2009 Third International Conference on Emerging Security Information, Systems and Technologies

A Two-step Execution Mechanism for Thin Secure Hypervisors

Manabu Hirano
Dept. of Information and Computer Engineering, Toyota National College of Technology
2-1 Eisei, Toyota, Aichi, Japan
hirano@toyota-ct.ac.jp

Takahiro Shinagawa, Hideki Eiraku, Shoichi Hasegawa, Kazumasa Omote*,
Koichi Tanimoto, Takashi Horie, Seiji Mune, Kazuhiko Kato
Dept. of Computer Science, Graduate School of ISE, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
{shina, hdk, s-hase, komote, tanimoto, horietk, mune}@osss.cs.tsukuba.ac.jp, kato@cs.tsukuba.ac.jp

Takeshi Okuda, Eiji Kawai, Suguru Yamaguchi
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, Japan
{okuda, eiji-ka, suguru}@is.naist.jp

Abstract

Virtual Machine Monitors (VMMs), also called hypervisors, can be used to construct a trusted computing base (TCB) enhancing the security of existing operating systems. The complexity of a VMM-based TCB causes the high risk of security vulnerabilities. Therefore, this paper proposes a two-step execution mechanism to reduce the complexity of a VMM-based TCB. We propose a method to separate a conventional VMM-based TCB into the following two parts: (1) A thin hypervisor with security services and (2) A special guest OS for security preprocessing. A special guest OS performing security tasks can be executed in advance. After shutting down the special guest OS, a hypervisor obtains preprocessing security data and next boots a target guest OS to be protected. Thus, the proposed two-step execution mechanism can reduce run-time codes of a hypervisor. This paper shows a design, a prototype implementation and measurement results of lines of code using BitVisor, a VMM-based TCB we have developed.

1. Introduction

Computer systems in current governmental and commercial organizations are processing massive amounts of data every day. As a result, these organizations are facing with the high risk of information leak cases. This paper employs Virtual Machine Monitors (VMMs), also called hypervisors, as a key component to enhance the security of existing operating systems. A VMM is a technology to encapsulate an operating system, which was originally developed for mainframe computers [1]. An ideal VMM technology offers complete isolation of virtual machines (VMs) [2]. Orange Book [3], a classic computer security standard of US government, defines that *Trusted Computing Base (TCB)* contains all of the elements of the system responsible for supporting the security policy and supporting the isolation of objects (code and data) on which the protection is based. Thus, this paper employs a VMM technology to construct a TCB.

Many organizations and researchers have proposed VMM-based security mechanisms. For example, NetTop and Terra provide VMM-based security functions like encrypted communication channel and storage encryption [4, 5]. sHype developed by IBM research supports MAC (Mandatory Access Control) policies for virtualized servers [6]. In many cases, these security mechanisms employ existing commodity VMM software like VMware [7] and Xen [8] to prove the concept. Although commodity VMMs provide general-purpose properties and high usability, they require a certain degree of complexity. For example, Xen hypervisor has 100 KLOC (Kilo lines of code) [9] and the VMkernel of VMware ESX server has 200 KLOC [10]. The large size and high complexity of software sometimes

*He currently belongs to Japan Advanced Institute of Science and Technology.

978-0-7695-3668-2/09 $25.00 © 2009 IEEE
DOI 10.1109/SECURWARE.2009.27

causes their poor testability and high vulnerability [11].

The complexity of hypervisors is not preferable to construct a TCB. Therefore, some researchers have proposed tiny hypervisors specialized for security-purpose. Derek et al. propose a mechanism to reduce the complexity of Xen hypervisor to construct a TCB [9]. SecVisor [12] is developed as a tiny hypervisor that ensures code integrity for commodity OS kernels. SecVisor has small code size, only 1112 LOC (Lines of code), for the run-time portion using CPU-supported virtualization. Lei et al. shows a small hypervisor called Palacious VMM [13]. Palacious VMM hooks I/O operations between device drivers on a guest OS and physical hardware. The core of Palacious VMM has 20 KLOC and the additional part to hook I/O operations has 10 KLOC. As described above, the code size of a TCB is one of the important aspects to evaluate whether reliable security mechanisms or not.

