in Figure 1 and Figure 3. On receiving the requests, the master layer firstly checks if the VMEXIT violates the access control rules as specified in the page-access-control table (Figure 4), and then processes it. If no error detected, the master layer will return control back to the slave layer.

Figure 4 is an example of the page-access-control table. S_n in the figure means "shared to VM$_n$" and P_n means "privacy of VM$_n$". The areas of the master and slave layers are marked as SMRAM and P (global privacy) respectively. The region of initializer (BIOS) is marked as read-only.

It is worth noting that the transition from the slave layer to the master layer is not though calling software SMI. Once an SMI is sent, all CPU cores will be forced into SMM, which could be a fuse of Deny-of-Service (DOS) attacks. Besides, it costs CPU powers to call SMI frequently. Consequently, HypeBIOS configures the southbridge to mask the software SMI and trigger SMM by a southbridge timer [2]. With the help of this timer, the SMM is triggered periodically, and the system control is automatically passed from the slave layer to master layer (Figure 1). Ideally, master layer processes all the VMEXITs, but for performance optimization, not all of the VMEXITs should be passed into SMM to process. Only those suspected as privacy-breaching (e.g. memory probing from one VM to another) are necessarily delivered to the master layer. The interval of the timer-based transition

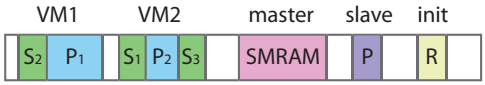


Figure 4: Page-access-control table maintained by the master layer. Each entry is corresponding to a page in memory.

will be discussed in Section 6, which slightly impacts the performance.

Residing in the SMRAM, the master layer (∼3K LOCs) is well protected and cannot be accessed from the software stacks in the protected mode. The only interface to access the master layer is via the slave layer. So the slave layer (∼ 1K LOCs) is the only TCB component that should be verified online. The idea of TCB decomposition thus makes the thin TCB even easier to perform integrity checking (a recent successful report of formal verification shows the capability of a general-purpose kernel with ∼8.7K LOCs [21] .

5.5 Secure Boot

On system power-on or reset, the TPM chip is initialized. It then measures the Core Root of Trust for Measurement (CRTM) in BIOS boot block, which will further measures the SMM handlers (HypeBIOS master layer) that will be written into SMRAM. All the measurement results will be extended into the Platform Configuration Register (PCR). Then the initializer copies HypeBIOS master layer code into the SMRAM and temporarily trigger SMM by calling SMI. The boot code of master layer will generate a fresh asymmetric key-pair (K^{Pub}_{SMM}, K^{Pri}_{SMM}), and extend the public key information into the PCR. Later all the cloud users can easily obtain this public key from TPM. The private key is kept in SMRAM and will never be touched from outside. Afterwards, the master layer will resume to protected mode and pass the control back to HypeBIOS initializer and continue the normal boot procedure, getting the slave layer loaded.

All the users on the platform are able to request for attestation at any time. Because the platform is shared by multiple tenants, the TPM architecture can be realized using the vTPM fashion [8]. The attestation procedure of the platform is described as follows:

(1) On receipt of a request for attestation challenge, the TPM chip will respond with the public Attestation Identity Key (AIK^{Pub}) and the certified public Endorsement Key (EK^{Pub}). The challengers can further validate the certification of the EK^{Pub} by forwarding the keys to the Privacy CA [25].

(2) If the certification of the EK^{Pub} is valid, the Privacy CA signs a certificate for AIK^{Pub}, and encrypts the certificate with a newly created session key SK$_{PCA}$. Along with them, SK$_{PCA}$ and AIK^{Pub}, encrypted by EK^{Pub}, are altogether sent back to the challenger. The challenger then deliver the second blob to the TPM chip.

(3) Once received them, the HypeBIOS TPM decrypts the SK$_{PCA}$ and AIK^{Pub} using its EK^{Pri}, and checks if the

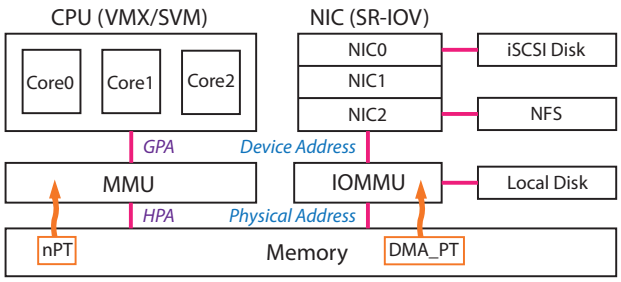


Figure 5: Memory and I/O management in HypeBIOS.

