Architecture Support for Controllable VMI on Untrusted Cloud

Jiangyong Shi(✉) and Yuexiang Yang

National University of Defense Technology, Changsha, China
{shijiangyong,yyx}@nudt.edu.cn

Abstract. This paper combines architecture isolation with latest Intel SGX technology to make a controllable virtual machine introspection architecture on untrusted cloud. The main goal of SGX is to protect important applications from being attacked by untrusted OS, while the main goal of VMI is to protect OS from being attacked by untrusted applications. So it seems like contradictory, but actually they are complementary. By combining SGX and VMI, we can both monitoring the behavior of untrusted applications and preventing sensitive applications from being monitored. This is very practical in public cloud, as the cloud server provider is untrusted, but we still rely on its resource to provide computing. As far as we know, this is the first proposal to implement security monitor in an untrusted cloud with the help of trusted hardware. Preliminary security analysis and performance evaluation show that our architecture can ensure the confidentiality and integrity of the VM hosted on untrusted cloud server while providing VMI services with less than 20% overhead.

Keywords: VMI · SGX · Enclave · Untrusted cloud · Security design · Privacy

1 Introduction

Virtual Machine Introspection (VMI) [1] has been evolved over the last decade. While it indeed improved the security of VMs by stealthily monitoring the execution of VMs of different tenants, it also has the risk of privacy invasion, especially when the cloud server is untrusted. The condition of cloud service provider (CSP) being untrusted includes: (1) The CSP itself abuse of privileges. Even though large CSP companies such as Amazon and Microsoft have good reputation in information area, they might still collect users' data and behavior using techniques including VMI in order to better provide service or coordinate with the investigate demand of the government. Besides, internal staff who is vicious or act as a spy might intentionally leave backdoors or manipulate the privileges to steal users' data or break VM integrity. (2) The cloud server being attacked. This is especially true when a malicious or compromised VM escapes the constraint of VMM and breaks the isolation between VMs. When that happens, the

R. Deng et al. (Eds.): SecureComm 2016, LNICST 198, pp. 307–316, 2017.
DOI: 10.1007/978-3-319-59608-2_18

attackers can either shutdown the VM on the same physical machine to cause deny of service attack, or steal sensitive information of other VMs by VMI. As the VMM is a privilege layer below the VMs and CSP takes control of all the VMMs, these two situations are hard to prevent by the users.

To change that, we investigate the hardware routine to control VMM's access to resources of VMs. The resources a VM owns include disk, memory and network packet. The network flow can be protected through security protocol such as SSL. The disk can be encrypted using full disk encryption (FDE) technique. The most difficult to protect is volatile storage including memory and CPU state due to several reasons. Firstly, volatile storage is frequently read and wrote which makes encryption extremely performance costing. Secondly, multiple VMs' share of the same physical memory space and frequently memory scheduling make it hard to define the memory range to protect. While virtualization provides good memory isolation between different VMs, it doesn't consider the VMM being malicious. So the VMM can arbitrarily manipulate the physical memory space to intercept sensitive data or inject malicious code, including VMI operations.

Based on the analysis, we propose a controllable VMI architecture based on existing hardware feature and new hardware enhancement technology. Our architecture can provide VMI as a service based on the users' requirements, while in the same time prevent the VMM from monitoring the VM without the permit of the user.

The organization of the paper is as follows. Relate works with VM and VMM isolation are discussed in Sect. 2. To make our work more clear, the trusted model of our architecture is described in Sect. 3, followed by our architecture design in Sect. 4. The detailed implementation of our architecture is described in Sect. 5. Sections 6 and 7 analyze the security and performance of our architecture. Lastly, a short conclusion is drawn in Sect. 8.

2 Related Work

CryptVMI [2] proposes an encrypted VMI framework to prevent cloud manager from forging VMI requests and intercepting VMI results. Even though it considers the problem of cloud manager being malicious, it assumes that the VMM is secure and trusted, which is not always true as we discussed in the introduction part. In order to control VMM's access to memory of VMs, we need to re-design the hardware architecture to support memory isolation between VMs and VMM. Related researches include HyperWall [3], HyperCoffer [4] and Intel SGX [5].

