Cerberus: A Novel Hypervisor to Provide Trusted and Isolated Code Execution

Chen Wen-Zhi, Zhang Zhi-Peng, Yang Jian-Hua, and He Qin-Ming
College of Computer Science and Technology
Zhejiang University
Hangzhou, China
Email: {chenwz, zhangzp, yangjh, heqm}@zju.edu.cn

***Abstract*—Cerberus is a tiny x86 virtual machine monitor. It allows security sensitive codes to be executed in an isolated circumstance. The codes could attest their integrity to a remote party by a two-step attestation provided by Cerberus. Cerberus does not require the security sensitive applications to be modified or recompiled to run on it. These applications are packaged with the operating systems as virtual appliances (VA). The on-disk VA files are read-only to simplify the attestation process. Any storage file is sealed to the corresponding secure domain. Cerberus leveraged the nested paging technology to isolate the memory regions efficiently. And it also introduced a novel secure display sharing technology. It can guarantee the security property even when the attackers get control of everything but the core hardware infrastructures. Our performance experiment results show that the overhead introduced by Cerberus is less than 5%.**

Keywords—Virtual Machine Monitor; Code Integrity; Code Attestation; Isolated Codes Execution; Secure Display Sharing

I. INTRODUCTION

Commodity operating systems are pervasively used in home, commercial companies, governments and military settings. They tend to contain increasingly valuable information for personal or/and corporations. Unfortunately, the security facilities that they provide are not always adequate for protecting the sensitive data against various attackers [1, 2]. Moreover, the main-streaming commodity operating systems, such as Windows and Linux are becoming larger and more complex, which makes the security flaws inevitable in OS software.

Unfortunately, traditional approaches[1, 3] are either too complicated to apply[4] or are not enough functional to attest itself to a remote party, some implementations even have some security flaws[5-8].

We offer an alternative named Cerberus. Cerberus allows secure sensitive codes to be executed in an isolated circumstance from the main domain, and could be attested by a remote party. This protects the secure sensitive codes and their saved sensitive data against the malicious codes in the main domain, e.g. the kernel rootkits. Cerberus can guarantee this property even against attackers who get control of everything but the core hardware infrastructures, i.e. the main chip, the trusted platform module (TPM)[9, 10], the CPU, the memory controller and the system memory chips.

Cerberus is composed of the underlying virtual machine monitor (VMM) and a kernel driver running on the main domain. The kernel driver is used to allocate memory from the main domain, just like the balloon driver[11] does, and communicates with the VMM.

II. THEREAT MODEL

Cerberus protects codes and data of the secure domain from being subverted and read from the main domain.

We assumed that intruders can attack any entity in the main domain, i.e. they can subvert and take over complete control of the main domain. In this case, they can execute any instructions in the main domain both in privileged level and user level. These attacks can bypass security functionalities in the main domain. Above and beyond that, an intruder can hide any process to the main domain.

We also assumed that the VMM is transparent to the attackers from the main domain. This means that an attacker to the main domain can't subvert the underlie VMM through the main domain. If the VMM is bypassed or modified in early stage of booting time, it will be aware to the remote party by remote attestation. The attackers could neither cheat in the remote attestation nor retrieve the information saved in the encrypted files which are sealed with the VMM's state by hardware TPM.

III. ARCHITECTURE VIEW OF CERBERUS

This section presents the architecture of Cerberus in a nutshell. Details will be described in the next section.

Cerberus is composed of the underlying virtual machine monitor (VMM) and a kernel driver running on the main domain. The kernel driver is used to allocate memory from the main domain, just like the balloon driver does, and communicates with the VMM. The Architecture of Cerberus is as Fig.1 illustrates.

Cerberus leverages the nested paging mechanism to isolate the memory regions of main domain and the secure domain. When the platform starts, Cerberus map the whole memory but the memory region Cerberus itself resides to the main domain by setting the nested page table of main domain. Cerberus allocates memory space for the secure

978-0-7695-4132-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ISME.2010.172

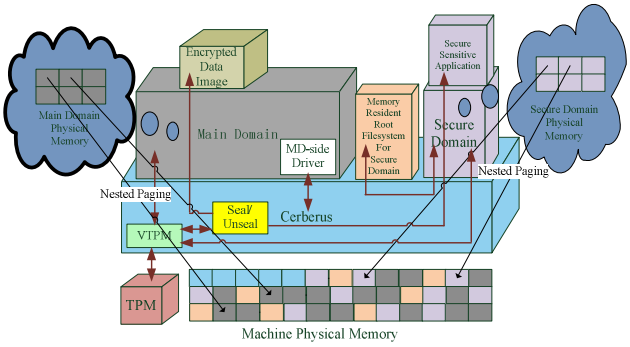

Figure 1. Architecture view of Cerberus

domain in its first execution and unmapped these memory regions from the nested page table of the main domain with assist of the main domain side drive.