AIK^{Pub} matches with the one it owns. If everything goes well, the HypeBIOS releases the session key SK$_{PCA}$.

(4) With the session key, the challenger can obtain the signed certificate of AIK^{Pub}. Now that AIK^{Pub} is authenticated, the user can generate a random number n to perform the real-time platform attestation by asking for the signed PCR value along with this one-time random number.

(5) The TPM chip should reply with the Stored Measurement Log (SML, containing the BIOS, MBR and VMM measurement fingerprints as well as the public key of SMM handlers, K$^{Pub}_{SMM}$), along with the PCR value and the random number signed by AIK^{Pub}. Upon receiving them, the challenger first of all check if n is correct, and then applying the PCR's extend operation on SML (to see if it is the same with the received PCR value). If everything matches, and the platform measurement fingerprints in SML satisfying the challenger's requirement, the attestation is finished with success and the K$^{Pub}_{SMM}$ can be confidently used later in authentication or key exchange.

5.6 Memory and Device Virtualization

Both Intel and AMD have extended two-layer address translation to three-layer address translation (nested paging). The guest page table (gPT) specified by CR3 register in guest VM is responsible for translating guest virtual addresses (GVA) to guest physical addresses (GPA). A new table called nested page table (nPT) controlled by the VMM is responsible for translating GPA to machine frame number (HPA). The address of nPT is specified by nCR3, described by a VMCS/VMCB field. As shown in Figure 5, HypeBIOS (mainly the master layer) maintains the nPT and MMU will automatically translate guest physical address to machine address. Once set up, the memory translation process will be automatically done by MMU and no interaction with the virtualization software is necessary. The master layer will only be called for nPT updating when a nested page fault happens. But, again, the host physical address is guarded by the page-access-control table. The master layer will not acknowledge any memory access attempts.

Except for memory translation, I/O management is another important issue to be considered. HypeBIOS fully makes use of the hardware extensions for virtualization, including IOMMU (VTd [15] for Intel and AMD-Vi [1] for AMD) and SR-IOV [17], to minimize the TCB size. The input/output memory management unit (IOMMU) connects a DMA-capable I/O bus to the main memory. Like a traditional MMU, which translates CPU-visible virtual addresses to physical addresses, the IOMMU takes care of mapping device-visible virtual addresses (also called device addresses or I/O addresses in this context) to physical addresses. With the help of IOMMU, devices can be directly assigned to VMs. This kind of direct assignment of devices provides very fast I/O and eliminated drivers from VMM. However, it prevents the sharing of I/O devices. To solve this problem, peripheral devices start to support SR-IOV, which provides a mechanism by which a Single Root Function can appear to be multiple separate physical devices, called virtual functions (VFs). As shown in Figure 5, the Ethernet adaptor on HypeBIOS platform is configured to appear in the PCI configuration space as multiple functions. The slave layer can assign different VFs to different VMs.

For a device that does not support virtualization, like hard drives, there are two solutions. First, cloud users can mount iSCSI disks [29]. Second, HypeBIOS can redirect the disk I/O requests to the control VM, who controls the local disk. However, this solution exposes users' data to cloud provider, so I/O encryption is required for it.

5.7 Scheduling

To simplify the system design, HypeBIOS directly supports two modes of VM scheduling, one-VM-per-core or round-robin. In the case of one-VM-per-core, the situation is similar to NoHype [20]. Every VM occupies a unique CPU core and thus a unique LAPIC. The good side of doing so is that the scheduling code can be extremely simple and it is difficult for side-channel attack [33] to succeed. However, dedicating cores to VMs restricts the dynamic resource allocation in cloud computing, and increases the cost of cloud providers.

An alternative to one-VM-per-core solution is the round-robin scheduling algorithms, which is not complicated either. VMs assigned on the same core will share the CPU time equally. If a VM is in the waiting-for-VMEXIT-handling state, the scheduler will simply pass it and dispatch the next available VM.

By intercepting VMLOAD/VMRUN/VMSAVE, HypeBIOS has supported nested virtualization on AMD platforms. This indirectly introduces the third scheduling mechanism - scheduling by the guest VMM. On intercepting virtualization instructions launched by the guest VMM, HypeBIOS will make changes to corresponding VMs on behalf of it. In this way, HypeBIOS can operate with complicated scheduling algorithms without actually implementing them.

