International Journal of Computer Applications Technology and Research
Volume 2– Issue 2, 94 - 98, 2013, ISSN: 2319–8656

A Trusted Integrity verification Architecture for Commodity Computers

Angela Francis
Karunya University
Coimbatore,
India

Renu Mary Daniel
Karunya University
Coimbatore,
India

Vinodh Ewards S.E.
Karunya University
Coimbatore,
India

Abstract: Trust is an indispensable part of the computing environment, the validity of any transaction or information depends heavily on the authenticity of the information source. In this context, many mechanisms for ensuring the authenticity of the information source were developed, including password verification and biometrics. But as the attacks are directed towards the computing platform and the applications running on the computer, all these initial security mechanisms are not sufficient. It is essential to ensure before making a secure transaction that the system is in a good state (or say some authorized state) and maintains its integrity throughout the execution time. The emergence of the Trusted Platform Module (TPM) has added to the security feature of a computer. Mechanisms are in place which guarantee system integrity but very little is known about the state of the applications running on them. We propose a system which notifies the user if the integrity of an application is violated and stops it. Our system also compares the current system state with a known good value to ensure platform integrity.

Keywords: Trust; Trusted Platform Module (TPM); Integrity Measurement; Sealing; Application Security

1. INTRODUCTION

Ensuring trust in cyber space has been a prime concern since the epidemic growth of online transactions and communications. Commodity computers are increasingly used to access banking transactions, sending sensitive e-mails, accessing personal and confidential information from remote systems, where it becomes the prime necessity to assure the user that security sensitive operations executes always on secure and trusted state of system. Authenticity of the information source and non-repudiation can be achieved through many mechanisms like passwords, biometrics, digital signatures and cryptographic protocols. These mechanisms ensure that the user is genuine and authorized to view the information. They also guarantee that the integrity of the information during transmission is maintained. But can we know with absolute certainty that the system with which we are communicating is not malicious? In order to establish trust in computer and verify its existence, it is required to know something more other than the authentication. And what is that more requires understanding of the following: what is meant by trusted system? What are the components involved in it? How to boot the system in trusted state? Does booting the system in trusted state guarantee that system will remain in trusted state while execution? As attacks are directed towards the BIOS, boot loader and kernel, maintaining the system integrity is extremely difficult. To ensure that the system is in a trusted state, the Trusted Computing Base (TCB) of the system should be verifiable. But owing to the enormous code comprising the TCB, is it possible to vouch for the integrity of the system during each transaction?

Trusted Computing (Trusted Computing Group, 2007) aims at establishing trust in commodity computers and the transactions performed by them. TCG's Trusted Platform Module (TPM) (Bajikar, 2002) is a cryptoprocessor chip, that computes the current platform state during boot time. But, TPM is a passive device, it does not notify the user if there is any change in the system state. The values stored by it can be used for later verification with a known good state. Many mechanisms like Tboot (Trusted Boot, 2012) and OSLO (Kauer, 2007) were developed to provide trusted boot, where the platform state will be compared to a set of known good measurements. These mechanisms require a system with Intel TXT or AMD SKINIT instruction and virtualization technology support. Our system makes use of the TPM chip's boot time integrity measurement to check if the system is in a trusted state without any additional requirements.

As TPM does not compute the hash of the applications or services in the system it cannot stop a service or application if it is compromised. So, if the applications and services are running alongside untrusted applications, can we guarantee the genuineness of these applications? They can be targeted and compromised. Thus there is a need to provide isolation to the execution of security sensitive code, so that attacks directed towards it during execution can be thwarted. Flicker (McCune et. al., 2008) is one such project which aimed at providing isolated execution of a security sensitive code by switching from untrusted environment to the minimal trusted environment. It ensures run time integrity of security sensitive code. We propose a system in which the application integrity can be verified before launch and stopped if found to be malicious, thus providing a way to extend trust to the application and service level.

2. BACKGROUND

2.1 Trusted System and Trusted Computing Base

There are various definitions which have been proposed to define "trusted system". Schneider (Shirey, 2007) defines trusted system as, "a system that operates as expected, according to design and policy, doing what is required – despite environmental disruption, human user and operator errors, and attacks by hostile parties – and not doing other things."

According to Neumann's definitions (Neumann, 1995), "an object is trusted if and only if it operates as expected."