Although tiny hypervisors described above [12][13] provide basic security functions, they do not provide enough functions to enfoece security policies in end-point computers. To increase the security of end-users' computers in governmental and commercial organizations, we have developed a novel thin hypervisor called *BitVisor* specialized for I/O device security [14]. BitVisor provides transparent security services like storage encryption, encrypted communication channel and ID management framework in the VMM layer. For example, BitVisor can authenticate a user using her or his smart card and the authenticated ID can be used to enforce security policies like storage encryption for USB thumb drives to prevent information leak cases. The core run-time portion of BitVisor (version 0.7) has approximately 27 KLOC only. However, these optional security services increase the code size of the entire VMM-based TCB.

This paper proposes a two-step execution mechanism to reduce the run-time codes of a hypervisor. Althogh some researchers propose a design of a thin hypervisor with a single security function [12] [13], we propose thin hypervisor with multiple and complex security functions like user authentication using PKI-based smart cards, VPN and storage encryption. Our proposal intends to reduce the complexity of such advanced security services in a VMM layer. The basic concept of our proposal will help to construct more reliable VMM-based TCBs to enforce security policies for end-users' computers in organizations.

2 Challenges to a Thin Secure Hypervisor

This section first shows basic architecture and security services of BitVisor. Then, we show a problem in the complexity of its ID management framework.

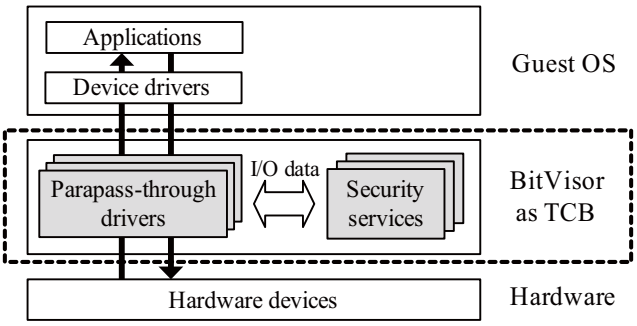

Figure 1. Basic architecture of BitVisor

2.1 Overview of BitVisor Architecture

Figure 1 shows basic architecture of BitVisor. BitVisor is a thin hypervisor to provide I/O device security [14]. Special drivers called *parapass-through driver* in BitVisor work as reference monitors. *Parapass-through drivers* intercept the data I/Os and they can apply transparent security services to them. These security services are executed in the VMM layer. Even if an attacker compromises a guest operating system worked on a VM, the VMM layer of BitVisor can enforce these security services certainly. In addition, *parapass-through drivers* check the control I/Os to prevent attacks to the hypervisor itself. Users can choice the bare minimum set of *parapass-through drivers* to reduce the size of the entire TCB.

Figure 2 shows security components of BitVisor architecture. BitVisor consits of core hypervisor, parapass-through drivers and optional security services. For example, a parapass-through driver for Intel Pro/1000 NIC can enforce an IPsec-VPN (Virtual Private Network) service to communication channels. A parapass-through driver for a USB UHCI (Universal Host Controller Interface) and a MSD (Mass-storage class device) can enforce an XTS-AES based storage encryption service to USB thumb drives. BitVisor can link these optional security services as needed onto the run-time portion. Therefore, BitVisor architecture can keep the minimum code size of the entire TCB.

2.2 The Complexity of the ID Management Framework

BitVisor also supports ID management framework [15]. The ID management framework is a key component of BitVisor that offers user authentication and authorization services using PKI-based smart cards. The ID management framework provides a user authentication function to VPN, secret key management to storage encryption, a user ID based access control mechanism to boot guest OSs and a user ID based security policy enforcement mechanism for