5.8 Cloud Management

Unlike the traditional architectures, the cloud provider only controls the unpriviledged control VM in HypeBIOS design. The control VM is responsible for resource manage-

ment. The management work is indirect and should be done through the interface provided by the slave layer. Currently HypeBIOS follows the design of Xen, by assigning the software interrupt 0x82 as the hypercall interface. Any requests from the control VM will be captured by the slave layer and further sanitized by the master layer. Any resources allocation changes requested by the control VM will be logged and the impacted VMs will be notified (if they install a special driver). In this way, the cloud provider cannot stealthily manipulate users' secrets.

To create a VM, HypeBIOS will allocate the resources according to the request from the control VM. The cloud user can remotely attest the platform as described in 5.5, and send session key to HypeBIOS encrypted by K_{SMM}^{Pub}. Afterwards the cloud user can upload an image along with the hash value encrypted by the session key. If HypeBIOS can successfully verify the image, it will launch the VM until a destroy request is received.

If the resources allocated to a cloud user are expired or no longer needed, HypeBIOS will destroy the data first, and then mark them as allocate-able to the control VM. There is an argument upon whether the control VM should be able to forcibly re-collect guest VMs memory pages. If this is allowed, a compromised control VM may be a huge threat to the whole platform. Although HypeBIOS can clean all the content in the re-collected memory pages, the effects due to missing pages would still leak side-channel information to the control VM. However, if the forcibly re-collection is not allowed, cloud providers may have a lot of troubles if they are not willing to continue providing service to some VMs. There is no way to find out an ultimate solution to this argument. If we want to satisfy the users' desire of privacy, we must sacrifice the power of cloud providers. In our prototype, we make the trade off to disable forcibly re-collection of memory but keep charging the users who do not give up the resources.

Since the control VM's memory access is also enforced by the page-access-control table and any privileged CPU or I/O instructions will be captured and checked by the double-layer VMM, it is impossible for cloud provider's internal employees to launch insider-attacks.

6. Implementation and Performance

Our prototype is built on the hardware platform with an AMD Athlon II X2 260 processor, one-dimm 2GB RAM, ASUS M5A88-V motherboard (northbridge: AMD RS880, southbridge: AMD SB850), and 2MB BIOS. We utilize AMD SVM to enable hardware based virtualization, and build the prototype based on the open source BIOS project, coreboot [10] (formally known as LinuxBIOS). The version of the coreboot code that we build our prototype with is 4.0. Note that we only use code in coreboot to perform the necessary hardware probing and initialization. Our prototype adds $\sim$3K lines of code as the HypeBIOS initializer.

For performance evaluation, we boot Ubuntu 8.04 Linux as the VM, with one CPU core (dedicated) and 512 megabytes memory (nested paging enabled). Because the northbridge of our evaluation platform does not support IOMMU, we simply directly assign the Ethernet adapter and local disk to the VM, without the interception of I/O flows. This is not a big issue since we only test on one VM. For real-world cloud computing, an IOMMU is required to support device sharing among multiple VMs.

To investigate the performance impact due to TCB minimization and decomposition, as well as the overhead introduced by the timer-based double layer mode switching, we evaluate the platform under five configurations:

1. Booting the Linux by coreboot without virtualization. We configure the boot options so that the Linux only uses one core and 512MB memory (labelled as"No_virt" in the following figures).
2. Loading the Linux with virtualization. In this case we do not decompose the TCB. Both the master and slave layers are relocated in normal memory areas and they communicate directly without SMM triggering (labelled as "No_dec" in the following figures).
3. Loading the Linux with HypeBIOS and configuring the mode transition interval as 50 ms (labelled as "50ms" in the following figures).
4. Loading the Linux with HypeBIOS and configuring the mode transition interval as 1 ms (labelled as "1ms" in the following figures).
5. Loading the Linux with HypeBIOS and configuring the mode transition interval as 500 μs (labelled as "0.5ms" in the following figures).

A series of performance evaluations are carried out as follows. The values obtained from the second test case is treated as the reference (with virtualization but without decomposition) because most of the cloud solutions adopt this system model. Testing results obtained from other configurations can be normalized according this test case (Figure 6, 7 and 9).

6.1 Common System Tasks

The first measurement is to uncompress the official Linux kernel linux-2.6.27.62.tar.bz2, which is 50.4MB in size, followed by the kernel compilation with the minimum configuration ("allnoconfig"). The wall-clock time is measured and results are shown in Figure 6. The "virtualization without decomposition" case has already caused a 20% performance downgrade compared to the "no virtualization" case. According to the gathered VMEXIT statistics, this is because of a large number of page faults (exit code 0x4E). When the timer-based mode transition is introduced in the system, the performance slightly goes down. Even if we rapidly trigger SMM, the downgrade is within 25%.

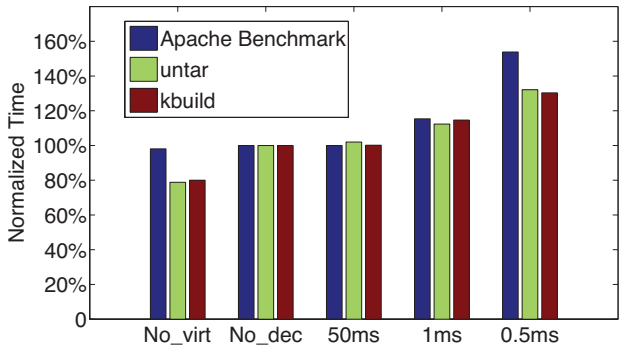


Figure 6: Kernel building, uncompressing and Apache benchmarking results of HypeBIOS.

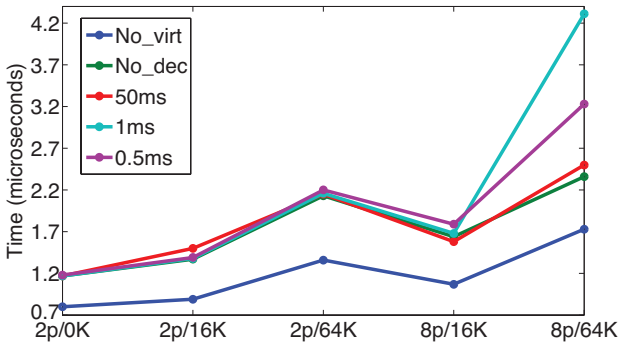


Figure 8: Context switching efficiency. The X-axis tick labels describe the process number of data size transferred between processes. Data are gathered using lmbench [28].

Another commonly used assessment is the Apache benchmarking. We send 100 number of requests with the concurrency number as 20 to the Linux web server and measures the average delay. From the results (Figure 6) we can find that Apache service is insensitive to virtualization but sensitive to SMM mode transition. The increment of the delay has an exponential trend.

6.2 CPU computing performance

Except for web services, cloud users also tend to outsource computing to cloud VMs. Thus the measurement of CPU computing efficiency is important. In this measurement, the operations of addition, multiplication and division are evaluated upon 32 bit integer, 64 bit integer, float number and double numbers, as shown in Figure 7. Again, the enabling of virtualization lows down the performance by 20% but the frequency of mode transition does not impact the performance quite much.

Another benefit of imposing little overhead to CPU computing efficiency is that the cloud users' encryption/decryption performance will not be greatly impacted. Some heavy-duty workload such as encrypted disk I/O can still be considered a good complement to HypeBIOS.

6.3 Context Switching Efficiency

We make use of lmbench [28] to measure the overhead introduced in multi-process context switching. The same observation can be drawn that the performance downgrade due to the mode transition is smaller than the overhead of virtualization. However, when the number of processes increases to 8 and the data size transferred within processes increases to 64K, we can see an apparent effect on the context switching efficiency. We may set this as part of our future work to optimize the multi-process (and multi-VM) context switching performance.