HyperWall designs a confidentiality and integrity protection (CIP) architecture to protect the VM memory from being tampered by VMM [3]. The architecture uses resource isolation (focusing on the memory of the virtual machines), as opposed to cryptographic isolation, to implement hypervisor secure virtualization. The architecture includes modification of hardware, hypervisor and VM, such as new instructions and registers to CPU, new procedure for updating memory mapping, and new random number generators. By isolating memory of VM and VMM, it can successfully prevent VMM from tampering with VM

memory. However, this also adds challenge to VMI, as VMI requires mapping the VM memory to where VMI applications are deployed. HyperWall scrubs any memory pages that are freed and prevents them from being shared, which disables most VMI applications that relied on memory sharing.

Compared with HyperWall, HyperCoffer has less effect on hypervisor as it adds another separate layer called VM-shim to cooperate with the secure CPU [4]. The security of VM-shim is ensured by combining each VM-shim with VM to reduce the impact on TCB. Besides, it uses hardware encryption and integrity checking to protect the data of VM memory and disk, which can defend both VMM attack and hardware attack. However, it still needs the cooperation of VM OS and hypervisor to work. Besides, as the hypervisor is authorized part of the data access according to the type of VMEXIT, it is still possible to attack the VM. Moreover, encryption of the VM memory disabled VMI tools, which is a loss to security.

Both HyperWall and HyperCoffer aim at protecting the whole VM from being accessed by VMM, however, this would add difficulties to management of VMs as VMM is the main management layer. Besides, all-VM encryption and integrity checking add high performance cost and are inflexible. To overcome these disadvantages, Intel proposed new protection architecture named Intel Software Guard Extension (SGX) in its sixth generation processor Skylake. SGX is designed to protect the memory of applications, instead of all-VMs, from being tampered and snooped by VMM without the acknowledgement of the VM user. As it is application granularity, SGX is more flexible than HyperWall and HyperCoffer. Besides, only encryption specific application memory brings lower performance cost. Applications based on SGX include Haven [6], VC3 [7] and M2R [8], most of which aim at providing a new protection of the existing sensitive and vulnerable applications.

However, SGX only can protect the ring 3 applications from being accessed by ring 0 software, including OS kernel and VMM. SGX cannot protect kernel-level security applications, such as AV and Firewall, while these tools must run in kernel mode to monitor other programs.

Our core idea is to put VMI-applications into a disjunctive VM (VMI Server) and prevent VMM from being able to read or write other VMs' memory. If VMM is able to map memory addresses of the introspected VM, it might map the VM memory to its own address space so as to read or write its contents, which must be prevented in un-trusted cloud. The details will be discussed in Sect. 5.

3 Trusted Model

We assume the cloud service provider and the cloud server to be untrusted. This indicates following potential threats to cloud users' VM security and privacy:

1. The delivery of the VMI requests might be intercepted by CSP, and they might use the intercepted data to replay the VMI requests. The CSP might also forge VMI requests to get the state of the VM.

2. The VMM might arbitrarily access the physical memory of the user's VM and use VMI to get sensitive information. As VMM has higher privilege than the VM, it is hard to prevent by traditional cloud architecture.
3. The result of the VMI operations might be intercepted by VMM, as the results would finally be sent via physical network which is controlled by VMM.

To counter these threats, our proposed architecture use following methods:

1. The VMI operations need the participation of the CSP, but only to locate the position of the cloud server to which the VM locates, which requires only the User-ID and VM-ID. The VMI command is encrypted by a user key and always kept secret to CSP.
2. The memory access of the VM is strictly controlled by a hardware access control list located in physical memory and protected by CPU. The memory to store the ACL table is initialized before the VMM launches and only accessible to the CPU.
3. The VMI results are encrypted using a CPU key and only the user can decrypt it.