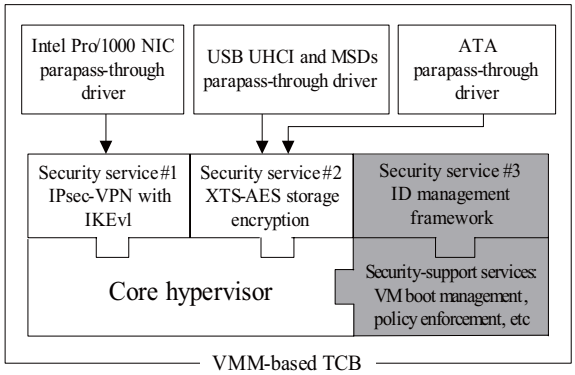

Figure 2. Security components of BitVisor

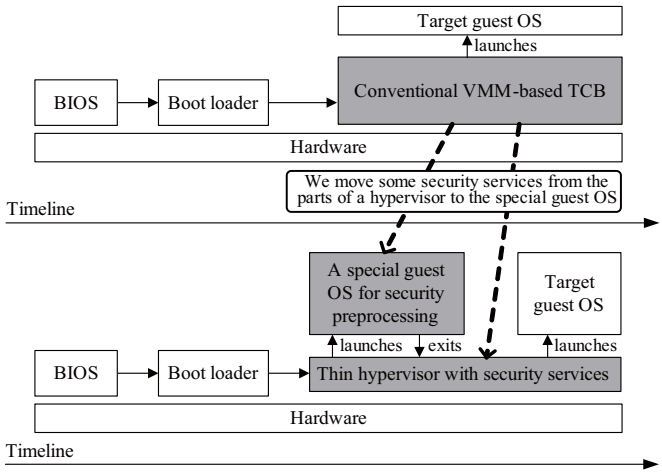

Figure 3. Basic concept of a two-step execution mechanism

each security hook. Moreover, we have proposed a Role-based Access Control (RBAC) extension [16]. As described above, the ID management framework of BitVisor provides useful security functions in the VMM layer. However, it must support smart card drivers, PKCS#11 and other public key cryptographic libraries, therefore it has a complex structure. As shown in Figure 2, a security service of ID management also linked in the run-time of BitVisor. Therefore, we have to reduce the code size of the ID management framework to ensure the reliability of BitVisor as a VMM-based TCB.

3 A Two-step Execution Mechanism for Thin Secure Hypervisors

We assume that a VMM-based TCB like BitVisor is used for foundation of security policy enforcement for distributed end-point computers in organizations. Therefore, the TCB must be small code size to achieve high testability and reliability. To reduce the code size of the TCB, this paper presents a separation method of security processes. Figure 3 shows a basic concept of the proposed two-step execution mechanism. We separate a conventional VMM-based TCB into the following two parts: (1) A thin hypervisor with security services and (2) A special guest OS for security preprocessing. A special guest OS performing security tasks can be executed in advance, then a thin hypervisor obtains preprocessing security data. Finally, a thin hypervisor with run-time security services boots a target guest OS to be protected.

To apply the proposed two-step execution mechanism, we have to distinguish whether security processes can be executed before or not. This paper shows an example to distinguish such security processes in the case of BitVisor and its ID management framework described in Section 2.2. Table 1 shows functions of the ID management framework used in BitVisor. BitVisor as a TCB can authenticate users

Table 1. Functions of ID management framework

Security function	Target service	Preprocessing
User authentication (ID/Password, PKI)	VM boot management	OK
User authentication (PKI)	IKEv1 (IPsec-VPN)	Partial
Key management	Storage encryption	Partial
Checking smart card connection	Core hypervisor	NO

based on PKI-based smart cards to control VM boot operations. Each user's smart card can manage storage encryption keys securely. BitVisor also provides a user authentication function based on her or his smart card for an IPsec-VPN service. Moreover, BitVisor checks status of smart card connections periodically to verify the presence of an authenticated user. This mechanism prevents unauthorized accesses to the computer system physically. As shown in Table 1, we can move some parts of security services, both preprocessing is "OK" and "Partial", from the parts of a hypervisor to a special guest OS. As a result, the run-time codes of a hypervisor become small, lightweight and more reliable.

4 Design

Figure 4 shows the process flow of a special guest OS for security preprocessing. BitVisor first boots this special guest OS, which performs prior tasks for future security services in a hypervisor. An initial program on the guest OS

