

Remote Attestation for HDD Files using Kernel Protection Mechanism

Keisuke Takemori¹, Adrian Perrig², *Members, IEEE*
 Ning Qu², and Yutaka Miyake¹, *Non-members, IEEE*

¹KDDI R&D Laboratories, Fujimino, Saitama, JAPAN

²CyLab/CMU Pittsburgh, PA, USA

A remote attestation that measures files on a hard disk drive (HDD) is important for intrusion detection on a data center server. When the server is infected by a rootkit or when a file measurement application is manipulated, the response of the kernel or the measurement application is not reliable. A trusted platform module (TPM) that achieves a chain of trust from BIOS to kernel upon booting is proposed to provide the remote attestation. However, as the data center server is rarely rebooted, the TPM is ill suited for file measurements of the running server. In this paper, we propose an on-demand remote attestation scheme for HDD files of the server. We designed and implemented a trust chain from the BIOS via the kernel and the file measurement application to the HDD files on a running server for secure integrity measurement. A memory virtualization technique is applied to guarantee the integrity of the running kernel, and the file measurement application is verified using a code signature. Also, we implement a mechanism that attaches the server's signature to a measurement result in a trusted kernel. Finally, our proposed scheme achieves a result whereby the remote verifier can measure the integrity of the server files securely at any time.

Keywords - Remote Attestation, Kernel and File Integrity, Linux Security Module, Chain of Trust

I. INTRODUCTION

INTRUSIONS have now become a critical issue on Internet servers. Hence, the integrity measurement of files has become a popular method of detecting intrusions and manipulations of system files. Here, the integrity measurement is performed to compare hash computations of target files with their legitimate hash list managed by a trusted verifier. In the case of a datacenter server, remote attestation via a communication channel is important to measure file integrity, because most system managers keep watching the status of servers from a trusted external host. However, when the server is infected by a rootkit, the kernel may reply with fake responses. In addition, when a file measurement application or its result is manipulated by intruders, the measurement response becomes unreliable. Detection and prevention schemes against kernel manipulations have been proposed [1-3, 10]. They can only ensure kernel code integrity, but guarantee neither the integrity of HDD files nor their measurement result. A chain of trust from the trusted kernel to the measurement result is important for remote attestation of the datacenter server.

The servers may contain a TPM [4], which can be a root of trust and form a trust chain from BIOS to kernel upon system bootup [5-7]. However, as the data center server manages many types of application files on the HDD and is rarely rebooted, the TPM is ill suited for the file measurement of the running server. Pioneer [8] enables a dynamic root of trust and has been proposed for secure process invocation. A small piece of code is sent from a remote verifier and executed exclusively on the CPU. As the kernel and all applications are suspended and no malicious code can break in the Pioneer execution, a secure process, such as a file measurement application, is the only item invoked and executed. However, it is also ill suited for a datacenter server because Pioneer suspends the kernel and all applications.

In this paper, we propose a remote attestation scheme for files on the HDD. A trust chain from the BIOS via a kernel and a file measurement application to HDD files on the running server is designed and implemented for secure measurement. The integrity of the running kernel is guaranteed by memory virtualization techniques [9-10] that control the kernel memory access against any kernel code manipulation, such as a rootkit. The file measurement application that includes a code signature for an integrity check [11] is also applied. When the file measurement application is invoked, the trusted kernel verifies the integrity of the file measurement application at a check point of a Linux Security Module (LSM) [12] by comparison with the attached code signature. In addition, we design and implement a signature mechanism that attaches a signature to a measurement result in the trusted kernel. The signature of the measurement result will be verified at a remote verifier. Consequently, we form a trust chain from the BIOS to the measurement result and achieve a secure remote attestation for HDD files on datacenter servers that manage many files and are rarely rebooted. Because the file measurement application runs as a user process, it is more easily modified and controlled than the conventional TPM framework.

The rest of this paper is organized as follows. Section II surveys conventional techniques and discusses requirements for remote attestation. Section III discusses related work and Section IV presents the framework of our proposed model. Section V describes our design and implementation of the trust chain from the BIOS to HDD files, which provides trusted measurement and remote attestation schemes. In Section VI, we evaluate the performance on our experimental system and Section VII concludes this paper.

