MUSHI: Toward Multiple Level Security Cloud with Strong Hardware Level Isolation

Ning Zhang* Ming Li† Wenjing Lou* Y. Thomas Hou*
*Complex Network and Security Research Laboratory, College of Engineering, Virginia Tech, VA
†Department of Computer Science, Utah State University, Logan, UT

***Abstract*—Multiple Level Security (MLS) has always been a center of focus since the usage of computers in military and intelligence systems. Extensive studies have been done on how to utilize virtualization technologies to provide multiple level secured execution in the cloud, yet the general assumption is that all components in the cloud service provider are trusted. With the advanced persistent threats currently faced by the military and intelligence community, it is unrealistic to assume complex information systems can remain trustworthy all the time. In this work, we present MUltiple level Security cloud with strong Hardware level Isolation (MUSHI), a novel framework that can provide hardware level isolation and protection to individual guest virtual machine (VM) execution. With MUSHI, a user can maintain confidentiality and integrity of her VM in a multicore environment even in the presence of malicious attacks from both within and outside the cloud infrastructure.**

I. Introduction

Cloud computing has defined a new paradigm shift in modern world computing. With virtualization as the key enabling technology, service providers are now able to dynamically provision infrastructure to meet the ever changing demand from users. In addition, such economy of scale has provided opportunities for better service at lower price. US military has always been striving for information supremacy, it is anticipated that the military system will also adapt to the new cloud computing paradigm[1], [2]. However the sensitive nature of multiple classification military system has created a unique challenge for system deployments. Information lost due to security breach in commercial market can usually be converted to certain monetary value, yet incidents in military system can sometimes imply death of millions. Therefore information assurance is of critical importance in MLS systems.

One of the fundamental design principles of multilevel security system is the spatial and temporal separation of information of different classification levels [3], [4], [5], [6], [7]. Virtualization software is the component that enables sharing of physical devices while enforcing isolation between VMs of different users in the cloud environment. This implicitly implies that the virtualization software and hardware will have to be trusted. It has been generally accepted that there is risk of attacks on the hypervisor from malicious guest VMs as demonstrated by several recent attacks and vulnerability reports [8], [9], [10], [11]. However, hypervisor is not the only attack surface. An adversary could also attempt to compromise the management software residing in the hosting and management servers with vulnerabilities in [8], [9], [12]. Lastly, adversaries might even use social engineering to compromise accounts of the management personnel. Given the advanced persistent threats faced by the military today, it is unrealistic to assume the cloud infrastructure can be secured against various types of malicious activity including social engineering at all time. To address this challenge, we proposed MUSHI, a hardware level isolated parallel execution environment to enable secured operations in the presence of both malicious cloud host and collocated malicious VMs. Specifically MUSHI has the following unique attributes

- Our system does not rely on a secured cloud infrastructure, therefore the security properties of user's information remains the same even if the hosting infrastructure is compromised. We believe this is a necessary requirement for future computing systems in the military environment.
- MUSHI depends on a very small set of Trusted Computing Base (TCB) including only the hardware, hardware assisted virtualization, BIOS and System Management Mode (SMM). It makes the task of analyzing and protecting the system much more manageable.
- The architecture can be realized with commodity hardware. In MUSHI, we rely on SMM memory (SMRAM), which is readily available in current generation of processors, for the isolation between the host and VM. Trusted Platform Module (TPM) is used for remote attestation of VM image integrity. Furthermore, in order to remove the attack surfaces available to malicious VM, MUSHI rely on currently available hardware virtualization of I/O device [13], [14].

II. Previous Works

Multilevel security is a well-studied topic [6], [5]. Early systems use the reference monitor in security kernel to enforce the MLS policy within the system. Recently, Multiple Independent Level Security (MILS) has emerged as a new paradigm of designing MLS system[7], [4], which aims to separate the two types of service provided in traditional monolithic kernel, namely maintaining spatial temporal separation and security policy enforcement. Applying this concept of multilevel security system design to cloud, we would need to have separation and policy enforcement in MLS cloud as well. The policy enforcement is not the focus of this paper,

978-1-4673-3/12/$31.00 ©2013 IEEE

though it can be achieved via network separation through cryptographic operations along with high speed guards. MUSHI aims to provide the separation functionality in the MLS system. Providing isolation between virtual machine on the same physical platform has been an active area of research in the past decade due to the rapid development of cloud technology in the commercial market. There are generally three approaches to provide isolation of individual VM in the cloud environment, hypervisor based approach[15], [16], [17], [7], [18], [19], [20], hardware assisted isolated execution environment [21], [22], [23], and using direct hardware access to provide necessary isolation needed [24], [13]. Within the hypervisor based approach, research generally focuses on either minimizing hypervisor [15], [16], [17], [7], [18] or hardening it [20], [25]. In hardware assisted isolated execution environment [21], [22], [23], systems use hardware functions to maintain the isolation. Our work is inspired by SICE[22] in which Azab et al. proposed to use SMM memory (SMRAM) to isolate individual workloads from potentially malicious hypervisors. However, the system still relies on the correctness of security manager and device emulation in the legacy host. There is still an attack vector from the guest VM to security manager. Furthermore the malicious legacy host who emulates all the devices can also launch an MMIO attacks [26] on the isolated workload, in which the malicious legacy host might intentionally map the MMIO memory of a manipulated device such that it overlaps with an original intended endpoint. The last type relies completely on the isolation provided by the hardware [24] [13]. In [24], [13], Szefer et al. proposed to completely remove the hypervisor layer, such that malicious VM will not have an attack surface to reach legacy host. However, this system assumes that infrastructure is always trusted which can be violated. MUSHI aims to remove such trust dependency on the legacy host.

III. Background

A. System Management Mode

System Management Mode (SMM) is a special CPU mode that is different than protected mode and real address mode. Its main purpose is to perform platform specific system resource management such as power control. SMM is entered via system management interrupted (SMI) which could be triggered by both software and hardware. Upon entering SMM mode, the microprocessor saves its entire state in a separate memory region known as system management memory (SMRAM), and continues to execute the SMI handler which also resides within SMRAM. When SMI handler finishes execution, a special instruction RSM is executed to exit SMM. Within SMM Mode, all memory protection and interrupts including the non-maskable ones are disabled. Both AMD and Intel provide locking capability such that access to SMRAM outside SMM is disabled. In AMD architecture, SMRAM can be defined dynamically through two mode specific registers (MSR) SMM_Addr and SMM_Mask which are both local to a processor core. SMM_Addr identifies the base address and SMM_Mask determines the range. The protection on this dynamic range of SMRAM can be locked or unlocked by writing a 64bit password onto the SMM_KEY MSR [27]. A technique to safe guard memory using this per core SMRAM range capability was proposed by Azab et al. in SICE[22] called memory double view. The special protection offered by SMRAM was used in memory double view to protect the memory space of individual VM from the malicious legacy host. The SMM_Addr and SMM_Mask of the processor cores that legacy host runs on are configured such that SMRAM covers the entire memory region of individual VMs to protect. On the other hand, the access to memory of legacy host or other physically collocated VMs by the malicious VM is prevented with nested paging mechanism configured upon VM setup.

B. Kernel Modification for Hardware Level Virtualization

The concept of eliminating the hypervisor was first proposed by Keller et al[24]. It was later implemented with a prototype demonstration by Szefer et al. [13]. The key idea behind NoHype[24] is to remove the surface that a malicious VMs can use to attack the hypervisor. This way, even if there are vulnerabilities in the hypervisor, there will not be an attack vector for the malicious VM to penetrate through. Kernel modules of the virtual machines that would require privilege instructions were modified such that they would cache the results during the assisted boot up. Later in the execution, it would not be neccessary to run any privilege instruction, because the results are readily available. When the user modules are loaded, the VM should be able to remain in VM mode without causing any VM exit. As a result, the hypervisor is completely removed from the execution, and thus the attack surface.

C. Secure Boot and Trusted Platform Module

Trusted computing is a standard defined by trusted computing group[28]. This technology aims to build a provable chain of trust based on the trusted platform module (TPM), which is a microcontroller that provides storage for keys and a set of limited cryptographic operations. TPM also has platform configuration registers (PCR) that are used to store measurements for attestations. Furthermore, in order to facilitate remote attestation, each TPM is equipped with an attestation identity key (AIK) which is used to sign PCRs.