Because the secure domain is always trimmed very compact, we design the secure domain's root filesystem to reside on the memory to enhance performance and security. Thus the filesystem for secure domain is volatile, every time it shuts down, the data it saved in its filesystem will be cleared. Cerberus provides the secure domain with a disk image to save its data. The disk image is implemented as a file in the main domain's file system and any data content saved/read from that file is sealed/unsealed by Cerberus transparently.

The virtualization of TPM is implemented in VMM, i.e. Cerberus. It catches the i/o ports and memory mapped i/o (MMIO) memory region accesses in main domain and the secure domain and emulates the hardware TPM operations.

The display sharing is implemented by mapping the VESA mode video memory to different domains to demarcate display regions of main domain and the secure domain.

IV. Design and Implementation

A. Cerberus Memory Management

Cerberus leverages the nested paging mechanism to manage system memory in an efficient way.

Cerberus resides in the top 16MB memory of the system physical memory space. When the platform starts, Cerberus set the nested page table of main domain to map to the whole machine memory space but the memory region that Cerberus itself resides and the MMIO space of the hardware TPM. The layout of the system physical memory is shown in Fig. 2.

The nested page table[12, 13] is set as that the entry's value is equal to the physical address of main domain with 4K size aligned; only the addresses from 0 to top address minus 16MB have nested page table entries. The P bit of the nested page table entry is set to 0, if the address is in the MMIO region of the hardware TPM. Thus if these addresses are accessed, a nested page fault exception will occur. Cerberus could catch this exception and handle it. The nested page fault handler will check whether the address is in the MMIO region of the hardware TPM. If the exception is caused in this situation, Cerberus will invoke the virtual

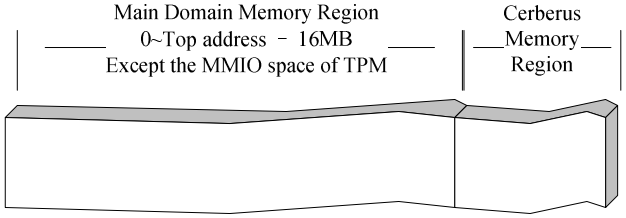

Figure 2. Layout of the system physical memory before secure domain starts

TPM routine to emulate the TPM operations.

When Cerberus starts a secure domain, it allocates memory space from main domain with the main domain side driver and set the P bit of the corresponding nested pagetable entry to 0. After this, Cerberus builds the nested pagetables of the secure domain and its root filesystem respectively to map to the memory space ripped from the main domain. It does not map all ripped memory. It reserves some memory for access violation pool. In the view of the main domain, the ripped memory space is allocated to the main domain side driver and shouldn't access it in normal situation. If the main domain tries to access such a memory space, owing to the P bit of the nested pagetable entry being set to 0, a nested page fault exception will occur and Cerberus could catch it. In this situation, Cerberus will map the nested pagetable entry to a page from the access violation pool if the pool isn't empty.

When secure domain is shut down, Cerberus erases the memory the secure domain and its root filesystem have used. After the erasure, the main domain side driver releases the memory it has allocated and Cerberus will return the memory to the main domain.

B. Secure Display Sharing

If the secure domain is displayed in a normal way, e.g. through the vncviewer in the main domain, a malicious program running on the main domain could retrieve the sensitive information as easy as falling a log. All it has to do to grab the secrets displayed on secure domain is just to print the whole screen and send it back to the intruder. To avoid the information leaking when displayed, Cerberus provides a secure display sharing approach.

When a secure domain is created, Cerberus sets the video card work in VESA mode with the function number 0118h, which is 1024*768 resolution and 32bit per pixel. Thus each row on the display is mapped to the video memory as 4KB size, which equal to one page size. Meanwhile the main domain side driver registers a dialog window with 1024 pixels width on the main domain and sends its location to Cerberus. Cerberus maps the corresponding video memory of the dialog to the secure domain and other video memory to the main domain by modifying their nested pagetables. Beyond and above that, Cerberus also maps some system memory from its pool to the nested pagetables to complete the domain video MMIO region. For example, if the dialog occupies row number 300-500 on the display, Cerberus maps the corresponding video pages of these rows and other 567 (equals to 768 minus 201) pages from its pool to the secure

domain video card MMIO region. The mapping to the main domain is similar.