An important factor in establishing trust in computer system or any computing device is identifying the *trusted computing base* (TCB). It is a totality of protection mechanisms within a computer system, including hardware, firmware, and software, the combination of which is responsible for enforcing a security policy (U.S. Department of Defense, 1990) and critical to its security. Any vulnerability or weakness inside the TCB components may potentially affect the security of whole system and hence system may get compromised, whereas the vulnerabilities or weakness (software or hardware) outside the TCB must not affect the security of system beyond the confined area.

Rushby, (1981) defines the trusted computing base as the combination of *kernel* and *trusted* processes. The trusted processes are special process that are allowed to violate the system's access-control rules.

Whereas Lampson et al. (1992) define the TCB of a computer system as simply *"a small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security"*.

The Orange Book (Department of Defense, 1985) further explains that [t]he ability of a trusted computing base to enforce correctly a unified security policy depends on the correctness of the mechanisms within the trusted computing base, the protection of those mechanisms to ensure their correctness, and the correct input of parameters related to the security policy.

2.2 Boot Time Integrity

The best time to measure the identity of software code is before it starts execution. The identity of these components can be computed by taking the cryptographic hash of its binary as well as any inputs, libraries or configuration files used, also known as measurements. This requires the identity of all software components participate in the current state of computer namely BIOS, boot loader, and operating system (Gu et. al., 2009) (Parno et. al., 2011). The measurements taken at the clean state of system is termed as golden measurements (or golden images).

The software currently in control of the platform is measured by the software which had control of the platform previously. And the currently running software will measure the next software before it start execution. The process of measurement and execution continues till the system reaches to intended state and a chain of trust (Parno et. al., 2011) is thus established. The raises the fundamental question that, who initiated the chain of trust? It must be an immutable piece of code that initiates the chain of trust and forms the foundational root of trust (Parno et. al., 2011). TPM provides a programme code that serves as the Core Root of Trust for Measurement (CRTM), to initiate the measurement chain.

Once these identities (golden measurements) are measured, it can be used to boot the system in some authorized state known as secure boot and trusted boot. Secure boot assumes that measured software is trustworthy and only ensures a secure initial state i.e. at time t0. An immutable piece of code initiates the chain of trust by measuring the initial BIOS, verify against the golden measurement and execute if found correct else halt. Similarly, the boot chain continues till the kernel.

Whereas in trusted boot (techniques first used by Gasser et. al., 1989) chain of trust initiated by secure hardware (co-processor), it measure the next software, accumulate (or append) the measurement in memory and execute the software. It communicates the current state of system to user via attestation and can prove that system is booted in a known configuration, which enables the user to verify the state and establish trust that no malicious software is running.

2.3 Threat Model

The most vulnerable entry point for attacks are software applications as opposed to operating systems and the platform. Application layer hosts a major part of all vulnerabilities that facilitate cyber crime. As these applications are pervasive, they can be exploited to steal sensitive information. For instance, an ordinary user or an adversary may come across a bug in the application and gain access to privileged information. The attacks are mostly directed towards the information and resources being used by the applications, its users and developers. Since processes share information through shared memory regions, these attacks might be used to compromise the operating system through buffer overflows and invalidated input exploits. Changes made to the kernel may not be easily detected and can cause major damage. Thus before the launch of any security sensitive application platform integrity must be verified and the trust chain must be extended to include the applications and services.

3. RELATED WORK

Web browsers and operating systems do not provide any mechanism by which a user can be sure that the sensitive information is reaching the intended destination unaltered. Software-only protection schemes cannot ascertain the integrity of software since it can be corrupted in many ways like improper installation, upgradation and malware attacks.

Flicker is a secure infrastructure that allows the security-sensitive code to run in complete isolation by utilizing the concept of late launch provided by Intel and AMD processors and Dynamic Root of Trust for Measurement (DRTM) provided by TPM v1.2 chips. Flicker (McCune et. al., 2008) allows application developers to focus on the security of their code without blindly trusting an unverifiable quantity of code executing below. Flicker guarantees that the security sensitive code will execute in isolation without requiring a reboot, a change of OS, or a VMM. It can operate at any time and does not require a new OS or even a VMM.

