# Launch Time Application Security: The Trusted Approach

**Renu Mary Daniel[1], Angela Francis[2], Vinodh Ewards S. E.[3]**

[1]Department of Computer Science and Engineering, Karunya University,
Coimbatore, Tamil Nadu, 641114, India
*renumary@karunya.edu.in*

[2]Department of Computer Science and Engineering, Karunya University,
Coimbatore, Tamil Nadu, 641114, India
*angelafrancis@karunya.edu.in*

[3]Assistant Professor, Department of Computer Science and Engineering, Karunya University,
Coimbatore, Tamil Nadu, 641114, India
*ewards@karunya.edu*

**Abstract:** ***In the day to day lives of human beings 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. The emergence of the Trusted Platform Module (TPM) has added to the security feature of a computer. Much work has been done to ensure the integrity of the system using a TPM. But what happens when the applications running on top of the Trusted System are malicious? The sensitive data provided to such applications becomes vulnerable for exploit. We propose a system which notifies the user if the integrity of an application is violated and stops it. The system makes use of the functionalities of the TPM chip.***

**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 [1] aims at establishing trust in commodity computers and the transactions performed by them. TCG's Trusted Platform Module (TPM) [2] 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 [3] and OSLO [4] were developed to provide trusted boot, where the platform state will be compared to a set of known good measurements.

But, TPM does not compute the hash of the applications or services in the system. TPM 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 [5] 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. Literature Survey

### 2.1 Trusted System and Trusted Computing Base

There are various definitions which have been proposed to define "trusted system". Schneider [6] 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 [7], "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 [8] 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,[9] defines the trusted computing base as the combination of *kernel* and *trusted* processes. The trusted processes are special processes that are allowed to violate the system's access-control rules.

Whereas Lampson et al.[10] 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 [11] 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 [12][13]. 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 [13] 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 [13]. 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. [14]) 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 is 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. Existing security measures such as firewalls, intrusion detection techniques as well as anti-viruses fail to provide a complete sense of security.

## 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 [5] 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 render it impractical for use. When Flicker session executes, the user thinks that the system has momentarily hanged. TrustVisor [15] 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.

## 5. System Design

### 5.1 Work-flow of the System

We assume that the system is trusted i.e. boot time integrity is maintained and we propose a design which notifies a user if any changes are made to a service or application before it starts. TPM measures the integrity of the BIOS, bootloader, operating system, etc. which is stored in the Platform Configuration Registers (PCR) from 0-7 [2]. These PCRs provide a secure storage and can be used for verifying the integrity of an application with the help of the sealing and unsealing mechanism provided by TPM.

We start 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:

$$PCR \leftarrow Hash\ (PCR \parallel Hash(config\ file))$$

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

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.

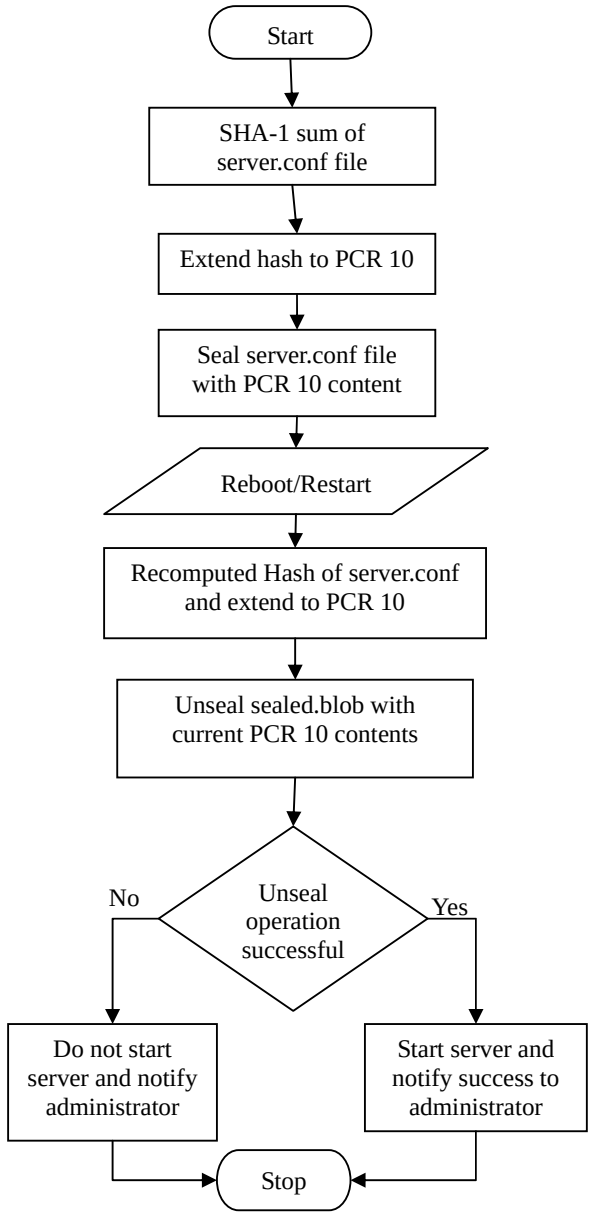


**Figure 1:** Work-flow of the integrity checking process

### 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. As commands for extending a value to a PCR are not available we use TrouSerS Programming [16]. 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 sealed.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 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 recommend TBOOT [3] or OSLO [4] (depending on the processor), for maintaining 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 [5], 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.

## References


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[16] David Challener, “Programming with TrouSerS,” John Hopkins University, 2011

## Author Profile

**Renu Mary Daniel** graduated in Computer Science and Engineering from Anna University, Coimbatore, India. She is currenly pursuing Master’s Degree in Computer Science and Engineering from Karunya University, Coimbatore. Her research interests focus on the areas of trusted computing, cryptography and cyber security.

**Angela Francis** received the Bachelor’s Degree in Computer Science and Engineering from Karunya University. She is working towards the Master’s Degree in Computer Science and Engineering at Karunya University, Coimbatore. Her research interests include trusted computing, privacy enhancing technologies and other topics in computer security.

**Vinodh Ewards S.E.** is working as Assistant Professor in the Department of Computer Science and Engineering, Karunya University, Coimbatore, India. His field of interest and research includes cloud computing and network security.