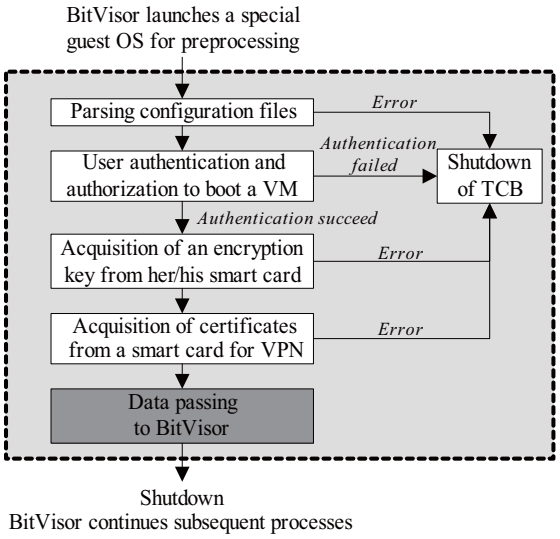

Figure 4. Process flow of a special guest OS for security preprocessing

first parses configuration files of BitVisor. In general, parsing processes of text-based configuration files are complex. Thus, we can remove these unnecessary codes from the run-time portion of a hypervisor. Then, the initial program executes PKI-based user authentication using user's smart cards. If the authentication is failed then the special guest OS shuts down. The initial program next loads a user's secret key and a user's certificate from her or his smart card. They are used in a storage encryption service and user authentication of an IPsec-VPN service after the hypervisor boots a target guest OS. The initial program on the special guest OS has to pass these prior data to BitVisor. Because a guest OS on a VM cannot access to the hypervisor directly, the initial program executes a special instruction to access a hypervisor. We discuss this passing method in Section 5. Finally, BitVisor shuts down the special guest OS and boots a target guest OS to be protected.

Although this paper especially shows an example to reduce the code size of BitVisor and its ID management framework, a basic concept that separates a conventional VMM-based TCB into two parts is useful for many existing VMM-based TCB mechanisms.

5 Prototype Implementation

This section describes a prototype implementation of the proposed two-step execution mechanism using BitVisor. Table 2 shows hardware specification and Table 3 shows software used in the prototype implementation.

Table 2. Hardware specification

Client PC	Intel Core 2 Duo 1.86GHz 512 MB RAM
Smart card	eLWISE TYPE-B ISO/IEC14443, 7816 (NTT Communications)
Smart card reader	ASE drive IIIe ISO/IEC7816 (Athena Smartcard Solutions)

Table 3. Software used in the prototype implementation

VMM-based TCB	BitVisor 0.7
Special guest OS for security preprocessing	Linux kernel 2.6.25.3 Libusb 0.1 OpenSSL 0.9.8g
Target guest OS	Windows XP Professional SP2

5.1 Detailed Flow

Figure 5 shows the detailed flow of the prototype implementation. Our prototype implementation first launches a boot loader (Grub). The target OS image is encrypted by a user's secret key. Therefore, a user has to launch the special guest OS for security preprocessing before executing the target guest OS. The special guest OS provides execution environment with least privilege, therefore it consists of the minimum Linux kernel, the initial RAM disk called *initrd*, the minimum set of libraries and device drivers for smart card readers. The Linux kernel and the RAM disk image are executed on BitVisor. Our proposal assumes that the entire TCB includes both the special guest OS and BitVisor. Therefore, we have to ensure the reliability and authenticity of the special guest OS environment before. The initial process (*init* program) of the RAM disk executes operations shown in Figure 4. The *init* program next performs data passing to BitVisor and finally shuts down the special guest OS without mounting an ultimate file system. After this, a user can execute a target guest OS on BitVisor. BitVisor obtains the user's encryption key from the preprocessing data, decrypts the target guest OS image and boots it.

5.2 Data Passing Between the Separated TCBs

As shown in Figure 5, the *init* program on the special guest OS has to pass data to the BitVisor. However, a guest OS on a VM cannot access VMM functions directly. To invoke a special VMM function for data passing, the *init*

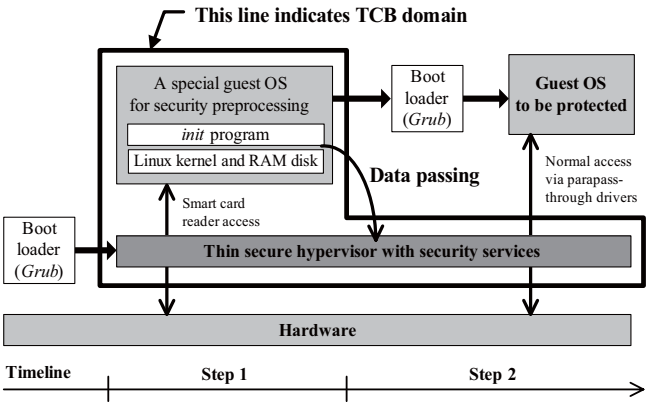

Figure 5. The detailed flow of the prototype implementation

```
0x00100000   MOV (VMM call number for data passing ), %eax
0x00100005   MOV $0x00100010, %ebx
0x0010000A   (VMCALL for Intel-VT or AMD-V)
0x0010000D   INT3    ; The following lines are not executed
0x0010000F   JMP $   ; Infinite loop
0x00000010   (Data to be passed)  ; The following lines are data
```

Figure 6. The codes for kexec system call

program executes `VMCALL` instruction supported by Intel VT [17]. In the prototype implementation, the *init* program also executes *kexec* system call [18] to shut down devices before booting the target guest OS. The *init* program sets up `VMCALL` instruction and the preprocessing data itself using *kexec_load* system call. Figure 6 shows the program codes for the *kexec_load* system call. We use physical memory address from `0x00100000` for the data passing. After setting up the *kexec_load* parameters, the *init* program reboots the special guest OS itself. When the special guest OS is shut down, *kexec* performs codes including `VMCALL` shown in Figure 6. As a result, BitVisor can execute a function to obtain the prior data from physical memory area from `0x00100010`. Finally, BitVisor wipes out the memory area for the data passing and the special guest OS to prevent data stealing.

5.3 Measurement Result of LOC

We have measured lines of code of the ID management framework libraries using *SLOCCount* program [19]. Table 4 shows the measurement results before and after applying the proposed two-step execution mechanism. Application-level libraries like standard I/O API, ID management API [15] and PKCS#11 API [20] become reduced in size. However, lower-level libraries like PC/SC [21] and CCID smart card driver do not become reduced.

Table 4. Measurement result: the code sizes of the improved ID management framework libraries

API name	Before [LOC]	After [LOC]	Rate of reduction [%]
Standard I/O	1,072	352	67.1
ID Management	3,219	657	79.5
PKCS#11	5,520	4,604	16.5
PC/SC	1,965	1,965	0.0
CCID smart card driver	2,467	2,467	0.0
Total	14,243	10,045	29.5

The core run-time portion of BitVisor (version 0.7) has 27,214 LOC. The original code size of ID management libraries is 14,243 LOC. As shown in Table 4, by applying the proposed two-step execution mechanism, we have reduced 4,198 LOC (29.5 %) from the original ID management libraries. The code size of the core run-time portion of BitVisor with the original ID management libraries was 41,457 LOC. The code size of the core run-time portion of BitVisor with the ID management libraries applying the proposed two-step execution mechanism has become 37,259 LOC. Thus, we have been able to reduced 10.1 % of LOC in the total run-time portion of BitVisor.

6 Discussion

The basic concept of the proposed two-step execution mechanism that separates preprocessing tasks is simple. Therefore, we will be able to apply the proposal to other hypervisors with complex security functions like ID management libraries of BitVisor. When we implement the proposal to other hypervisors, we have to consider the following issues: (1) secure data passing method between a special guest OS and a hypervisor, (2) a run-time protection for a special guest OS and a hypervisor, (3) a protection for the boot sequence like the measured launch mechanism, and (4) a mechanism to guarantee the authenticity of each TCB component.

In particular, the proposed mechanism has to guarantee the boot sequence of a boot loader, a hypervisor, a special guest OS for security preprocessing and a target OS as shown in Figure 5. Moreover, we have to verify the authenticity of a boot loader, a hypervisor and a special guest OS we have proposed. We can employ Intel Trusted Execution Technology (TXT) [22] to guarantee a boot sequence and authenticity of each software. In future work, we will

apply the proposed two-step execution mechanism to other VMM-based TCBs to show the usability.

The data passing part is the most important process of the proposed two-step execution mechanism. If an attacker can compromise the data passing part, the attacker can executes a user's guest OS and they can steal its data. Especially, it is important to protect encryption key of users' guest OS images. Therefore, the memory area of the data passing has to be zero-cleared after booting the target guest OS or resetting the hardware.

7 Conclusion

This paper has shown a two-step execution mechanism to reduce the code size of a VMM-based TCB. The proposed mechanism separates a conventional VMM-based TCB into the two parts: (1) A thin hypervisor with security services and (2) A special guest OS for security preprocessing. This paper has introduced an additional TCB domain for security preprocessing as the special guest OS works on a hypervisor. This paper also presented a design, a prototype implementation and measurement results of lines of code using BitVisor we have developed. The measurement results of the prototype implementation indicate that the proposed mechanism can reduce approximately 10.1 % of LOC in the total run-time portion of BitVisor. Although this paper has shown an example to reduce the code size of BitVisor and its ID management framework, a basic concept that separates a conventional VMM-based TCB into two parts is useful for many existing VMM-based TCB mechanisms. In future work, we should consider the detailed security of the data passing mechanism between a special guest OS and a hypervisor.

Acknowledgments

This work is supported by Special Coordination Funds for Promoting Science and Technology of Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] L. Seawright and R. MacKinnon. VM/370 - a study of multiplicity and usefulness. *IBM Systems Journal*, pages 4–17, 1979.

[2] Stuart E. Madnick and John J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation, Proceedings of the workshop on virtual computer systems. In *ACM Press*, pages 210–224, 1973.

[3] DEPARTMENT OF DEFENSE STANDARD. Epartment of defense trusted computer system evaluation criteria. *DoD 5200.28-STD*, Dec 1998.

[4] Meushaw, R. and D. Simard. NetTop: Commercial Technology in High Assurance Applications. In *National Security Agency Tech Trend Notes*, 2000.

[5] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 193–206, 2003.

[6] Reiner Sailer, Trent Jaeger, Enriquillo Valdez, Ramon Caceres, Ronald Perez, Stefan Berger, John Linwood Griffin, Leendert van Doorn. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *Annual Computer Security Application Conference*, pages 276–285, 2005.

[7] VMware. http://www.vmware.com/.

[8] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[9] Derek Gordon Murray, Grzegorz Milos, and Steven Hand. Improving Xen security through disaggregation. In *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 151–160, New York, NY, USA, 2008. ACM.

[10] VMware. VMware ESX Server Virtual Infrastructure Node Evaluators Guide, 2005. http://www.vmware.com/pdf/esx_vin_eval.pdf.

[11] Lenin Singaravelu, Calton Pu, Hermann Härtig, and Christian Helmuth. Reducing TCB complexity for security-sensitive applications: three case studies. In *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 161–174, New York, NY, USA, 2006. ACM.

[12] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 335–350, New York, NY, USA, 2007. ACM.

[13] Lei Xia, Jack Lange, and Peter A. Dinda. Towards Virtual Passthrough I/O on Commodity Devices. In Muli Ben-Yehuda, Alan L. Cox, and Scott Rixner, editors, *Workshop on I/O Virtualization*. USENIX Association, 2008.

[14] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato. BitVisor: A Thin Hypervisor for Enforcing I/O Device Security. In *Proc. the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, March 2009.

[15] Manabu Hirano, Takeshi Okuda, Eiji Kawai and Suguru Yamaguchi. Design and Implementation of a Portable ID Management Framework for a Secure Virtual Machine Monitor. *Journal of Information Assurance and Security (JIAS), Dynamic Publishers*, 2:211–216, 2007.

[16] Manabu Hirano, Takahiro Shinagawa, Hideki Eiraku, Shoichi Hasegawa, Kazumasa Omote, Koichi Tanimoto, Takashi Horie, Kazuhiko Kato, Takeshi Okuda, Eiji Kawai, and Suguru Yamaguchi. Introducing Role-based Access Control to a Secure Virtual Machine Monitor: Security Policy Enforcement Mechanism for Distributed Computers. In *Proc. the IEEE International Workshop on Dependable and Secure Services Computing 2008*, Dec. 2008.

[17] Intel Corporation. Intel Virtualization Technology Specification for the IA-32 Intel Architecture, April 2005.

[18] Andy Pfiffer. Reducing System Reboot Time with kexec. http://developer.osdl.org/.

[19] David A. Wheeler. SLOCCount. http://www.dwheeler.com/sloccount/.

[20] RSA Laboratories. PKCS #11: Cryptographic Token Interface Standard v2.20, 2001.

[21] PC/SC Workgroup. PC/SC Workgroup Specifications 2.01.5, 2008.

[22] Intel Corporation. Intel Trusted Execution Technology Software Development Guide, 2008.