Adding only a few hundred lines to the TCB, Flicker protects fine granules of security-sensitive code. Due to the frequent use of hardware support for a dynamic root of trust for measurement, Flicker incurs significant performance overhead. In situations with demanding performance, several characteristics of Flicker renders it impractical for use. When Flicker session executes, the user thinks that the system has momentarily hanged. TrustVisor (McCune et. al., 2010) aims to achieve high performance for legacy applications and also to protect small security-sensitive code blocks within a potential malicious environment. A special purpose hypervisor called TrustVisor is developed that invokes the security-sensitive code module without trusting the OS or the applications for isolated execution.

4. PROBLEM STATEMENT

Based on the survey of trusted systems and TPM it is found that they mainly guarantee system integrity and very little is known about the state of the applications running on them. As stated earlier most of the work done for protecting the applications focuses on providing isolation for their execution but do not alert the user if any modification to the application is made. If these modifications or alteration are not known at an early stage and corrected then they may serve as vulnerabilities which can be easily attacked. Further, it should be possible to stop the malicious service or application. TPM being a passive device provides measurement and protected storage, but will not interfere with the execution of applications in the system. It only measures and does not provide a mechanism to verify the system integrity.

5. SYSTEM DESIGN

5.1 Work-flow of the System

When the system boots, TPM measures the integrity of the BIOS, bootloader, operating system, etc. which is stored in the Platform Configuration Registers (PCR) from 0-7 (Bajikar, 2002). These PCRs provide a secure storage and can be used for verifying the integrity of the system with the help of the sealing and unsealing mechanism provided by TPM.

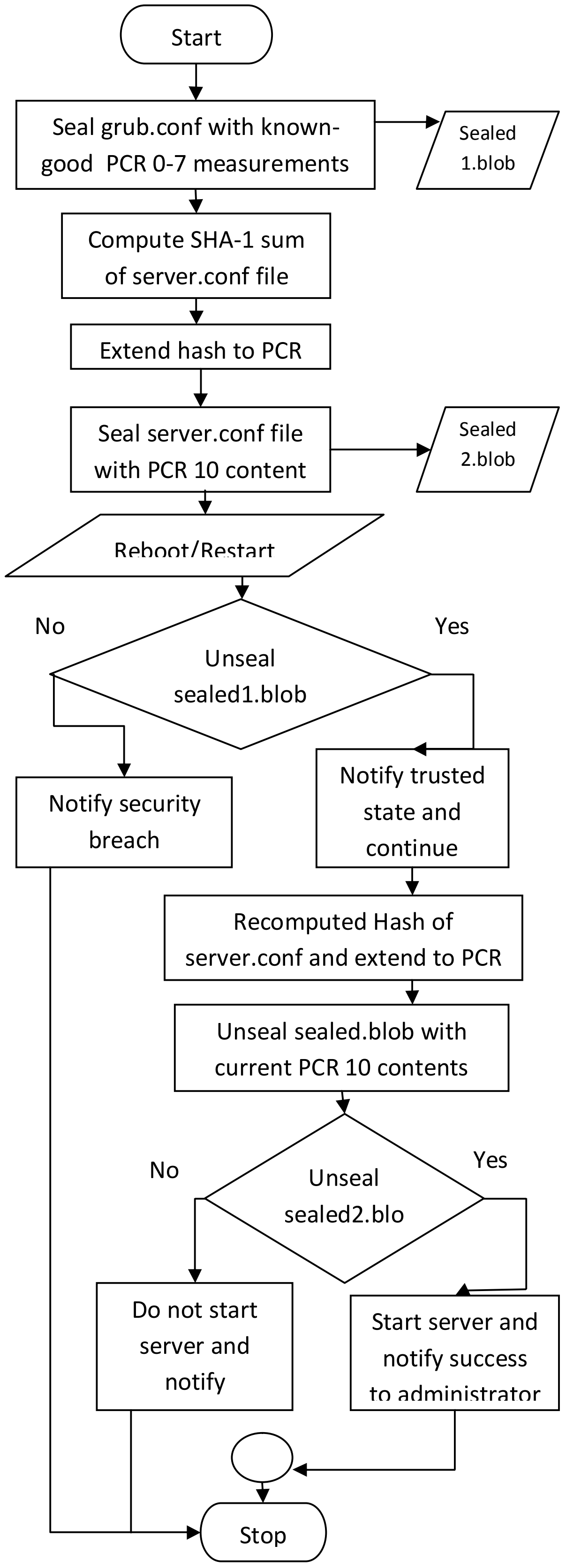
Start
Seal grub.conf with known-good PCR 0-7 measurements
Sealed 1.blob
Compute SHA-1 sum of server.conf file
Extend hash to PCR
Seal server.conf file with PCR 10 content
Sealed 2.blob
Reboot/Restart
No
Yes
Unseal sealed1.blob
Notify security breach
Notify trusted state and continue
Recomputed Hash of server.conf and extend to PCR
Unseal sealed.blob with current PCR 10 contents
No
Yes
Unseal sealed2.blo
Do not start server and notify
Start server and notify success to administrator
Stop

Figure. 1 Work-flow of the System

The same mechanism can be used to verify an application's integrity. We propose a design which notifies a user if any changes are made to the system at every boot and also checks for the integrity of a service or application before it starts.

When the system is in a good state PCR 0-7 have known good measurements. To check for system integrity a system configuration file is sealed with the known good measurements i.e. PCR 0-7. Sealing is a security mechanism provided by the Trusted Platform Module. It allows the data being sealed to be tied to a particular platform state as represented by one or more PCR contents. The Storage Root

Key will be used to encrypt the sealed data and for each sealing and unsealing, SRK password will be prompted by the system. This provides additional security as the private part of the SRK never leaves the TPM chip and is stored in the TPM NV-RAM. Unsealing is possible only if the platform state during unsealing matches the platform state during sealing.

This sealed file is then stored in a secure location. Sealed file in the secure location is attempted to be unsealed using the current PCR 0-7 contents at each system boot. If the unseal operation is successful the output file is written and the administrator is notified that the integrity of the system is maintained. Otherwise, unseal operation fails, output file cannot be written and administrator is notified about the security breach.

Application integrity checking begins by hashing the configuration file of the application or service using the SHA-1 algorithm. The result is then extended to PCR 10 i.e. PCR 10 is updated with the output of the hash and its current value. The following expression denotes the extend operation:

$$\text{PCR} \leftarrow \text{Hash} (\text{PCR} \parallel \text{Hash(config file)})$$

The configuration file is then sealed with the PCR 10 contents, i.e. the clean state measurement of the file.

After sealing the sealed blob will be generated and stored in a secure storage. During system start up, PCRs 0-16 comprising of the static PCRs will be reset to zero. The hash of the configuration file is again computed and extended into PCR 10 and using the current PCR 10 value, unseal operation is attempted. If the configuration file has not been altered, its measurement remains the same. Then, value of PCR 10 during sealing and unsealing remains the same and the sealed file can be successfully unsealed and the service or application is launched. If any modification is made to the configuration file the unseal operation fails, the service is not started and the administrator is notified.

5.2 Experimental Setup

A version 1.2 TPM is required and it must be enabled and activated in the BIOS. The system used for this implementation is HP-Compaq 8100 with Intel Core i5-650 vPro processor. The system is embedded with a TPM. TPM tools and TrouSerS were installed to communicate with the TPM. Our implementation is written in shell script and is assumed to be a part of the Trusted Computing Base (TCB) cause it measures and verifies the application's configuration files before execution. The grub.conf file is sealed with the contents of PCR 0-7 using the following command:

```
tpm_sealdata -i grub.conf -o sealed1.blob -p 0 -p 1 -p 2 -p 3 -p
4 -p 5 -p 6 -p 7
```

The output of the command which is the sealed1.blob file is stored in a secure location viz. a flash drive. Each time a system is booted and before any application or services start the sealed file in the flash drive i.e. sealed1.blob is unsealed using the tpm_unsealdata command.

```
tpm_unsealdata -i sealed1.blob -o unseal1.blob
```

If PCR 0-7 state is not same as it was while sealing then unseal operation fails. The administrator or user is notified about the state of the system.