IV. System Model

In this section, the overall system would be presented first followed by threat model and design goal.

A. System Overview

For simplicity, we assume there are only three types of servers within the cloud, load balancing server, image server and hosting server. Load balancing sever is responsible for resource allocation. Each VM image is composed of two separate partial images, the kernel image, which contains the OS and user image which contains user application or files. Kernel image is shared among users of same virtual platforms.

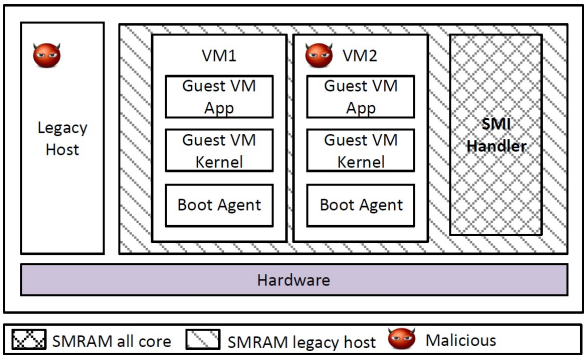

Fig. 1. MUSHI Architecture

The content in the user image is sensitive, thus encrypted with a secret key maintained by the user. The VM image server stores these two types of images. There are attackers within and outside the cloud infrastructure. Shown in Fig. 1 is a typical deployment view of MUSHI. Legacy host coordinates resource management. SMI handler resides in SMRAM that is protected from all processor cores for bootstrapping trust in VM execution. Individual user virtual machine runs directly on the hardware after trusted startup.

B. Threat Model and Assumption

Threat Model: We consider two types of adversaries. The first type is the insider adversaries who can be employees that work for the cloud service provider or people who have gained access to infrastructure's internal network by social engineering or other means. They have control over the all servers, thus capable of changing any user's image or the legacy host on the hosting servers. The goal of the insider adversaries is to steal documents and applications from the user's VM. Furthermore, these adversaries would not expose their insider access with a simple deny of service attack (DoS).The second type is the outsider adversaries who reside outside the cloud infrastructure. In particular, we consider them the malicious users who have control over the VMs residing on the same physical server. This type of adversary is capable to using various VM Exit to exploit the software bugs in the hypervisor. As a result, both types of adversaries are capable of gaining control of the host system and thus compromising the confidentiality, integrity and availability of legitimate user's VM. The goal of outsider adversaries is similar to the insider in that they want to compromise the confidentiality and integrity of the target user's VM. However, since it is relatively easy to gain access a guest VM, they are not concerned of being exposed and might launch DoS attack on neighboring VMs.

Assumptions: We assume physical security will be in place such that the hardware will be protected in controlled environment. Therefore any hardware level attacks such as physically probing the bus and memory would not be possible. Furthermore, we assume the platform MUSHI runs on is equipped with TCG's trusted boot [28] hardware that has Core Root of Trust for Measurement (CRTM) along with trusted platform module (TPM).

C. System Design Goal

The goal of MUSHI is to provide a trusted isolated environment for virtual machine execution, which is one of founding piece towards a trusted multilevel security cloud environment. Virtual machines should be instantiated securely and they should remain that way throughout the life cycle. In MUSHI, we try to achieve the following

Trusted Execution Upon startup of VM, the integrity of both the kernel and user image and MUSHI should be attested to the user, thus the trusted initial state.

Isolation The VMs running on the same physical platform should be isolated. This isolation will provide the confidentiality and integrity service to the user VM during execution. However it is not in the scope of the system to prevent side channel attacks such as timing, heat or power analysis or potential covert channels.

User Image Confidentiality Besides run time protection, MUSHI also aim to provide data at rest confidentiality service by encrypting the user image with key provided by the user.

V. MUSHI Design

Recall that threat model of MUSHI includes insider and outsider adversaries. Even with a compromised hosting infrastructure, MUSHI aims to provide trusted execution environment based on a very small TCB with commodity hardware. In the remaining of the section, we will first discuss the changes needed to build MUSHI into the current hosting platform, followed by the details on three key phases for the life cycle of a user virtual machine, from the request of user to start the virtual machine to the shutdown of the virtual system, including resource allocation, trusted startup and VM execution/exit.

A. Hosting System Integrtion with MUSHI

MUSHI which enables the trusted execution environment will need to be integrated into the server system. Furthermore, the hosting system should be able to attest to a remote user the integrity of MUSHI during the trusted execution environment initialization of the VM. There are two main integrated components of MUSHI in the hosting server. The first one is the generation of key pair used later to facilitate communication between MUSHI and the remote attester. The second component is the insertion of MUSHI into the SMI handler of the system, and further creating a measurement that later can be used for attestation. For the first component, besides the regular steps of a trusted boot of the platform, the initialization code will need to first generate a public/private key pair, K_{smm} and K_{smm}^{-1}. The public key K_{smm} will be extended onto the TPM's PCR, and thus its integrity can be guaranteed, while the private key K_{smm}^{-1} is stored in SMRAM. Then MUSHI is copied into SMRAM and inserted into the SMI handler table. And the initialization will measure the MUSHI and extends the measurement onto the TPM's PCR, and immediately lock the SMM to prevent any changes even by the hypervisor, and thus the integrity of SMM including

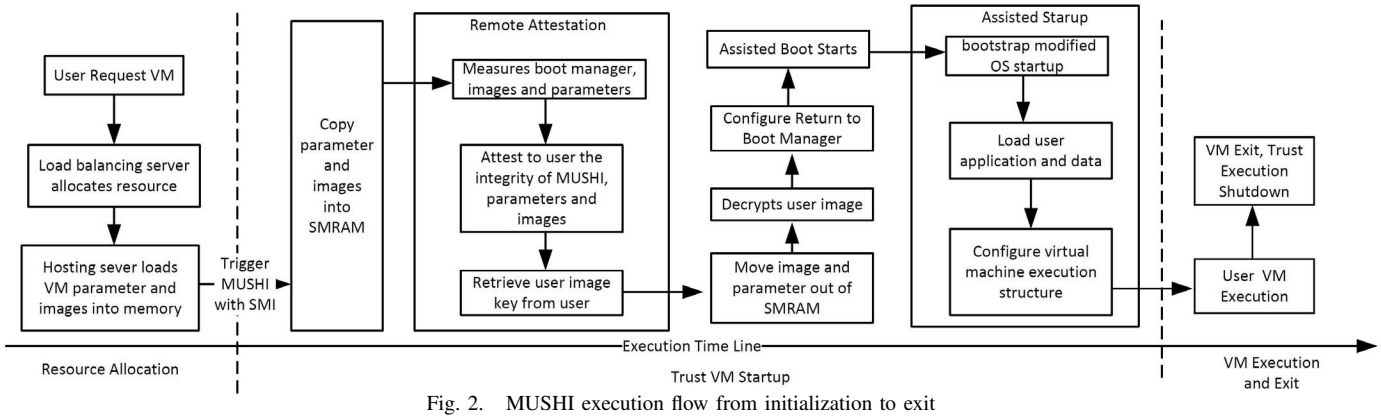

Fig. 2. MUSHI execution flow from initialization to exit

MUSHI can be maintained and attested to with the measurement later. Note that the confidentiality of the private key K_{smm}^{-1} can be is guaranteed because SMRAM is locked immediately after measurement of MUSHI, malicious VM or hypervisor will not be able to access it due to the property of SMRAM lock. One can implement MUSHI by using the coreboot[29], an open source BIOS, or by modifying the BIOS image binary such that SMM Mode remains unlocked until initialization code takes the measurement and locks it down.

B. Life Cycle of User Virtual Machine

A detail execution flow of a user VM life cycle is shown in Fig. 2, which we will give detail descriptions for

1) Resource Allocation: When a user requests for a virtual machine, the load balancing server will communicate with an available hosting server to allocate the physical resource for the user. The legacy host on the hosting server will then request the kernel image and the encrypted user image from the image server. The kernel image is shared among many users who designated the same virtualized platform, while the user image contains modules and files that are unique to individual user or user groups. The legacy host will load boot agent, which will assist system startup of the VM image, into memory. Once both images are received and loaded into memory, legacy host will save all the run time parameters of the requested virtual machine and trigger a SMI to invoke MUSHI. The triggering of SMI is end of the resource allocation phase. Once in SMM mode, the processors cores allocated to the user VM will start preparation while the other cores will resume normal operation. For the allocated cores, the parameters and run time images are copied into SMM memory for the integrity verification in the trusted startup phase.

2) Trusted Startup of User VM: Trusted startup of a user VM consists of two main steps, the remote attestation of integrity and the assisted start up for hardware only virtualization.

Remote Attestation: There are three goals in the remote attestation step. First, the integrity of MUSHI needs to be attested. Second, the integrity of images and parameters needs to be attested. Lastly, a secure communication channel needs to be established to retrieve the user image encryption key. To accomplish these three goals, MUSHI first generates a new public/private key pair, K_{com} and K_{com}^{-1}, which will be used later for secure communication with the remote user. The user then generates a random nonce, and request MUSHI to perform the attestation. With this nonce, MUSHI generates two signed values. The first one contains static measurement of TPM's PCR signed with the AIK key stored in the TPM platform, which attests the integrity of MUSHI as well as the K_{smm}, the public key generated during the system startup. Then the second value is signed with the K_{smm}^{-1}, consist of the measurement of the images, VM parameters as well as the public key used for communication K_{com}. User will only accept this measurement if the signature can be verified with the public key K_{smm} attested in the first signed value. This will allow the remote user to verify integrity of the user VM and the public key used for key transmission. And the public key of the user can be appended to the user image. With this setup, remote user and MUSHI can then established a secure connection which is then used to transmit the decryption key for the user image. Once the key is retrieved from the user, MUSHI will move the images into the intended memory region, and decrypts the user image. At this point, all the preparation for VM execution is done, and the system should move onto assisted startup of the virtual machine. However SMM cannot change most of the processor state, we can use the technique in [22] to modify the page table instead. Once modification is made, MUSHI will exit SMM mode, and hands the execution control over to boot agent, and thus the starting of the assisted startup step.

Assisted Startup: The boot agent will act temporarily as a traditional hypervisor during the assisted startup process. Processor and device discovery is one of the very first steps performed by operating system in a system startup. However, the virtualize environment which were specified by the user and finalized by the resource allocator is most likely not the same as actual platform hardware. A typical VM requested by user will most like use a very small portion of the server resources, such as 2 processor core out of 40 total available in the hosting server. As a result, when the operating system performs processor and device discovery, the hypervisor like boot agent will have to perform filtering or emulation such that the user virtual machine will be initiated with the designated resources. For PCI device discovery, the operating system use

values in a known range of memory as indicator, and the boot agent will have to emulate some of the memory read such that it would appear to the VM that such device does not exist. The same emulation capability will have to be included for the processors as well. In particular, boot agent will have to instrument the discovery of clock frequency, core identifier and processor features. Once system resource discovery is done, the guest OS can continue to boot and load modules. Though we now can bootstrap the guest OS in a virtualize environment, there is still the issue of privilege instructions. In traditional virtualization, privilege instructions are trapped and emulated by the hypervisor, however this will entail continuous interaction between the hypervisor and virtual machine which is undesirable. In this case, the operating systems will need to be modified such that results of privilege instructions such as CPUID are cached, and thus eliminating VM exits, removing the attack interface from VM to hypervisor. Once user image is loaded, the boot agent will then setup the environment for the guest virtual machine by setting up the appropriate hardware virtualization structures giving the full processor core control to guest VM. Then boot agent will execute the VM in hardware virtualized environment, and then execution move into the next phase of the life cycle.

3) User VM Execution and Exit: At this point, the control is handed off to guest virtual machine and user applications. When the guest shuts down the virtual machine or attempt any malicious activity, there will be a VM exit that will cause the execution context to fall back to the boot agent. Regardless what cause the VM exit, boot agent will save off the VM image if necessary, then perform any sanitization needed and terminate the VM execution, and free up the allocated resource. Meanwhile, a malicious legacy host might attempt to send nonmaskable inter processor interrupts (IPI) to disrupt the control flow of the isolated user virtual machine which would also cause a VM exit as well, however it is our assumption that adversaries who has gained insider access to the infrastructure will not expose themself by performing simple deny of service attack.

VI. Design Analysis

A. Security Analysis

1) Trusted Execution: In the proposed architecture, MUSHI is part of the trusted boot process which can be attested by the TPM. Building upon this trust, the integrity of the boot agent, kernel image and user image are attested to the user with the MUSHI in isolated SMRAM environment, therefore we've established chain of trust in our system startup.

2) Isolation of VM to legacy host and VM to VM: The run time isolation of individual VM from the legacy host and neighboring VM can provide run time confidentiality and integrity. With such isolation, the memory content and control flows can be protected. This is achieved by isolating processor, memory and I/O device.

Isolation of Processor Individual cores are dedicated to each guest VM, thus processors owned by different users are separated. A malicious legacy host can still send nonmaskable inter processor interrupts (IPI) to the user VM, which will cause a VM exit. However, the only action that boot agent performs after user VM execution is to shut down. Therefore there is no way for the legacy host to alter control flow in user VM by issuing interrupts.

Isolation of Memory Isolation of memory is achieved through two mechanisms. The memory of user VM is isolated from the malicious legacy host by SMRAM protection. Even though there are attacks on SMRAM based on cache poisoning[30], the mitigation[22] is fairly straight forward. This memory is also isolated from neighboring VM via extended page table, which is set up by the boot agent and requires privilege instruction to change, yet any VM exit will trigger shutdown of the virtual machine, therefore malicious VMs will not be able tamper with memory content of neighboring VM.

Isolation of I/O Device The virtualization of hardware devices is assumed to be provided by hardware, in which case, the virtualized hardware will appear as if it is a real physical hardware to the software system. And the isolation of I/O device is achieved by assigning different physical device or hardware virtualized device to individual VMs.

3) Data at Rest: For many DoD developed applications, it might not be desirable make the executable public. Malicious attackers can gain much better understanding by analyzing the binary images. Therefore in MUSHI, we aim to provide data at rest security by encrypting the user image with secret keys provided by the user. During execution, user will provide the key only if the attestation on isolated execution is successful. Therefore, even if an adversary gains internal access of the cloud infrastructure, she will not be able to look inside the user applications due to memory protection provided by MUSHI. The key for decrypting this application image is transmitted during the attestation using a secure channel, and thus it is confidentiality is guaranteed. The management of user keys can be accomplished using existing key infrastructures and is not in the scope of MUSHI.

4) Availability: MUSHI could not defend against insider who has controlled over the malicious legacy host for DoS attacks. An insider could simply remotely shutdown various servers in the cloud to deny access. However, it could defend against malicious neighboring VM quite effectively, since processor and memory are both isolated. The remaining share resource contains the bus and virtualized I/O devices, and it was demonstrated in [13] that such I/O DoS has little effect.

5) Covert Channels: Eliminating side channels in a shared infrastructure is almost an impossible task. It is generally accepted that high speed covert channels in MLS systems are highly undesirable. In MUSHI, individual VMs gets their own dedicated memory and processors, which reduces some of the high bandwidth covert channel like L1 cache or shared memory locations. However since the I/O infrastructure is still shared, it is possible to construct covert channels from the latency in the bus and etc. Detail analysis on the specific system will need to be performed before deployment.

B. Performance

In MUSHI, each VM has its own dedicated resource. Therefore user experience should be more stable since there is no oversubscription. Furthermore, since hypervisor is removed and device emulation is done via hardware, there should be an overall system improvement. The magnitude of improvement will be heavily system specific [13]. For the measurement of the images and decryption of user images, it is only a onetime cost that is associated with each VM start, and will not affect the operational performance.

C. Deployment

MUSHI relies on several hardware features to protect VM. Even though the security attributes of these features has been a topic in several research work and prototypes [22], [13], [24]. Careful examination of the hardware platform is still needed prior to deployment. In addition, BIOS image binary modification is not be scalable if the cloud infrastructure is highly heterogeneous, which will likely be the case. In addition, if a hardware mechanism is used to protect BIOS rom from malicious legacy host, it would be impossible to push an update to MUSHI remotely. For the future, it would be necessary to build MUSHI into UEFI[31] as a signed driver.

VII. Conclusion

In this work, we have expanded on the previous work's threat model in that we assume more realistic adversaries due to sensitive nature of information in military systems. Adversaries exist both inside and outside the cloud infrastructure. They can attack via the host server or the collocated VM. We have proposed a novel trusted isolated execution architecture for cloud based MLS systems that is capable of providing user confidentiality and integrity in the presence of such adversaries.