The trust root of our architecture is the new designed CPU, which reserves a part of physical memory to store ACL and control the access to VM memory, and provides a remote control mechanism for the user to adjust the policy of ACL.

4 Architecture

To overcome the limits of HyperWall, we propose a new controllable VMI architecture based on hardware support. The core idea is to separate the memory of VMM and VM by similar solutions with HyperWall, and enable VMI access to VM memory by stripping VMI from VMM to a separate VM and authorize the VMI VM to access target VM memory when there is a request from cloud user. The VMI Server can be kept reliable by trust initialization and persist integrity monitoring. Our architecture is shown in Fig. 1.

The main process of our architecture consists of the following steps.

1. The user sends its VMI request to cloud manager. The request is a 3-tuple consists of User-ID, VM-ID and VMI command, with the command encrypted using a public key EKvm and can only decrypted by server CPU with private key DKvm. The command is a set of security related instructions, such as process list and network connection list.
2. After receiving the request of user, the cloud manager will look up the corresponding relationship between (User-ID, VM-ID) and physical machine, thus locating the physical machine where the VM is. Then the cloud manager sending the request to the physical machine.
3. On receiving VMI request from cloud server, the physical CPU adjusts the ACL to enable VMI server's access to the VM's memory and decrypted the VMI request.

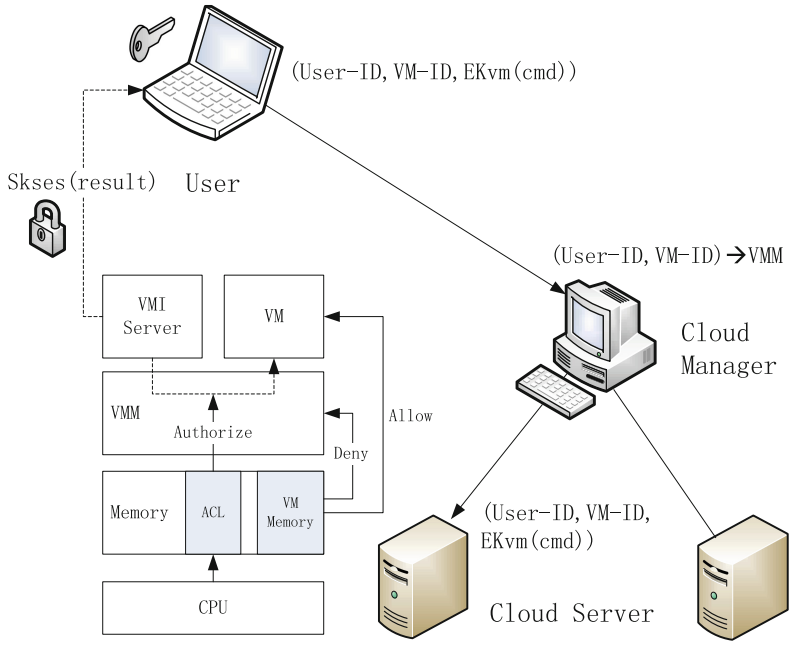

Fig. 1. Architecture of controllable VMI (VMI as a service)

4. On receiving plaintext request, the VMI server would look up the ACL list to see if the vm belongs to the user and executes the VMI command. Then encrypt the result and return it to the user. As the result is encrypted using a session key (SKses), the VMM or CSP cannot decrypt it.
5. There would be a counter calculating the fails of VMI requests on user side as the CSP might just discard the VMI requests or results. So if there are too many failures, the user can stop the execution of the VM. Another counter exists in the server side recording the number of VMI requests and is sent to user with the VMI results. So if there are any forged VMI requests, the user can discover it through the change of the counter.

5 Implementation

Our architecture mainly consists of three parts, including the memory isolation of VM and VMM, the VMI control mechanism and the establishment of secret communication channel. We will discuss them in the following parts. To disallow memory sharing between VMM and VMs, we proposed an memory isolation mechanism in Sect. 5.1. To enable and control VMI Server's access to the VM memory, we designed our access control list (ACL) which will be discussed in Sect. 5.2. And to enable secure communication between the VMI Server and the user, we enhanced the secure communication channel of HyperWall in Sect. 5.3.