For measuring and extending the application configuration to a PCR we use TrouSerS Programming (Challener, 2011). The command line arguments provided to the PCR extend program are shown in the following expression:

```
./pcr_extend.exe -p 10 -v `sha1sum app.conf`
```

where, third argument tells which PCR will be extended and the fifth argument is the hash of the configuration file which will be extended. For experimental purpose we use Apache Web Server to verify its integrity before it starts. After extending, the apache.conf file is then sealed with the contents of PCR 10 using the tpm_sealdata command. The command is as follows:

```
tpm_sealdata -i apache.conf -o sealed2.blob -p 10
```

The output of the command which is the sealed file is stored in a secure location viz. a flash drive. Each time a system is booted and before Apache starts the hash of the configuration file is taken and extended to PCR 10, then the sealed file in the flash drive i.e. sealed.blob is unsealed using the tpm_unsealdata command.

```
tpm_unsealdata -i sealed.blob -o unseal.blob
```

If PCR 10 state is not same as it was while sealing then unseal operation fails. The Web server is then stopped and the alteration is notified to the user.

6. RESULTS AND DISCUSSION

One of our design goals for the system was to notify the user if any change is made to the platform and configuration file of the application. The changes made, if notified at an early stage can be corrected and the system will be protected from prospective danger or invasion from attacks. We achieved this by executing a startup script assumed to be a part of the TCB using sealing and unsealing to check for integrity. This increases the size of the TCB by few lines. Currently some features are still unimplemented such as non-bypassability i.e. the startup script should not be changed by any user.

7. FUTURE WORK AND CONCLUSION

The system was designed to ensure the launch-time integrity of an application or service using the security features provided by the TPM. As the script is assumed to be part of the TCB, we have implemented a simple mechanism for verifying boot-time integrity of the system. Also, we are working towards ensuring the non-bypassability aspect of the system. The script is security critical and can be invoked as the Piece of Application Logic in Flicker (McCune et. al., 2008), to provide isolation during execution.

We have worked towards extending the trust aspect provided by the TPM to the application and services in the system. We have explored the extent to which the chain of trust is currently being made and have designed a system to ensure the integrity of applications before being started.

8. ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the faculty in Karunya University towards this research.

9. REFERENCES

[1] Trusted Computing Group, Incorporated, 2007. TCG specification architecture overview.

[2] Bajikar, S. 2002. Trusted Platform Module (TPM) based Security on Notebook PCs White Paper. Intel Corporation.

[3] Trusted Boot, sourceforge.net, Sept. 12, 2007. [Online]. Available: http://sourceforge.net/projects/tboot [Accessed: Aug. 10, 2012].

[4] Kauer, B. 2007. OSLO: Improving the Security of Trusted Computing. In Proceedings of 16th USENIX Security Symposium.

[5] McCune, J. M., Parno, B. J., Perrig, A., Reiter, M. K., and Isozaki, H. 2008. Flicker: An Execution Infrastructure for TCB Minimization. In Proceedings of 3rd ACM EuroSys European Conference on Computer Systems, pp. 315-328.

[6] Shirey, R. 2007. RFC 4949 – Internet Security Glossary, Version 2 (IETF).

[7] Neumann, P. G. 1995. Architectures and formal representations for secure systems, SRI Project 6401, Deliverable A002 (Computer Science Laboratory, SRI International).

[8] U.S. Department of Defense. 1990. Glossary of Computer Security Terms (Aqua Book) (National Computer Security Center, Fort Meade).

[9] Rushby, J. 1981. Design and Verification of Secure Systems. In 8th ACM Symposium on Operating System Principles. Pacific Grove, California, US. pp.12–21.

[10] Lampson, B., Abadi, M., Burrows, M., and Wobber E. 1992. Authentication in Distributed Systems: Theory and Practice. ACM Transactions on Computer Systems, on page 6.

[11] Department of Defense trusted computer system evaluation criteria. 1985. DoD 5200.28-STD, In the glossary under entry Trusted Computing Base (TCB).

[12] Gu, J., and Ji, W. 2009. A secure bootstrap based on trusted computing. In International Conference on New Trends in Information and Service Science, IEEE.

[13] Parno, B., McCune, J. M., and Perrig, A. 2011. Bootstrapping Trust in Modern Computers, ISBN 978-1-4614-1459-9, Springer.

[14] Gasser, M., Goldstein, A., Kaufman, C., and Lampson B. 1989. The digital distributed system security architecture. In Proceedings of the National Computer Security Conference.

[15] McCune, J. M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., and Perrig, A. 2010. TrustVisor: Efficient TCB Reduction and Attestation. In IEEE Symposium on Security and Privacy, pp. 143-158.

[16] Challener, D. 2011. Programming with TrouSerS. John Hopkins University.