References

[1] "High-priority requirements to further usg agency cloud computing adoption." http://www.nist.gov/itl/cloud/upload/SP_500_293_volumeI-2.pdf.

[2] "Federal cloud computing strategy." http://www.cio.gov/documents /federal-cloud-computing-strategy.pdf, Feb. 2011.

[3] C. Boettcher, R. DeLong, J. Rushby, and W. Sifre, "The mils component integration approach to secure information sharing," in *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp. 1.C.2–1 –1.C.2–14, oct. 2008.

[4] J. M. Rushby, "Proof of separability: A verification technique for a class of a security kernels," in *Proceedings of the 5th Colloquium on International Symposium on Programming*, (London, UK, UK), pp. 352–367, Springer-Verlag, 1982.

[5] *Department of Defense Trusted Computer System Evaluation Criteria, Department of Defense, DOD 5200.28-STD*.

[6] J. P. Anderson, "Computer security technology planning study. technical report esd-tr-73-51, us air force." http://seclab.cs.ucdavis.edu/projects/history/seminal.html, 1972.

[7] J. M. Rushby, "Design and verification of secure systems," in *Proceedings of the eighth ACM symposium on Operating systems principles*, SOSP '81, (New York, NY, USA), pp. 12–21, ACM, 1981.

[8] "Vulnerability report for xen 4.x." http://secunia.com/advisories/product/33176/?task=advisories_2012, Feb. 2012.

[9] "National vulnerability database." http://nvd.nist.gov/.

[10] J. r. R Wojtczuk, "Xen 0wning triology," *Black Hat Conference*, 2008.

[11] K. Kortchinsky, "Cloudburst: A vmware guest to host escape," *Black Hat Conference*, 2009.

[12] A. Minozhenko, "How to hack vmware vcenter server in 60 seconds," *Defcon*, 2012.

[13] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, (New York, NY, USA), pp. 401–412, ACM, 2011.

[14] "Pci sig: Pci-sig single root i/o virtualization." http://www.pcisig.com/specifications/iov/single_root.

[15] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "sel4: formal verification of an os kernel," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, (New York, NY, USA), pp. 207–220, ACM, 2009.

[16] U. Steinberg and B. Kauer, "Nova: a microhypervisor-based secure virtualization architecture," in *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, (New York, NY, USA), pp. 209–222, ACM, 2010.

[17] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "Trustvisor: Efficient tcb reduction and attestation," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, (Washington, DC, USA), pp. 143–158, IEEE Computer Society, 2010.

[18] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, (New York, NY, USA), pp. 335–350, ACM, 2007.

[19] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 193–206, Oct. 2003.

[20] Z. Wang and X. Jiang, "Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, (Washington, DC, USA), pp. 380–395, IEEE Computer Society, 2010.

[21] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: an execution infrastructure for tcb minimization," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 315–328, Apr. 2008.

[22] A. M. Azab, P. Ning, and X. Zhang, "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*, CCS '11, (New York, NY, USA), pp. 375–388, ACM, 2011.

[23] F. Z. A. S. Kun Sun, Jiang Wang, "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, Fairfax, VA , USA, 2011.

[24] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "Nohype: virtualized cloud infrastructure without the virtualization," in *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, (New York, NY, USA), pp. 350–361, ACM, 2010.

[25] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky, "Hypersentry: enabling stealthy in-context measurement of hypervisor integrity," in *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, (New York, NY, USA), pp. 38–49, ACM, 2010.

[26] Z. Zhou, V. Gligor, J. Newsome, and J. M. McCune, "Building veriable trusted path on commodity x86 computers," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, (San Fransico, CA, USA), IEEE Computer Society, 2012.

[27] "Advanced micro devices. amd64 architecture programmers manual," vol. 2, Sept. 2007.

[28] "Trusted computing group." http://www.trustedcomputinggroup.org/.

[29] "Core boot." http://www.coreboot.org.

[30] R. Wojtczuk and J. Rutkowska, "Attacking SMM Memory via Intel CPU Cache Poisoning," Mar. 2009.

[31] "Uefi specification." http://www.uefi.org/home.