5.1 Isolation of VM and VMM

The context VMI need mainly consists of memory and register state. To prevent VMM from arbitrarily introspect the VM without the permit of the owner of the VM, memory and register state should be protected.

Hyperwall encrypts the general purpose registers and generates a hash over the state so as to ensure its confidentiality and integrity when VM exits (except for that of hypercall) [3]. While to memory, a VM's memory pages are locked and protected by CPU when the VM is initialized. And there is a part of hardware-only accessible memory to store the CIP table (used for access control) and TEC table (used for attestation and logging) of the VM memory.

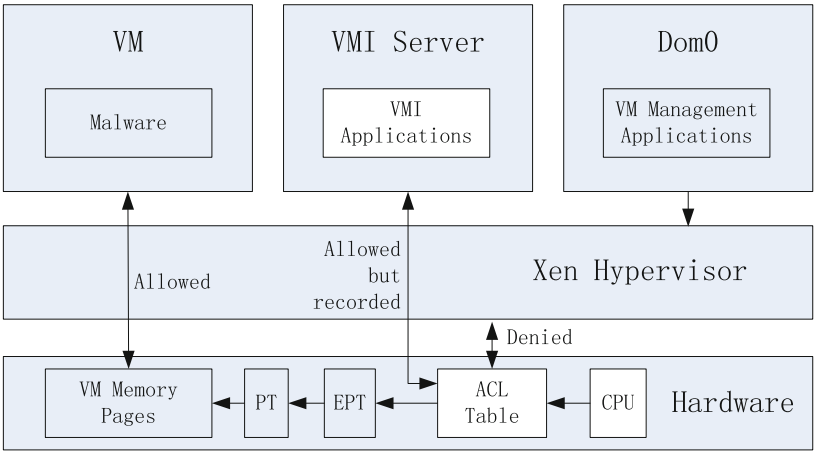

Fig. 2. Memory Isolation of VM and VMM

To disable VMM access to VM memory while in the same time keep the ability of introspecting a VM, we use the disjunctive VMI model proposed in [9]. The core idea is to move the VMI applications from VMM to a separate VM so as to reduce the size of TCB. The security of VM where VMI applications are deployed can be ensured by varies of methods, including security boot and integrity checking based on TPM, minimal kernel design to reduce the attack surface and even the latest Intel SGX to protect VMI applications from being tampered. By using SGX to execute VMI applications in enclaves, we can isolate the VMI application from being accessed by the kernel and other applications of the same VMI server. As shown in Fig. 2, and the others being untrusted. The CPU is the root of the trust chain and the trust relationship is controlled and transmitted by the ACL table.

5.2 VMI Control Mechanism

HyperWall control the access of VM memory by CIP table. To enable VMM access the VM memory, we just need to remove the HD (Hypervisor Deny)

Protection bit of the corresponding pages. Different from HyperWall, HyperCoffer adjust the policy of VM-shim to authorize access to its memory. By using *raw_ld* and *raw_st* instructions, the VMM can directly load the VM memory. Both the method of HyperWall and HyperCoffer will break the isolation again and give the VMM opportunity to subvert the VM and that's why we use a disjunctive VMI model as discussed in above section.

We use a tailored access control method of HyperWall to implement the access control of VMI requests. To enable VMI server access the VM memory, we add a ID (introspection deny) option to the protection field of the ACL. To enable users' attestation to the VMI actions, we add a counter field to the ACL table, which is used to record the VMI times.

Table 1. Fields in ACL table

Fields	Content	Length
MA	Machine address	48 bit
GPA	Guest physical address	48 bit
VMID	VM identifier, generated by CPU when VM is created	8 bit
In-use	Controls whether the page is usable	1 bit
Protection	Controls whether the page is accessible by VMM or VMI server	2 bit
Counts	Record the times of this page being accessed by VMI server	21 bit

The detailed component of VMI access control list (ACL) is shown in Table 1. The VMI Server deploys VMI applications in Enclave which are protected by SGX from the VMM or other applications and kernel of the VMI Server. After the hardware receive the VMI requests, it will enable memory share with VMI Server by removing the ID protection, then deliver the requests to the VMI Server. When the VMI requests arrive at VMI Server, the VMI applications parses the requests and maps the memory pages of the introspected VM to its own memory space. After that, the VMI applications are able to read the VM memory and extract the security related information. After the VMI operations, the counter field of the VM pages in ACL will increase. The VMI results and the counter are encrypted and sent back to user via secret communication channel, which will be discussed in detail in the following section. The user can compare the counter to verify that the VMI Server has introspect the VM as he/her requires, instead of arbitrarily introspecting the VM without the permit of the user.

5.3 Secret Communication Channel

HyperCoffer provides no secure communication routine between the user and VM. So we just discuss and improve the method of HyperWall. The keys we used during secret communication channel building are listed in Table 2.

Table 2. Keys in secure remote communication

Key	Function
SK_{cpu}	Encrypt other keys, enclave memory. Sign VMI and attestation results
(EK_{vm}, DK_{vm})	Public-private key pair used to build secure communication between user and VM
$Cert$	Certificate of CPU, issued by hardware manufactory, can verify the signature's validity of SKcpu
SK_{ses}	Session key, used to encrypt communication between user and VM

Secret communication channel is essential in securely transferring information between remote users and the VMI server. As the VMM can intercept these communications, it might suffer MITM attack. To counter this threat, user's data is send to VM via public-private key pair (EK_{vm}, DK_{vm}) so as to defend against MITM attack. The key pair is generated during VM initialization and the public key EK_{vm} is signed by CPU with its key SK_{cpu}, user verifies the sign using CPU's certificate to make sure the EK_{vm} is coming from the VM, which is very similar to SSL protocol.

HyperWall states that the user can establish secret channel using the EK_{vm}. However, as the VMM can intercept the EK_{vm} during the above handshake process, it can forge VMI requests and decrypt communication which is encrypted using DK_{vm}. So we cannot directly use DK_{vm} to encrypt the result data. Instead we add another modification to the process of HyperWall. After getting EK_{vm}, the user generates a session key and encrypts it with EK_{vm}, then send it to VMI server. After the VM server get that, they can establish secret channel using the session key. As the session key changes whenever a new VMI request is made, the VMM is unable to execute replay attack.

6 Security Analysis

Isolation is ensured by architecture support. We setup different separated memory regions for VMM and VMs, while keeping a ACL table to enable VMI server with necessary access to VMs' memory. This ACL table is controlled by CPU, thus disabling VMM from tampering or snooping on it.

The confidentiality of VMI applications is ensured by moving them into enclaves, which is provided by Intel SGX feature. This can disable the possible vulnerabilities of the VMI server which might affect the VMI applications as it might be compromised by an attacker. The confidentiality of communications is ensured using a modified SSL protocol. This can disable the VMM from replaying the VMI requests or snooping the VMI results.

Integrity is measured when the VM is created and can be attested in the runtime by user. The details are the same with HyperWall, so we will not discussed

them again as our main job is to enable VMI in untrusted cloud, instead of integrity protecting.

Availability is not a goal of our architecture. Anyhow, availability is what the cloud provides, without it there is no chance for the survival of the CSP.

7 Performance Evaluation

The main additional performance cost by our architecture is introduced by SGX encryption and ACL of memory. The encryption cost of SGX MEE (memory encryption engine) is evaluated by Intel, which is between 2.2% to 12%, with average 5.5% for SPECINT 2006 test [10].

As the ACL of memory happens only during VMI operations, VM creation, interruption and termination, it will have little impact on the overall CPU performance during VM running. As our ACL mechanism is similar to that of Hyperwall, the access overhead is similar too. In the worst case, it would introduce 36% additional memory accesses during hypervisor boot, but it will only introduce less than 15% additional memory accesses during the whole life cycle of VMs [3].

However, our architecture will cost more memory (DRAM) space to store the ACL table than Hyperwall. Each ACL entry is 14 bytes large as shown in Table 1, for a physical server with 32 GB memory and 4 KB page size, there would need $32\,GB/4\,KB * 14\,byte = 112\,MB$ space to store the ACL, compared with 4MB of Hyperwall [3]. As the DRAM is cheap enough, this storage cost is affordable.

Overall, our architecture will introduce less than 20% additional overhead to enable VMI operations while in the same time enforce access control from VMM to VMs. The practical performance cost depends on the type of VMI applications and VMI frequencies.

8 Conclusion

We designed a controllable VMI architecture on untrusted cloud. Relative works either totally prevent the VMI operations when the cloud is untrusted or enable VMI operations when the cloud is trusted. In fact, with new hardware support such as SGX, we are able to trust the VMI applications while distrust the VMM layer. Besides, we use our former proposed disjunctive VMI model [9] to provide VMI as a service.

We integrated SGX into HyperWall, so as to accomplish a two layer security. The first layer is the isolation provided by architecture support of VM memory and VMM memory, thus disabling VMM from introspecting VM. The second layer is the isolation between VMI applications and other components of the VMI server, including the OS kernel and other applications. The second layer is used to protect the key security tools from being tampered by malware on the same VM, which is often the case when there are system vulnerabilities.

We modified HyperWall to better support controllable VMI and audit. We add two key fields, ID and count, with the former one controls whether a page

is able to be accessed by a VMI server, the later one records the introspection behavior so as to provide attestation to user. We enhanced the original secure communication channel by introducing an additional session key which is randomly changed during each VMI requests, thus disabling forged VMI requests or replay attacks by VMM.

9 Future Work

As Sect. 7 analyzes, the overhead is mainly caused by two aspects, namely encryption and decryption of enclave memory, and ACL policy enforcement. However, implementation and quantitative performance analysis are still needed to further evaluate our architecture's performance in practical usage. Because of the lack of open source simulation tool for this comprehensive architecture, we need to further investigate similar works and modify related simulators to analysis our work in quantity.

References

1. Garfinkel, T., Rosenblum, M., et al.: A virtual machine introspection based architecture for intrusion detection. In: NDSS, vol. 3, pp. 191–206 (2003)
2. Yao, F., Campbell, R.H.: CryptVMI: encrypted virtual machine introspection in the cloud. In: 2014 IEEE 7th International Conference on Cloud Computing (CLOUD), pp. 977–978. IEEE (2014)
3. Szefer, J., Lee, R.B.: Hardware-enhanced security for cloud computing. In: Jajodia, S., Kant, K., Samarati, P., Singhal, A., Swarup, V., Wang, C. (eds.) Secure Cloud Computing, pp. 57–76. Springer, Heidelberg (2014)
4. Xia, Y., Liu, Y., Chen, H.: Architecture support for guest-transparent vm protection from untrusted hypervisor and physical attacks. In: 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013), pp. 246–257. IEEE (2013)
5. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for CPU based attestation and sealing. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, p. 10 (2013)
6. Baumann, A., Peinado, M., Hunt, G.: Shielding applications from an untrusted cloud with haven. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI) (2014)
7. Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., Russinovich, M.: VC3: Trustworthy data analytics in the cloud using SGX (2015)
8. Dinh, A., Saxena, P., Chang, E.-C., Ooi, B.C., Zhang, C.: M2R: Enabling stronger privacy in mapreduce computation. In: 24th USENIX Security Symposium (USENIX Security 2015), Washington, DC (2015)
9. Shi, J., Yang, Y., Li, C.: A disjunctive VMI model based on XSM. In: The Fifth International Symposium on Cloud and Service Computing (2015)
10. Gueron, S.: A memory encryption engine suitable for general purpose processors, Cryptology ePrint Archive, Report 2016/204 (2016). http://eprint.iacr.org/2016/204