II. CONVENTIONAL TECHNIQUES AND REQUIREMENTS

A. TPM: Remote Attestation

The TPM [1] is a trusted chip that includes cipher functions, i.e., hash chaining, key-pair generation, and signature functions, and append-only registers. A rootkit cannot contaminate the TPM because of its independent processing from the kernel. The TPM is considered trusted.

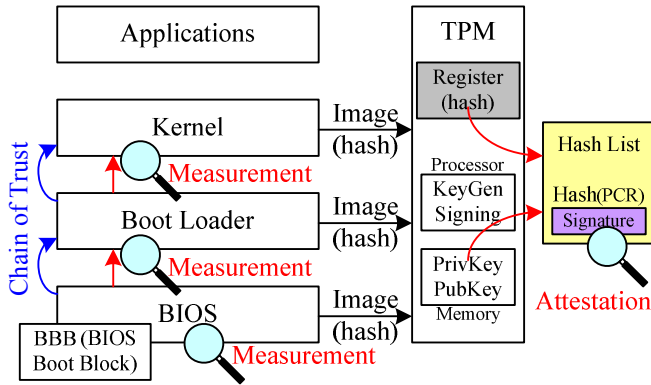


Fig. 1. Measurement and Attestation Using TPM upon System Bootup.

Figure 1 shows a remote attestation scheme using the TPM upon system bootup. The BIOS measures itself using the BBB chip and extends the hash value into the TPM. Also, the BIOS measures the boot loader, extends the hash value into the TPM and invokes the boot loader. The latter then measures the kernel, extends the hash value into the TPM and invokes the kernel. As a result, a chain of trust from the BIOS via the boot loader to the kernel can be made. The hash value of the platform configuration register (PCR) in the TPM is signed by using the private key of the TPM and reported to a remote verifier. The remote verifier verifies the signature and inspects the hash list in order to check the integrity of the kernel.

B. Dynamic Root of Trust

A dynamic root of trust that generates a trust execution framework on an unreliable server is actively researched in order to isolate malicious code [8, 13]. Pioneer is proposed to generate a dynamic root of trust for secure code execution [8]. As Pioneer creates an untampered code execution environment during verification, malicious code cannot interfere with the verification process. In addition, the Turtle framework for security-sensitive code execution has been proposed to achieve a minimal Trusted Computing Base (TCB) [13]. The security-sensitive code, i.e., cipher and sign applications, is extracted from the original application and executed on the TCB. As the execution of the security-sensitive code is independent from both kernel and applications, no malicious code can undetectably tamper with the execution.

C. Requirements: Remote Attestation for HDD Files

As there are many files on the HDD of the datacenter server, i.e., kernel modules, user applications and their configurations, the mechanism for file measurement becomes complicated. It is hard to implement a file measurement application and its

rules using TPM, Pioneer and Turtle since they are designed for small security-sensitive code verification.

As a TPM attests to the integrity of the kernel only upon system bootup, it is ill suited for datacenter servers. When the kernel and system files are manipulated on a running server, the server's responses are unreliable until the next reboot.

When a communication channel between the remote verifier and the server is attacked, the measurement results of the server can be faked. Moreover, datacenter servers are usually highly loaded.

The remote attestation requirements for a datacenter server are as follows:

- The file measurement application that measures HDD files is implemented as a flexible user application.
- The integrity of the file measurement application and the communication service on the server should be verified before execution.
- Both the file measurement application and the communication service are invoked from the non-manipulated kernel.
- The measurement result is signed in a trusted manner before being output to the HDD.
- The additional CPU load for the protection mechanisms of the remote attestation should be minimal.

III. RELATED WORK

A. SecVisor: Lifetime Kernel Code Integrity

SecVisor is a tiny hypervisor providing lifetime kernel integrity by two steps: trusted boot and runtime memory protection mechanisms [10].

The trusted boot mechanism achieves a chain of trust: SecVisor runtime \rightarrow Linux kernel. SecVisor loader starts SecVisor runtime by calling SKINIT, which performs the following steps: 1) initializes the CPU into some known states, 2) protects the target memory used by SecVisor runtime, 3) measures SecVisor runtime automatically and extends the measurement result into TPM, 4) and finally transfers control to SecVisor runtime. Subsequently, SecVisor runtime will further protect the target memory used as the Linux kernel, measure the Linux kernel, and extend the result into the TPM. When the integrity of the Linux kernel is verified, SecVisor runtime transfers control to the same.