Only the pages that mapped to the machine video card MMIO will be presented on the display. Hence, both main domain and secure domain could only display parts of their graphic contents. They could change the displayed graphic region dynamically. Cerberus provides two approaches to change the displayed region of the domains. One way is to change the dialog's location in main domain. And the other way is to modify the display_startrow variable, which defines the first row to be shown of the secure domain.

The screenshot of display sharing is presented as Fig.3.

After the video mode is set to VESA function number 0118h, Cerberus modify the main domain's VMCS to prevent it from subverting the machine video settings.

C. *Virtualization of TPM*

We implemented the virtualization of TPM in Cerberus for performance and security concerns. Both main domain and the secure domain have a vTPM structure, which contains the TPM emulation variables, such as the virtual Platform Configuration Register (PCR) and key pairs. These structures are sealed with the PCR state of the machine.

Cerberus exploits both the hardware and virtual TPM to seal data when store the data of secure domain to the nonvolatile storage device, as Fig. 4 shows.

The virtual TPM generates seal key for the secure domain and seals the data using this key pair. Cerberus will unseal the data only for the secure domain that seals it. The key pair itself is sealed by the hardware TPM with the platform state when Cerberus is running. Hence, the data will be unsealed only for the right secure domain running on Cerberus.

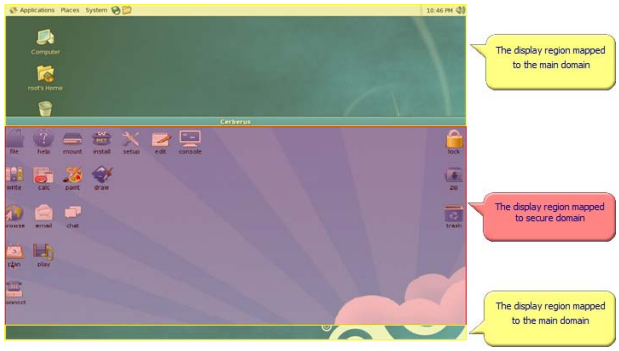

Figure 3. Screenshot of display sharing.

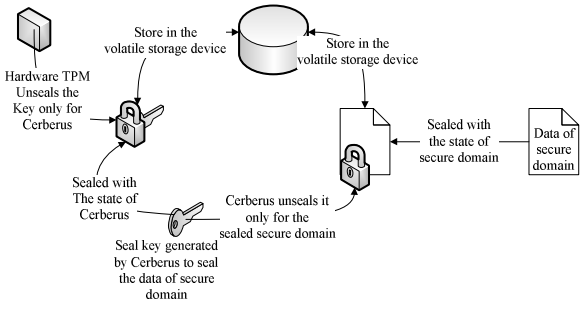

Figure 4. Sealing protocol when storing Data of secure domain.

The remote attestation procedure is similar to the data sealing, both Cerberus and the secure domain will be attested. Cerberus is attested by the hardware PCRs signed by the AIK of the hardware TPM, while the secure domain is attested by the virtual PCRs signed by the AIK of the virtual TPM.

V. PERFORMANCE EVALUATION

These experiments were conducted on an assembly computer configured with a MSI 785GTM-E45 main board, an AMD Phenom X4 9100e CPU, two Apacer DDR2 667MHZ memory chips and a Seagate 250GB 5400RPM disk running CentOS 5.2 as the main domain, Puppy 4.3.1 as the secure domain and CentOS 5.2 without Cerberus as native.

Tab .1 presents the results of process related operations.

Null call is a simple system call operation that retrieves the current process ID; null IO is a simple I/O read/write operation; stat is the operation to get a file stat; Fork proc is the operation that forks a new process and exits immediately; exec proc forks a new process and executes "execve" before exit; sh proc forks a new process and executes the shell program before exit.

The overhead is mainly caused by the nested paging mechanism which makes the address translation drag on. This experiment shows that the overhead introduced by Cerberus is less than 5%. The performance of main domain is competitive to the native OS. The overhead in secure domain is more than the main domain but still acceptable.

Tab .2 presents the result of context switching test.

2p/16K means that the workload is 2 processes handling 16K data concurrently; 8p/64K means that the workload is 8 processes handling 64K data concurrently and etc.

The overhead is mainly caused by the vm-exit and vm-enter events when schedule domains. This experiment presents that the overhead introduced by Cerberus in context switching is slight. It is also less than 5%. Both the performances of main domain and the secure domain are competitive to the native OS, which runs on a nonvirtualized environment.

TABLE I. PROCESSES OPERATIONS COST TIME (MICROSECONDS)

Test environment	NULL CALL	NULL IO	STAT	Fork proc	Exec proc	Sh proc
Native	0.37	0.74	4.65	172	499	2253
Main Domain	0.42	0.75	4.93	195	531	2495
Secure Domain	0.45	0.81	5.02	229	554	2608

TABLE II. CONTEXT SWITCHING COST TIME (MICROSECONDS)

Test environment	2p/0K	2p/16K	2p/64K	8p/16K	8p/64K
Native	1.86	2.03	9.27	4.26	11
Main Domain	2.02	2.15	9.66	4.31	11.6
Secure Domain	2.14	2.13	9.54	4.5	12.4

VI. CONCLUSION AND FUTRURE WORK

In this paper, we present Cerberus, a tiny hypervisor designed to provide trusted and isolated code execution based on hardware virtualization technologies.

Cerberus protects the secure sensitive codes against the malicious codes in the main domain by executing it in a trusted and isolated environment. By leveraging the latest hardware virtualization support, using the hardware TPM and virtualized TPM to attest the integrity of Cerberus itself and the secure domain respectively to a remote party and employing the display sharing technique, Cerberus has many advantages over previous works, such as easily adoptable, remote attestation and high assurance. Our performance experiments show that the overhead introduced by Cerberus is less than 5%. Both the main domain and the secure domain running on Cerberus have a performance competitive to the native operating systems executing on a nonvirtualized environment.

We fixed the display resolution to 1024 X 768 in Cerberus for performance concerns. It may be inconvenient for some situations. We are investigating some way to efficiently share the display in a wide range of resolutions.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (60970125) and the Major State Basic Research Development Program of China (2007CB310900)

REFERENCES

[1] Criswell, J., Lenharth, A., Dhurjati, D., and Adve, V.: 'Secure virtual architecture: A safe execution environment for commodity operating systems'. Proc. SOSP'07: 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, United states, October 14-17 2007, pp. 351-366, DOI:10.1145/1294261.1294295

[2] Seshadri, A., Luk, M., Qu, N., and Perrig, A.: 'SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes'. Proc. SOSP'07: 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, United states, October 14-17 2007, pp. 335-350, DOI:10.1145/1294261.1294294

[3] Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J., and Ports, D.R.K.: 'Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems'. Proc. Proceedings of the 13th international conference on Architectural support for programming languages and operating systems, Seattle, WA, USA, March 2008, pp. 2-13, DOI:10.1145/1346281.1346284

[4] Singaravelu, L., Pu, C., Hartig, H., and Helmuth, C.: 'Reducing TCB complexity for security-sensitive applications: three case studies'. Proc. Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, Leuven, Belgium, 2006, pp. 161-174, DOI:10.1145/1217935.1217951

[5] Wojtczuk, A.T.a.R.: 'Introducing Ring -3 Rootkits '. Proc. Blach Hat USA, Las Vegas, NV , USA, 2009

[6] Rutkowska, R.W.a.J.: 'Attacking Intel® Trusted Execution Technology'. Proc. Black Hat DC, Washington, DC , USA, 2009

[7] J Rutkowska, R.W.: 'Detecting & Preventing the Xen Hypervisor Subversions'. Proc. Black Hat USA, Las Vegas, NV, USA, 2008

[8] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A.: 'Xen and the art of virtualization'. Proc. SOSP'03: Proceedings of the 19th ACM Symposium on Operating Systems Principles, Lake George, NY, United states, October 19-23 2003, pp. 164-177, DOI: 10.1145/1165389.945462

[9] TCG Group, 'TCG Architecture Overview, Version 1.4'

[10] TCG Group, 'TCG Design, Implementation, and Usage Principles (Best Practices)'

[11] Waldspurger, C.A.: 'Memory resource management in VMware ESX server', SIGOPS Oper. Syst. Rev., 2002, 36, (SI), pp. 181-194, DOI: 10.1145/844128.844146

[12] AMD Corp., 'AMD-V™ Nested Paging White Paper'

[13] Intel Corp., 'Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide'