Trusted Execution Environment and Linux- A Survey

Prabhav S[1] , Madhav V Deshpande[2] , Rakshak R Kamath[3] , Rohan N[4] , Latha NR[5]

1,2,3,4 (IV year B.E., Department of CSE, BMS College of Engineering, Bangalore.)
5(Assistant Professor, Department of CSE, BMS College of Engineering, Bangalore.)

Abstract- *Improving security and user integrity in Linux operating system has always been a challenging task. One method of improving security is by implementing the Trusted Execution Environment (TEE). While Trusted Computing and Linux may seem antithetical on the surface, Linux users can benefit from the security features, including system integrity and key confidentiality, provided by Trusted Computing. Thisis a study on the key concepts and software components that are required to enable the use of the TEE. Also the existing implementation of TEE by IBM, Intel and TCG has been discussed here.*

Keywords- *Linux, Trusted Execution, Kernel, Security.*

.

1. INTRODUCTION

Linux has been the most widely used operating system. From personal computers to embedded systems and servers all use some form of Linux. Although its popularity has exploded since its inception, it has had its share of problems. One such problem is unauthorised access control hence gain of system access. Trusted Execution is the method discussed to solve this problem.

This paper is organized into sections covering the goals of Trusted Execution, a brief introduction to Trusted Execution, the components required to make an operating system a trusted operating system from the IBM, Intel perspective, the necessity for Trusted Computing, and finally concludes with verification of the integrity.

2. BACKGROUND

Attacks on IT infrastructure continue to grow in volume, complexity, sophistication, and stealth. According to the McAfee Threat Report of 2Q 2012, the number of new unique malicious code and other unwanted programs grew by over 8 million samples between the first and second quarter of 2012 alone. These include a growing number of nasty, sophisticated rootkits, with more than a 100,000 new rootkit variants discovered in each of the last 14 quarters. Kaspersky Lab forecast a 10fold increase in malicious programs, from 2.2 million to 20 million in 2008. The specialists at Kaspersky Lab detected the 25 millionth malicious program and added it to the company's anti-virus databases in June 2009. Clearly, the malware tide is rising unabated [6].

Making matters worse, the cost of a data breach is daunting. The average organizational costs of a data breach in the United States remained a stubbornly high USD 5.5 million in 2011-12 with lost business as the largest percentage of this cost. According to the same study, the cost on a per-record basis is a challenging USD 194 per record—so a large-scale breach can be very damaging. In 2012 Global Payments, a U.S.-based payment processing firm, cited costs of USD 84.8 million related to a recent data breach. According to a 2012 study by FTI Consulting and Corporate Boardroom both Corporate Directors and General Counsel citedData Security as their top concern [6].

The Linux kernel was having the most CVE (Common Vulnerabilities and Exposures) vulnerabilities of all other products from 1988 to 2012,and the distribution in four mainstream LINUX version is presented, as shown in Figure 1 [7][11].

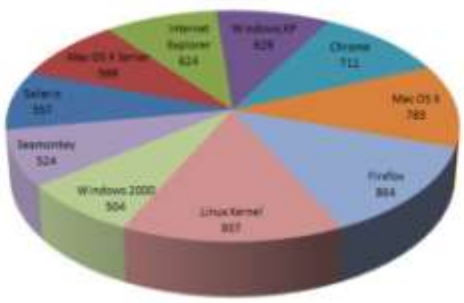


Figure 1. Products with most reported vulnerabilities[11]

Linux provided users with discretionary access control (DAC) as a means of restricting access to objects based on the identity of subjects and/or groups to which they belong to.

The discretionary access control(DAC) mechanism of Linux gives users (in a certain group) the same rights, and all processes created by a user have exactly the same privileges. The acquired permissions can also be transferred to other subjects, so a flaw in one software can lead to all the users' data being compromised.DAC is vulnerable to bypass and tampering and enhancing the primitive access control usually requires the kernel to be adjusted to accommodate [11].

Privilege escalation, an act of exploiting a design flaw, bug or configuration oversight in software application or operating system to obtain elevated access to resources that are normally protected from user or an application. Thus an application with more privileges than intended by the system administrator or application developer can perform unauthorized actions.

Threats to computer systems can be classified in different categories. One major category is the. With this type of threat, modules on a system are replaced with rogue modules that leave the system open to future access into and out of the system. It can also leave systems vulnerable to destruction of key files and data. System integrity is the basis of all other security aspects of a system. If the system integrity is compromised, all decisions and operations of that system may be compromised as well, as key system files and behaviour cannot be trusted or predicted [3].

Another common threat to computer systems is the threat of unauthorised access. This can be accomplished through several mechanisms. Direct brute force attacks such as password cracking software, or more subtle attacks, such as access gained by exploiting vulnerabilities in networking software, can result in unauthorized access [3].

A further manifestation of the threat of unauthorized access is the threat of unauthorised system administrator. Buffer overflows are a common mechanism used to gain unauthorized access. This type of attack gives a program more input data than it expects. The extra input data is crafted as an executable instruction set of some kind. When the program runs, the data area overflows into the stack area, and when the program returns, the overflowed data is executed. This is usually performed on privileged commands or executable. It causes code to be executed on the stack in the privileged state. These are commonly exploited mechanisms used to gain access to protected files or directories.

Trusted Execution is described in the following section as the solution to overcome the various problems in Linux in correspondence to the trusted execution environment present in IBM's AIX.

3. TRUSTED EXECUTION

3.1 BASICS

Trusted Execution (TE) refers to a collection of features that are used to verify the integrity of the system and implement advance security policies, which together can be used to enhance the trust level of the complete system.The usual way for a malicious user to harm the system is to get access to the system and then install Trojans, rootkits or tamper some security critical files, resulting in the system becoming vulnerable and exploitable. The central idea behind the set of features under Trusted Execution is prevention of such activities or in worst case be able to identify if any such incident happens to the system.

Using the functionality provided by Trusted Execution, the system administrator can decide upon the actual set of executables that are allowed to execute orthe set of kernel extensions that are allowed to be loaded. It can also be used to audit the security state of the system and identify files that have changed, thereby increasing the trusted level of the system and making it more difficult for the malicious user to do harm to the system. Figure 2 shows the possible implementation of a trusted environment.

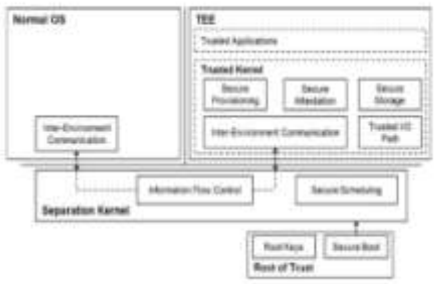


Figure 2. Trusted Computing Model [12]

The set of features under TE can be grouped into the following:

- Managing Trusted Signature Database
- Auditing integrity of the Trusted Signature Database

- Configuring Security Policies
- Setting Trusted Execution Path and Trusted Library Path.

Trusted Computing [4] is a technology that tries two answer two questions:

- Which software is running on a remote computer? (Remote Attestation)
- How to ensure that only a particular software stack can access a stored secret? (Sealed Memory)

3.2 KEY CONCEPTS

3.2.1 ROOTS OF TRUST

In the Trusted Computing Group's model, trusting the operating system is replaced by trusting the roots of trust. There are three roots of trust:

- Root of trust for measurement
- Root of trust for storage
- Root of trust for reporting

The root of trust for measurement is the code that represents the "bottom turtle"2. The root of trust for measurement is not itself measured; it is expected to be very simple and immutable. It is the foundation of the chain of trust.

The root of trust for storage is the area where the keys and platform measurements are stored. It is trusted to prevent tampering with this data.

The root of trust for reporting is the mechanism by which the measurements are reliably conveyed out of the root of trust for storage. This is the execution engine on the TPM [2].

3.2.2CHAIN OF TRUST

The chain of trust is a concept used by trusted computing that encompasses the idea that no code other than the root of trust for measurement may execute without first being measured. This is also known as transitive trust or inductive trust [2].

3.2.3VERIFICATION

Attestation is a mechanism by the Trusted Computing Group (TCG) for proving something about a system. The values of the Platform Configuration Register's(PCRs) are signed by an Attestation Identity Key and sent to the challenger along with the measurement log. To verify the results, the challenger must verify the signature, then verify the values of the PCRs by replaying the measurement log. A similar mechanism for verification of the digital signatures of Linux executables is identified later.