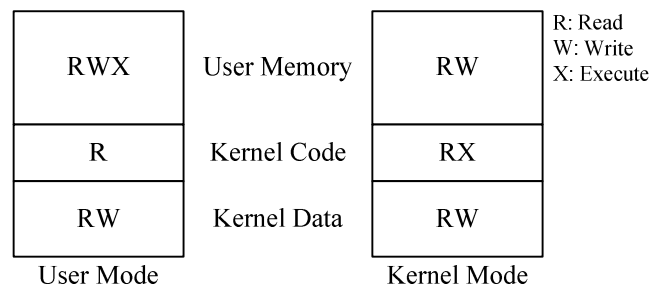


Fig. 2. Dynamic Permission Control for User and Kernel Memories.

The non-manipulated kernel (or trusted kernel) gains control after being verified. The memory protection

mechanism protects the kernel memory space from rootkits and direct memory access (DMA) attacks based on the support of secure virtual machine (SVM) technologies [9-10]. Figure 2 shows the memory permission mapping in user and kernel modes enforced by SecVisor. In kernel mode, only the memory containing the approved kernel code is executable, while in user mode, only application memory is executable. Moreover, in both modes, the approved kernel code is always read-only. Also, SecVisor setups IOMMU to prevent any DMA access to the approved kernel code.

B. DigSig: Trust Application Invoking

The DigSig provides a trusted invoking mechanism for a user application [11]. All applications on the server are attached with code signatures that are used for authentication and verification of code integrity.

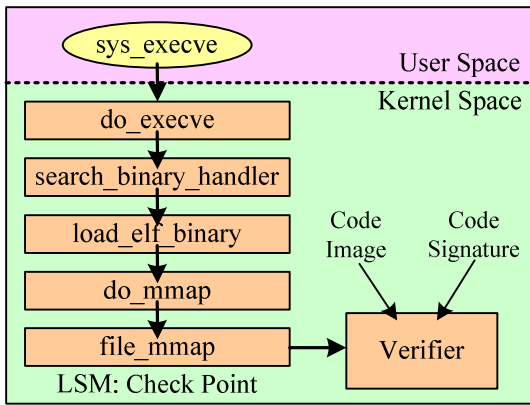


Fig. 3. Integrity Check of Execution Code.

Figure 3 shows the trusted invoking mechanisms of the DigSig. When the `sys_execve` is called in the user space, `do_execve`, `search_binary_handler`, `load_elf_binary`, and `do_mmap` are called one after another in the kernel space. Finally, the `file_mmap`, which is one of the checkpoints of the LSM, is called. Both the code image and the code signature of the user application are loaded onto the `file_mmap` LSM. A verifier hooks the `file_mmap` LSM and checks the integrity of the user application by comparing its code image with its code signature. When the integrity of the code is verified, the user application is invoked.

IV. PROPOSED MODEL

In this section, we describe a remote attestation framework that can measure the HDD files at runtime on a datacenter server. Figure 4 shows our proposed framework. The left side is a datacenter server, and the right side is the remote verifier.

A. Code Signature for User Application

The remote verifier attaches code signatures to all server applications using a private key of the remote verifier. The code signature is used for authentication and verification of the integrity of applications on the server. The signed applications are pre-installed in the HDD of the server. Only server applications including code signatures are allowed to be invoked on the server.

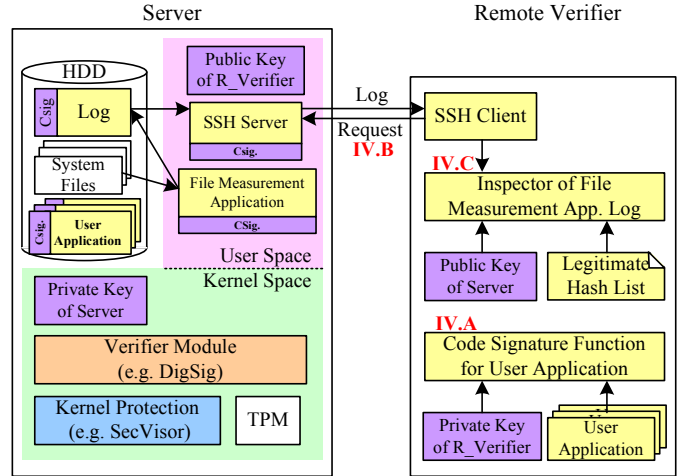


Fig. 4 Framework of Remote Attestation for Datacenter Server.