6.4 System Bandwidth and Latencies

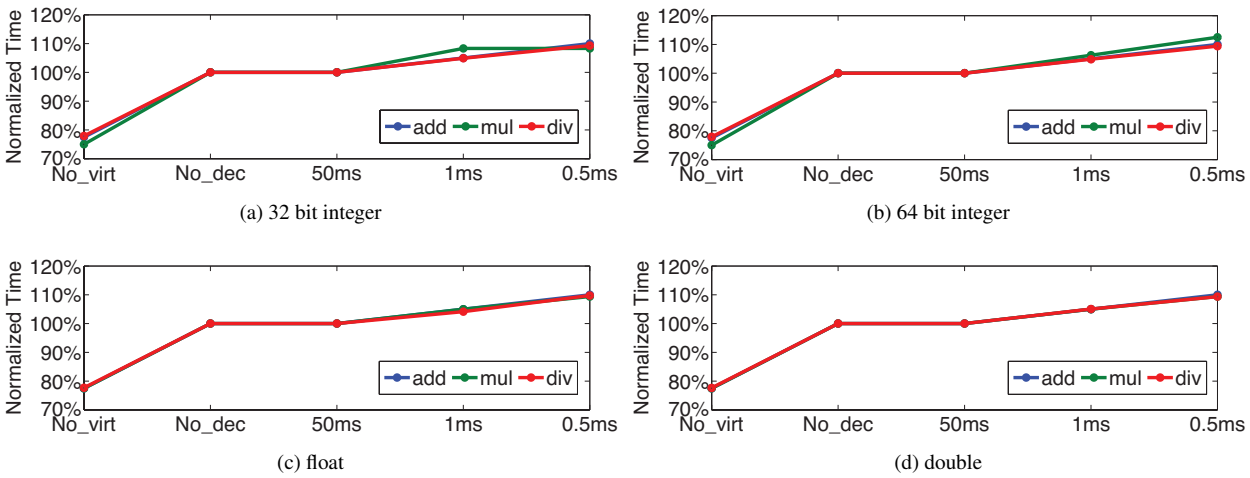
Figure 9 shows the results of a comprehensive measurement of system bandwidth and latencies, including file creating/deleting and virtual memory latencies (Figure 9a), local and remote communication bandwidth (Figure 9b), and cache/memory access latencies (Figure 9c). From the results we can conclude that TCB decomposition and mode transition do not impact the latencies and bandwidth related to physical access (R/W for cache and memory) too much but do have some influence on the virtual layers (e.g. paging and file system). However, the overhead (mostly under 25%) is totally acceptable.

7. Security Analysis

VM-to-VMM Attack Surface In any virtualization systems, executions should be intercepted if they attempt to perform privileged operations. For full-virtualization, VMEXIT happens on a privileged operation, while for para-virtualization hypercall replaces VMEXIT. No matter what kind of interception mechanisms the system adopts, the VMM must interact with the VM frequently, and thus leave its large software stack under attacks.

In HypeBIOS design, TCB size is largely reduced: those complicated drivers, management programs, and complex scheduling codes are all excluded from the TCB. In addition, we retrench and merge all the platform initialization code of BIOS, bootloader and VMM into the BIOS ROM. To make the system even more secure, we decompose the VMM into double layers. The slave layer lies in the legacy host mode to perform normal operations while the master layer resides in the SMRAM to carry out privacy-crucial executions. In this way, half of the TCB is under the native protection of hardware and is immune to both CPU stream access and Direct Memory Access (DMA). The only attack surface is the slave layer left outside. Due to decomposition, the slave layer can be quite small ($\sim$1K LOCs in our prototype) and can be easily verified (recent work has shown the capability to verify $\sim$8.7K LOCs VMM [21]). As long as the outside TCB component is secured, the interface towards the master layer is well guarded.

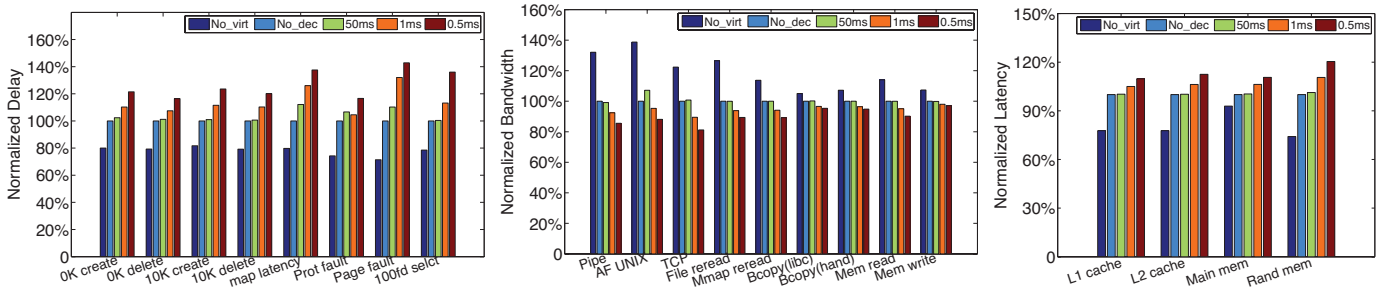
There exist some attacks poisoning SMRAM via cache on Intel's platforms [36]. However, the protection of SMRAM is out of the scope of our work. Actually, Intel has



(a) 32 bit integer (b) 64 bit integer

(c) float (d) double

Figure 7: CPU computing performance evaluation, including the operations of addition, multiplication and division upon 32 bit integer, 64 bit integer, float number and double numbers



(a) File and virtual memory latencies (b) Communication bandwidth (c) Memory latencies

Figure 9: Latency and bandwidth measurements, measured by lmbench [28].

published patches to resolve the currently known SMM attacks so far. Moreover, a proper configuration of the System Management Range Register (SMRR) [16] can clear up this problem nowadays. On AMD platforms, SMRAM cache has already been protected so that cache-poisoning attacks [37] are immunized.

VM-to-VM Attack Surface The security of HypeBIOS TCB ensues the enforcement of VM isolation. Thus many VM-to-VM attacks are immunized. Every sensitive instruction will be verified by the master layer, according to the page-access-control table. If a VM attempts to access memory pages not belonging to it, HypeBIOS will check whether this access has been authorized by the pages' owner or not. In this way, privacy breaching through memory access can be prevented.

Some may concern that if a VM can launch VM-to-VM Deny-Of-Service (DOS) attacks by causing a lot of unauthorized memory accesses that forces the master layer to process VMEXITs frequently, and takes CPU time slices away from the slave layer and other VMs. However, the handling of VMEXITs is very quick (Xen can deal with 10^5 VMEXITs per second), so it would not be a big issue if a mass of VMEXITs come to the master layer. Even if the problem exists, HypeBIOS can still defend against it by recording the VM abnormally causing VMEXITs (e.g. with a large number of unauthorized memory access) and quarantining it.

Insider Attack Surface In HypeBIOS design, the cloud provider owns only the control VM and indirectly manages cloud resource allocation through the interface provided by the slave layer. Any resources allocation changes requested by the control VM will be logged and the impacted VMs will be notified (a dedicated driver is needed). In this way, the cloud provider cannot stealthily manipulate users' secrets. Moreover, the control VM is not more privileged than any normal guest VMs. Even if the control VM is compromised or exploited by inside attackers, the access towards the resources allocated to the other VMs will be intercepted by HypeBIOS.

Under the enforcement of the page-access-control table, the only way to access the guests' resources for cloud provider is to map them as "shared with the control VM". However, this operation can only be done on the request of the pages' owner.

8. Discussion and Future Work

There could be a lot of extensions and improvements to HypeBIOS. Our future work will include but may not be limited to:

Nested Virtualization with HypeBIOS We have already supported trap-and-emulation for AMD VMLOAD/VMRUN/VMSAVE instructions. Thus nested virtualization with HypeBIOS on AMD platforms are now possible. However, due to the experiment platform limitation (our RS880 northbrige chip does not support IOMMU), we have not implemented the nested virtualization prototype. We will develop such kind of system in our future work.

Extending HypeBIOS Archiecture to Other Modes The HypeBIOS architecture is not limited to platforms with SMM. What we present here is the idea of TCB minimization and decomposition. Based on this philosophy, HypeBIOS can have many other variants to satisfy different goals. For example, Intel's TXT technology and its Measured Launched Environment (MLE) [18] can be a good candidate to replace SMM. MLE has two memory regions that are protected from malicious breaching: DRAM Protected Range (DPR) and Directed I/O Protected Memory Regions (PMRs). By relocating the master layer into the TXT measured memory regions, we can guarantee the safety of HypeBIOS from DMA attacks and can measure its integrity periodically. In this way, the crucial software components are under protection without a high performance overhead to measure the whole TCB. We have started to port SMM based HypeBIOS prototype to MLE and we will publish the evaluations in future papers.

According to a recent interview of Intel's CTO, Justin Rattner, Intel has strived to explore a "stealth" mode to provide securely processing [38]. In the long term, there should be a general-purpose solution. We need an architectural breakthrough which allows an open platform to selectively and programmatically become closed during a secure computational phase. With the architecture proposed by Intel, HypeBIOS can reside in the stealth mode, being waken up for brief periods of time, and then come back into the open. Our future effort will follow this direction once Intel publishes the "stealth" mode.

HypeBIOS as A Neutral Agent Privacy protection has motivated many approaches coming into being to keep cloud users' secrets away from the cloud provider. Nevertheless, there is a compelling reason for cloud provider to monitor its users: to ensure the "law-abiding" of the whole cloud platform. Without the ability to monitor the users, the cloud provider cannot prevent the VMs to be malicious tools. It has been reported that the attackers can utilize Amazon EC2 instances to attack other VMs on the same physical node via the cache side channel [33]. And the famous Sony PlayStation Network attack has shown Amazon EC2 as a hackers' paradise [7]. Consequently, simply depriving the capability of monitoring cloud users from the cloud provider is not a perfect solution. Fortunately, as a neutral agent in the architecture, HypeBIOS can shoulder this big beam. Without leaking users' privacy to the cloud provider while sensing the users' computation and data based on the security policies provided by the cloud provider, HypeBIOS is able serve the purpose well.

9. Conclusion

In this paper, we propose HypeBIOS to enforce isolation of VMs based on a verifiable thin virtualization Trusted Computing Base (TCB). Unlike the traditional architectures, HypeBIOS excludes the unnecessary initialization components in the boot chain and shifts the control VM (management VM) out of the TCB. The reduced TCB is further decomposed into two layers. The master layer works in the System Management Mode (SMM) and contains crucial handlers. The slave layer resides in the legacy virtualization host mode to cooperate with the master layer. We build a prototype of HypeBIOS on the x86 platform with moderate slowdown. HypeBIOS is not only an effective solution for cloud TCB securing but also a promising way to alleviate other cloud problems.

10. Acknowledgement

This work was partially funded by NSF CNS-1100221.

References

[1] ADVANCED MICRO DEVICES. AMD I/O Virtualization Technology (IOMMU) Specification, February 2009.

[2] ADVANCED MICRO DEVICES. AMD SB800-Series Southbridges Register Reference Guide, May 2011.

[3] ADVANCED MICRO DEVICES. AMD64 Architecture Programmers Manual Volume 2: System Programming, December 2011.

[4] ANDERSON, R., AND KUHN, M. Tamper resistance: a cautionary note. In *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2* (Berkeley, CA, USA, 1996), WOEC'96, USENIX Association, pp. 1–1.

[5] AZAB, A. M., NING, P., WANG, Z., JIANG, X., ZHANG, X., AND SKALSKY, N. C. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the 17th ACM conference on Computer and communications security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 38–49.

[6] AZAB, A. M., NING, P., AND ZHANG, X. Sice: a hardware-level strongly isolated computing environment for x86 multi-core platforms. In *Proceedings of the 18th ACM conference*

on Computer and communications security (New York, NY, USA, 2011), CCS '11, ACM, pp. 375–388.

[7] BAGH, C. *Sony PlayStation Network attack shows Amazon EC2 a hackers' paradise*, 2011. http://www.ibtimes.com/articles/146224/20110516/.

[8] BERGER, S., CCERES, R., GOLDMAN, K. A., PEREZ, R., SAILER, R., AND DOORN, L. vtpm: Virtualizing the trusted platform module. In *In USENIX Security* (2006), pp. 305–320.

[9] BIEDERMAN, E. Kernel korner: About linuxbios. *Linux J. 2001*, 92 (Dec. 2001), 7–.

[10] COREBOOT. http://coreboot.org/.

[11] EC2. http://aws.amazon.com/ec2/.

[12] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: a virtual machine-based platform for trusted computing. *SIGOPS Oper. Syst. Rev. 37*, 5 (Oct. 2003), 193–206.

[13] GELLMAN, R. Privacy in the clouds: Risks to privacy and confidentiality from cloud computing. Tech. rep., World Privacy Forum, 2009.

[14] GRUB2. http://www.gnu.org/software/grub/.

[15] INTEL CORPORATION. *Intel® Virtualization Technology Specification for Directed I/O Specification*. www.intel.com/technology/vt/.

[16] INTEL CORPORATION. Intel® 64 and IA-32 Architectures Software Developer's Manual, December 2009.

[17] INTEL CORPORATION. Intel® PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology, January 2011.

[18] INTEL CORPORATION. Intel® Trusted Execution Technology, March 2011.

[19] JHALA, R., AND MAJUMDAR, R. Software model checking. *ACM Comput. Surv. 41*, 4 (Oct. 2009), 21:1–21:54.

[20] KELLER, E., SZEFER, J., REXFORD, J., AND LEE, R. B. Nohype: virtualized cloud infrastructure without the virtualization. In *Proceedings of the 37th annual international symposium on Computer architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 350–361.

[21] KLEIN, G., ELPHINSTONE, K., HEISER, G., ANDRONICK, J., COCK, D., DERRIN, P., ELKADUWE, D., ENGELHARDT, K., KOLANSKI, R., NORRISH, M., SEWELL, T., TUCH, H., AND WINWOOD, S. sel4: formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 207–220.

[22] KORTCHINSKY, K. Cloudburst: Hacking 3d (and breaking out of vmware). In *Black Hat Conference* (2009).

[23] KRAZIT, T. *CNET News. Google fired engineer for privacy breach.* http://news.cnet.com/8301-30684_3-20016451-265.html.

[24] KUN, S., JIANG, W., FENGWEI, Z., AND ANGELOS, S. Secureswitch: Bios-assisted isolation and switch between trusted and untrusted commodity oses. Tech. Rep. GMU-CS-TR-2011-7, Department of Computer Science, George Mason University, 2011.

[25] MAYES, K. E., MARKANTONAKIS, K., AND TOMLINSON, A. Introduction to the tpm. In *Smart Cards, Tokens, Security and Applications*. Springer US, 2008, pp. 155–172.

[26] MCCUNE, J. M., LI, Y., QU, N., ZHOU, Z., DATTA, A., GLIGOR, V., AND PERRIG, A. Trustvisor: Efficient tcb reduction and attestation. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2010), SP '10, IEEE Computer Society, pp. 143–158.

[27] MCCUNE, J. M., PARNO, B. J., PERRIG, A., REITER, M. K., AND ISOZAKI, H. Flicker: an execution infrastructure for tcb minimization. *SIGOPS Oper. Syst. Rev. 42*, 4 (Apr. 2008), 315–328.

[28] MCVOY, L., AND STAELIN, C. lmbench: portable tools for performance analysis. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 1996), ATEC '96, USENIX Association, pp. 23–23.

[29] METH, K. Z., AND SATRAN, J. Design of the iscsi protocol. In *Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)* (Washington, DC, USA, 2003), MSS '03, IEEE Computer Society, pp. 116–.

[30] MURRAY, D. G., MILOS, G., AND HAND, S. Improving xen security through disaggregation. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (New York, NY, USA, 2008), VEE '08, ACM, pp. 151–160.

[31] NGUYEN, A. M., SCHEAR, N., JUNG, H., GODIYAL, A., KING, S. T., AND NGUYEN, H. D. Mavmm: Lightweight and purpose built vmm for malware analysis. In *Proceedings of the 2009 Annual Computer Security Applications Conference* (Washington, DC, USA, 2009), ACSAC '09, IEEE Computer Society, pp. 441–450.

[32] PAN, W., ZHANG, Y., YU, M., AND JING, J. Improving virtualization security by splitting hypervisor into smaller components. In *Data and Applications Security and Privacy XXVI*, N. Cuppens-Boulahia, F. Cuppens, and J. Garcia-Alfaro, Eds., vol. 7371 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 298–313.

[33] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security* (New York, NY, USA, 2009), CCS '09, ACM, pp. 199–212.

[34] SINGARAVELU, L., PU, C., HÄRTIG, H., AND HELMUTH, C. Reducing tcb complexity for security-sensitive applications: three case studies. *SIGOPS Oper. Syst. Rev. 40*, 4 (Apr. 2006), 161–174.

[35] STEINBERG, U., AND KAUER, B. Nova: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European conference on Computer systems* (New York, NY, USA, 2010), EuroSys '10, ACM, pp. 209–222.

[36] WOJTCZUK, R., AND RUTKOWSKA, J. Xen 0wning trilogy. In *Black Hat Conference* (2008).

[37] WOJTCZUK, R., AND RUTKOWSKA, J. Attacking smm memory via intel cpu cache poisoning. In *Invisible Things Lab* (2009).

[38] WOLFE, A. *Wolfe's Den: Intel CTO Envisions On-Chip Data Centers*, 2009. http://www.informationweek.

com/global-cio/interviews/wolfes-den-intel-cto-envisions-on-chip-d/221900325.

[39] XEN. http://www.xen.org/.

[40] ZHANG, F., CHEN, J., CHEN, H., AND ZANG, B. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2011), SOSP '11, ACM, pp. 203–216.