3.2.4SECURE BOOT

A common approach to implement secure boot is to use code signing combined with making the beginning of the boot sequence immutable by storing it within the TCB (e.g., on ROM of the device processor chip) during manufacturing; the processor must unconditionally start executing from this memory location. Boot code certificates that contain hashes of booted code, and are signed with respect to a device trust root, such as the device manufacturer public key that is immutable on the device, can be used to verify the integrity of the booted components. The TCB must be enhanced with cryptographic mechanisms that validate the signature of the system component launched first (e.g., the boot loader) that can in turn verify the next component launched (e.g., the OS kernel) and so on. If any of these validation steps fail, the boot process aborts. Integrity of the cryptographic mechanisms can be ensured by storing the needed algorithms on ROM [9].

4. COMPONENTS OF TRUSTED EXECUTION IBM

4.1 TRUSTED DATABASE

To hold the trusted files, a trusted database is created. A trusted file is a file that is critical from the security perspective of the system, and if compromised, can jeopardize the security of the entire system.

Typically the files that match this description are the following:

- All set uid programs
- All set gid programs
- Any program that is exclusively run by the root user or by a member of the system group
- Any program that must be run by the administrator while on the trusted communication path (for example, the ls command)
- The configuration files that control system operation
- Any program that is run with the privilege or access rights to alter the kernel or the system configuration files
- Every trusted file should ideally have an associated stanza or a file definition stored in the Trusted Signature Database (TSD).

- A file can be marked as trusted by adding its definition in the TSD using a command. IBM AIX implements this using a trustchk command [3].

4.2 INTEGRITY OF THE TRUSTED DATABASE

IBM uses the trustchk command can be used to audit the integrity state of the file definitions in the Trusted Signature Database (TSD) against the actual files.

If the trustchk command identifies an anomaly, then it can be made to automatically correct it or prompt the user before attempting correction.

If anomalies like size, signature, certificate or hash value mismatch, the correction is not possible. In such cases, the trustchk command would make the file inaccessible, thereby rendering it useless and containing any damage.

4.3 SECURITY MECHANISM

The Trusted Execution (TE) feature provides you with a run-time file integrity verification mechanism. Using this mechanism, the system can be configured to check the integrity of the trusted files before every request to access those file, effectively allowing only the trusted files that pass the integrity check to be accessed on the system.

When a file is marked as trusted (by adding its definition to Trusted Signature Database), the TE feature can be made to monitor its integrity on every access. TE can continuously monitor the system and is capable of detecting tampering of any trusted file (by a malicious user or application) present on the system at run-time (for example, at load time). If the file is found to be tampered, TE can take corrective actions based on pre-configured policies, such as disallow execution, access to the file, or logging error. If a file being opened or executed, and has an entry in the Trusted Signature Database (TSD), the TE performs as follows:

- Before loading the binary, the component responsible for loading the file (system loader) invokes the Trusted Execution subsystem, and calculates the hash value using the SHA-256 algorithm (configurable).
- This run-time calculated hash value is matched with the one stored in the TSD.
- If the values match, the file opening or execution is permitted.
- If the values do not match, either the binary is tampered, or somehow compromised. It is up to the user to decide the action to be taken. The TE mechanism provides options for users to configure their own policies for the actions to be taken if the hash values do not match.
- Based on these configured policies, a relevant action is taken.

IBM implements a certain set of policies to implement the discussed mechanisms [3] [5].

4.4 EXECUTION PATH

Trusted Execution Path (TEP) defines a list of directories that contain the trusted executables. Once TEP verification is enabled, the system loader allows only binaries in the specified paths to execute. Trusted Library Path (TLP) has the same functionality, except that it is used to define the directories that contain trusted libraries of the system [3].

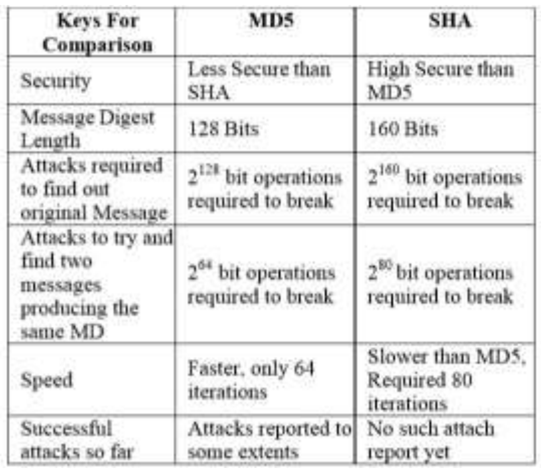
5. VERIFICATION OF SYSTEM INTEGRITY

System integrity involves capturing the good state of the system in a non-modifiable form and using it as a reference to check the state of the system periodically. The requirements of a good system-integrity tool can be listed as follows:

- Integrity Measurement: Provide administrator tools to detect changes to the system. The changes are identified by comparing the current state to a well-known previous state (also called as the baseline).
- Lockdown: Provide an administrator means to lock down the information established as baseline. This lockdown will prevent an intruder from modifying the state of the system and then recreate the baseline such that later the administrator won’t be able to detect the modifications.
- Monitor and Protect: Provide a means to monitor executions of executables, libraries, and kernel extensions [5].

5.1 INTEGRITY USING HASHING

The changes to a system can be measured by hash values. The SHA [10] algorithm proves superior to MD5[1]in various ways.



Keys For Comparison	MD5	SHA
Security	Less Secure than SHA	High Secure than MD5
Message Digest Length	128 Bits	160 Bits
Attacks required to find out original Message	2^{128} bit operations required to break	2^{160} bit operations required to break
Attacks to try and find two messages producing the same MD	2^{64} bit operations required to break	2^{80} bit operations required to break
Speed	Faster, only 64 iterations	Slower than MD5, Required 80 iterations
Successful attacks so far	Attacks reported to some extents	No such attach report yet

Figure 3. Comparison between SHA and MD5 [13] [8]

The above table hold proof for using SHA because of its high security, bigger digest and its ability to thwart attacks. The selection of SHA512 [10] for this project is to have a large digest size to prevent any imminent collisions.

Hash values of system executables can be stored in the secure database and any changes can be to these values indicate potential threats or errors. Also the loader verifies the integrity before the executables are loaded from the memory. In such cases, malicious code is not loaded into the system hence thwarting off any potential threats the code may carry.

AIX ships SHA256 hashes and signatures for various system files that are critical for system operation and hence need to be monitored. These hashes/signatures (and other file-security attributes) are automatically included during various package installations and are captured in the TSD, along with the signatures and corresponding digital certificates required for signature verification.[5]

6. CONCLUSION

In conclusion, it is seen that trusted computing techniques implemented by the Intel, TCG and IBM have much potential to stem any security violation and can be efficiently integrated into existing operating systems to deliver a secure user experience.

The paper has also covered the various components in the implementation of such an environment in Linux and also the key concepts supporting the same. However the discussed approaches are limited to only specific operating systems. Future work may entail a generic trusted environment which provides security along with providing the user with a seamless experience.

ACKNOWLEDGEMENT

The work reported in this paper is supported by the college [BMSCE, Bangalore] through the TECHNICAL EDUCATION QUALITY IMPROVEMENT PROGRAMME of the MHRD, Government of India.

REFERENCES

[1] *"The MD5 Message-Digest Algorithm"* R.Rivest , MIT Laboratory for Computer Science and RSA Data Security, Inc. April 1992

[2] *Trusted Computing and Linux*, K Hall, T Lendacky, E Ratliff, K Yoder- Linux Symposium,2005

[3] *AIX V6 Advanced Security Features Introduction and Configuration*, 2007.

[4] *OSLO: Improving the security of Trusted Computing*, Bernhard Kauer, 16th USENIX security symposium, Pp.229- 237, 2007

[5] *Verifying System Integrity*, Sourabh Desai, George Koikara, Pruthvi Natarajan and Ravi Shankar, 2009

[6] *Intel Trusted Execution technology*: White paper 2012

[7] Younan Y. *25 Years of Vulnerabilities*: 1988-2012[J], Sourcefire Crop, 2013.

[8] *"A review of Comparative Study of SHA and MD5 algorithm"*, International Journal of Computer Applications (0975 – 8887) Volume 104 – No.14, October 2014

[9] *The Untapped Potential of Trusted Execution Environments on Mobile Platforms*. Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan, 2014

[10] *"Review paper on Secure Hashing Algorithm and its variants"*,Priyanka Vadhera, Bhumika Lall, International Journal of Science and Research (IJSR) ISSN (Online): 2319-7064,2014

[11] *"Overview of Linux Vulnerabilities"*. International Conference on Soft Computing in Information Communication Technology (SCICT 2014)"

[12] *Trusted Execution Environment: What it is, what it is not.* 2015 IEEE Trustcom/BigDataSE/ISPA

[13] *"A Comparative Analysis Of SHA and MD5 Algorithm"*Piyush Gupta et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 4492-4495