B. Request for Remote Attestation

The remote verifier sends requests via a SSH channel to the server to measure important files, i.e., configuration files and password files. The SSH service on the server is a trusted application that is authenticated using the code signature. The SSH service invokes a file measurement application that is also authenticated and verified using the code signature. The file measurement application measures the important files on the HDD and outputs a hash list as the measurement result. The latter is signed by a private key of the server in the trusted kernel.

As the purpose of the measurement is to detect the manipulation of important files on the HDD, those that could be changed at runtime, i.e., log files and databases, are not targeted for the measurement.

C. Verification and Inspection of the Measurement Result

The measurement result is sent back to the remote verifier across the SSH channel. The remote verifier verifies the attached signature by using the public key belonging to the server.

The remote verifier manages a legitimate hash list that indicates the original hash values of the important server files. The replied hash list is compared with the legitimate hash list to detect any manipulations. When all hash values in the measurement result are equal to the legitimate hash list, the remote verifier guarantees the integrity of server files. When inconsistencies are detected, the remote verifier regards the server as compromised.

V. DESIGN AND IMPLEMENTATION

In this section, we design and implement the remote attestation scheme for HDD files, which invokes a trusted file measurement application and signs a measurement result in the trusted kernel. Our scheme achieves a chain of trust from the BIOS via the boot loader, the kernel, the file measurement application, and the HDD files to the measurement result on the running server.

A. Assumptions and Trust Model

The security threats of this research assume a remote attacker who accesses to the server from the Internet. No local and/or physical attackers can physically access the datacenter room, while the operators of both the datacenter and the remote verifier are trusted.

The kernel protection module called SecVisor collaborating with the TPM can be a root of trust on the server.

The interface of the file measurement application is simply based on the file read function. As it is easy to implement without vulnerabilities, using only the file read interface, we assume that the file measurement application loaded on the user memory cannot be manipulated by attackers (e.g., buffer overflow) and measures the HDD files completely.

B. Chain of Trust from BIOS to Application

Given its requirement to invoke the trusted file measurement application on the non-manipulated kernel, we consider how to achieve a chain of trust from BIOS to the application on the running server.

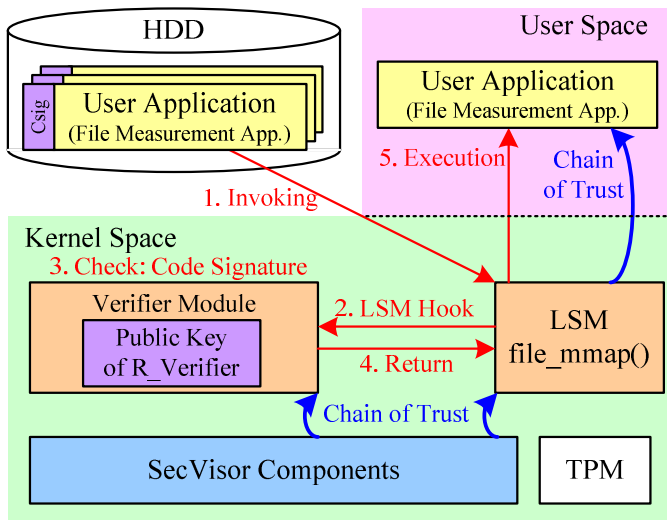


Fig. 5. Invoking Mechanism of Non-manipulated Application from Non-manipulated Kernel.

BIOS and boot loader are measured by the TPM. When the manipulations of the BIOS or boot loader are detected, the system bootup process is terminated. As the TPM is only used for the initial measurement, the TPM operations are less complicated and easier to implement.

SecVisor loader measures SecVisor runtime and starts it when the integrity of the latter runtime is verified. SecVisor runtime measures the Linux kernel. When the integrity of the Linux kernel is also verified, SecVisor runtime transfers control to the Linux kernel. Also, SecVisor verifies the dynamically loaded kernel modules and guarantees the code integrity of both kernel and modules at runtime. Next, we design the chain of trust from the kernel to a user application. The non-manipulated kernel verifies the code signature attached to the user application at the LSM check point shown in Figure 5. The LSM check point is implemented inside the kernel that is protected by SecVisor.

- 1) An invoking request of the user application is called.
- 2) The LSM check point of `file_mmap()` hooks the invoking request.
- 3) The verifier module verifies the code signature of the user application. If the hash value that is calculated from the code signature using the public key of the remote verifier matches what is calculated by the user application, the integrity of the user application is verified. Otherwise, the user application is considered to be manipulated.
- 4) The verifier module returns an “OK” or “stop” signal to the `file_mmap()` LSM.
- 5) Only in the case of “OK” signal, the user application is invoked to run in memory. In the case of the “stop” signal, the invoking process is canceled.

C. Chain of Trust from Application to HDD Files

One of the verified user applications is the file measurement application that measures HDD files on the running server. When a measurement result is output on the HDD, an attacker can manipulate it, hence the importance of signing the measurement result to detect manipulations. We design and implement a secure signing mechanism that attaches a signature to the measurement result in the trusted kernel before outputting it on the HDD. Thus, a chain of trust from the user application to the HDD files is achieved as shown in Figure 6.

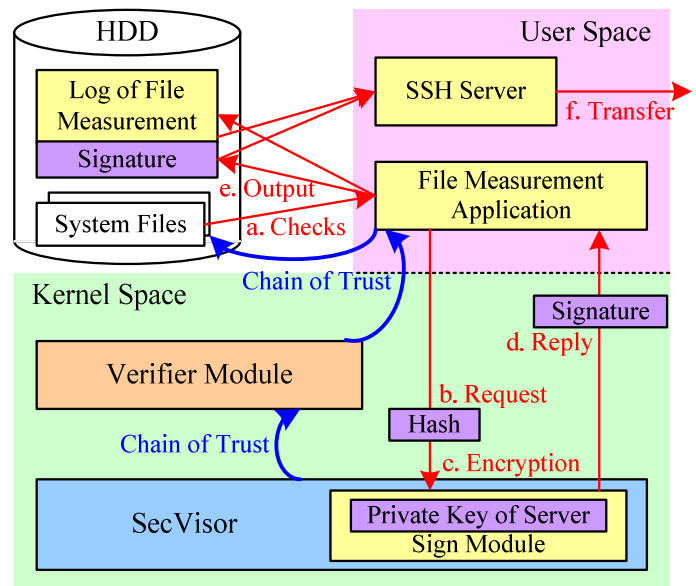


Fig. 6. Secure Sign Mechanism for Measurement Result.

- a) The file measurement application reads the important files and derives their hash values as a measurement result. The measurement result is output as a hash list on the application memory. Also, a hash value is derived from the measurement result.
- b) The hash value derived from the measurement result is forwarded to SecVisor via `sysfs` file systems [14], which are considered a communication channel between the user and kernel spaces.

- c) A sign module on the SecVisor encrypts the hash value using the private key of the server. The encrypted hash value is considered to be a signature of the measurement result.
- d) The signature of the measurement result is returned to the file measurement application from SecVisor via *sysfs* file systems.
- e) The file measurement application outputs both the measurement result and its signature to the HDD.
- f) Both the measurement result and the signature will be sent to the remote verifier via the SSH channel and verified using the server public key.

Upon system bootup, public and private keys are generated in a sign module that was implemented in SecVisor. Only the public key is output on the HDD files and submitted to the remote verifier. As the private key is managed only inside SecVisor's memory, it can be protected from both user and kernel space. This means that no processes, i.e., with either user or root privileges, can have read/write access to the private key. With this scenario, since the key pair differs at every boot cycle, the remote verifier is aware of any system reboot. As the server in the datacenter is rarely rebooted, the procedure of public key submission is acceptable for the server administrator.

D. Maintenance of the Trust Chain

In the case of the datacenter server, the kernel, libraries, and user applications are frequently updated. We consider the maintenance of the trust chain from the kernel to the user application.

When the kernel modules are modified, the original hash list managed by the SecVisor is updated via a trusted management terminal. When the user application is modified, the code signature attached to the user application is replaced by the remote verifier. As these procedures are uncomplicated, the overall maintenance scheme can be well retained.

VI. EVALUATION

In this section, we evaluate the CPU overhead of computation-intensive applications and the throughput of the *httpd* process on our proposed model.

A. Evaluation System

We implemented our scheme on a server with the following conditions:

- HP Compaq dc5750 Microtower
- 2.2 GHz AMD Athlon64 (dualcore CPU)
SVM hardware virtualization support
- 2GByte RAM
- Linux 2.6.20.14

To present the detailed performance overhead, we evaluate the four systems shown in Table I: (i) regular kernel, (ii) application verifier module on regular kernel, (iii) trusted kernel, and (iv) verifier module on trusted kernel.

TABLE I EVALUATION PATTERN.

ID	Conditions
(i)	Linux 2.6.20.14 "= Regular Kernel"
(ii)	Linux 2.6.20.14 + Application Verifier Module
(iii)	Linux 2.6.20.14 + Trusted Kernel
(iv)	Our Proposed Model: Linux 2.6.20.14 + Application Verifier Module + Trusted Kernel

B. Performance Evaluation of Mathematical Application

To evaluate the CPU overhead of the computation intensive application, a sample program for a CPU time measurement has been implemented, which calculates the circular constant " $\pi=3.14159\dots$ ". The " π " application was run 5 times and the results in the figure show the average CPU time. The CPU time includes invoking, calculating, and terminating processes.

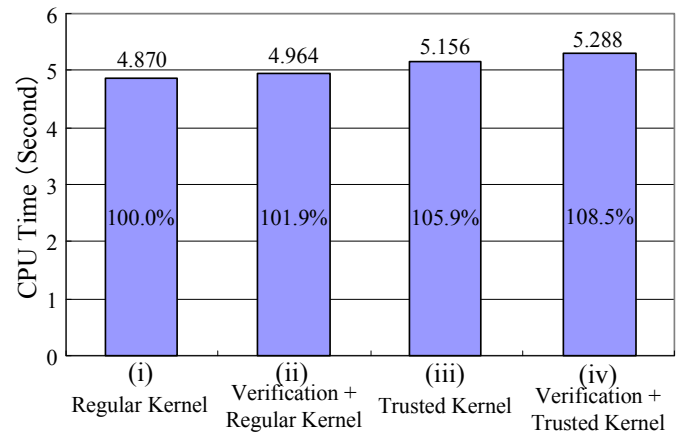


Fig. 7 Evaluation of the CPU Time.

The evaluation results are shown in Figure 7. The verification runtime of the benchmark is calculated as (ii) - (i) = 0.094 seconds. The overhead of the verification is 1.9%. For (iii), the runtime of the benchmark on the trusted kernel is 5.156 seconds. The overhead is 5.9%, this is because SecVisor switches memory access permission frequently, the runtime increases. And finally, for (iv), the runtime of the benchmark is 5.288 seconds. The overhead is 8.5%, because of the increased memory permission switches. The performance penalty on our proposed model is relatively small, which means the remote attestation using the kernel protection mechanism can apply to datacenter servers.

C. Performance Evaluation of the *httpd* Process

The kernel protection and integrity measurement mechanisms are effective against remote attacks on the datacenter. Next, we evaluate the communication throughput of the *httpd* process of the web server. As the *httpd* process receives and replies packets, DMA and socket operations are often controlled by the kernel protection mechanisms. When a traffic generator sends many request packets "*HTTP/1.0 GET index.html*" to the evaluation system, the successful requests/second is measured in the form of communication throughput. Figure 8 shows the throughput performance on (i) - (iv) kernel types.

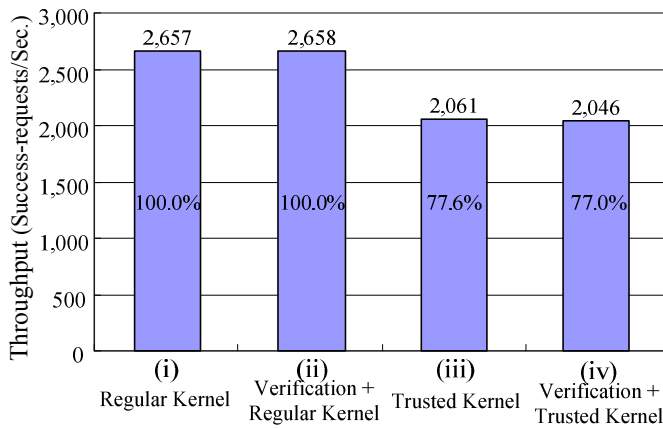


Fig. 8. Evaluation of the HTTP-GET Throughput.

The throughput of (i) and (ii) are at the same level. Because the *httpd* daemon have been invoked, no verification overhead for the *httpd* daemons is added. For (iii) and (iv), the throughput of *httpd* decreases about 23%. The network interface card (NIC) receives the request packets and transfers it to the kernel memory via the DMA controller where SecVisor is involved to prevent the buffer overflow attack. Also, the web page “index.html” is replied to the NIC via a socket operation that is a scenario where the SecVisor is involved. Though the *httpd* operation includes a considerable overhead in our proposed model, the throughput degradation remains however at a relatively minor level.

VII. CONCLUSION

This paper proposes a system for secure and flexible measurement of HDD files on datacenter servers. We propose a remote attestation scheme that implements a non-manipulated file measurement application on a trusted kernel. The file measurement application measures HDD files on datacenter servers at runtime. Our scheme achieves a chain of trust from the BIOS via the boot loader, the trusted kernel, and the trusted file measurement application to HDD files. Also, we design and implement a signature mechanism that signs the measurement result on the trusted kernel before outputting it to the HDD. Because our proposed model can attest files on the HDD, not only at boot time but also during runtime, it can be used for intrusion and manipulation detection on datacenter servers.

In Section VI, only the CPU time and http throughput without additional CPU load are measured. It is not sufficient to evaluate the runtime attestation on the datacenter server. We reserve the evaluations of the CPU time and communication throughput with some server loads for future study.

REFERENCES

- [1] Min Xu, Xian Jiang, Ravi Sandhu, and Xinwen Zhang, "Towards a VMM-based Usage Control Framework for OS Kernel Integrity Protection," Proceedings of the 12th ACM symposium of SACMAT, pp. 71-80, June, 2007.
- [2] Ryan Riley, Xuxian Jiang, and Dongyan Xu, "Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing," Proceeding of the RAID'08, pp. 1-20, September, 2008.
- [3] Peter A. Loscocco, Perry W. Wilson, J. Aaron Pendergrass, and C. Durward McDonell, "Linux kernel integrity measurement using contextual inspection," Proceedings of the ACM workshop on STC, pp. 21-29, November, 2007.
- [4] Trust Computing Group:
http://www.trustedcomputinggroup.org/groups/TCG_1_4_Architecture_Overview.pdf
- [5] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," Proceedings of the 13th Conference on USENIX Security Symposium, Aug. 2004.
- [6] Integrity Measurement Architecture (IMA):
http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html
- [7] Dries Schellekens, Brecht Wyseur, and Bart Preneel, "Remote Attestation on Legacy Operating Systems with Trusted Platform Modules," Proceedings of REM 2007, Sep., 2007.
- [8] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn and Pradeep Khosla, "Pioneer: Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems," ACM, Proceeding of the SOSP'05, October, 2005.
- [9] AMD Virtualization:
http://www.amd.com/us-en/0,,3715_15781,00.html?redir=SWOP08
- [10] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig, "SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes," ACM, SOSP Symposium, October, 2007.
- [11] A. Apville, M. Pourzandi, D. Gordon, and V. Roy, "Stop Malicious Code Execution at Kernel-Level," Linux World, Vol. 2, No. 1, January, 2004.
- [12] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, and Greg Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," Proceedings of the 11th USENIX Security Symposium, pp. 17-31, August, 2002.
- [13] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri, "Minimal TCB Code Execution," IEEE International Symposium on Computer and Communication, Vol. SP, pp. 267-272, May 2007.
- [14] The sysfs Filesystem:
<http://kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf>