Securing the Operating System

A Software Based Approach in Linux

*Prabhav S, *Madhav V Deshpande, *Rakshak R Kamath, *Rohan N, #Rakesh Kumar, $Latha N R
*Undergraduate Students, #Head, CCSD/CIG, ISAC, Bengaluru, India
$ Assistant Professor Dept. of Computer Science and Engineering BMS College of Engineering Bengaluru, India

***Abstract*—*Linux is one the most widely used operating systems. With its inherent ubiquity come its many flaws. In this implementation the aspect of system security is considered. We have implemented a kernel patch which isolates the execution of ELF files in Ubuntu. The signatures of these files are verified before the loading and execution can proceed. We verify the path of the file and its hash value. Any change in path or the contents of the file and hence change in its hash value will prevent it from being executed and hence a safe execution environment is provided. The kernel with the security patch takes an average of 0.006 seconds more time than the kernel without the patch which means the user of such a system will not feel any delays in execution.*

Keywords *- linux; kernel; system security; cryptographic hash;*

I. 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 as seen in [7] and [8]. One such problem is unauthorized access control hence gain of system access. IT infrastructure has been constantly bombarded by malicious content that is more complicated, sophisticated and easier to hide. As stated by the McAfee Threat Report [4], the number of unique malicious and unwanted programs, in the first and second quarters of 2012 alone, has increased by 8 million. These 8 million samples are inclusive of dangerous rootkits which are a growing menace. According to the threat report over 100,000 rootkit variants have been reported in the last 14 quarters.

A similar report from Kaspersky Lab [4] shows a tenfold growth in malicious programs in 2008. The increase from 2.2 million to 20 million malicious programs shows that malware is increasing without inhibition. This results in software security being crucial for any organization. Linux provides some security by discretionary access control as a means of restricting access to objects based on the identity of subjects and/or groups to which they belong to. This mechanism provides same privileges to a group. Any processes created by a user in the group will have the same privilege. This means that one flaw in the software eventually can lead to compromise of data of the entire group. Other threats include the act of giving an application more permission than they require and also replacing system modules with rogue modules. Thus the mechanisms provided by Linux can clearly be circumvented.

Hence, we have made an attempt to increase the security of Linux systems through the implementation of a Trusted Environment for ELF files. We compare the signatures of these file with the signatures in a database before execution. The files which fail verification are written to log files which can be checked by the user who can accordingly modify the database. Two user space daemons monitor the integrity of database and log files respectively. We also have an application through which the database can be changed. This software implementation can be deployed easily as the Trusted Environment is in the Linux kernel itself without the need for any other specific software and external hardware. Also the fact that similar Linux kernels are used by many distributions means that the environment can also be deployed on any of these distributions. Although the application is in the user space and must be downloaded, it is easier than using restricted software implementing security mechanisms or hardware dependent mechanisms which may not be deployable on a large scale.

II. RELATED WORK

This section gives a brief overview of the existing work in Trusted Environment developed by various organizations. An important distinction is that all these implementations are hardware based implementations. Firstly, IBM implemented a secure computing environment in UNIX. According to [2] Trusted Execution refers to a set of features which implement advanced security policies to increase the trust level of the computer. Through a set of hardware switches and the AIX, IBM has been successfully able to produce a version of UNIX operating system with augmented security. They implement security policies to measure and verify integrity of the system files using a database called Trusted Signature Database. A trusted file is a file that is paramount to uphold the security of a system and on being compromised can lead to a breach in security. Also, they use a Trusted Execution Path which defines a list of directories that contains executables. This means the loader allows only those binaries which are loaded from any of the directories

Intel has also developed a hardware based approach to Trusted Execution. It is called Trusted Execution Technology. It defines Trusted Execution technology as “a highly versatile set of hardware

extensions to Intel chipsets that with the appropriate software can enhance the security capabilities" [5].This works by creating Measured Launch Environment (MLE) to compare the critical components of launch against a good source. On boot, the measured launch of BIOS to check integrity ensures startup security and subsequently the measured launch of hypervisor which in turn loads the software. On successful boot, the trust level of the platform can be reported. Intel TXT also defines roots of trust for successful evaluation of the environment.

ARM implements its version of Trusted Environment called TrustZone. ARM defines TrustZone as "The TrustZone hardware architecture aims to provide a security framework that enables a device to counter many of the specific threats that it will experience. Instead of providing a fixed one-size-fits-all security solution, TrustZone technology provides the infrastructure foundations that allow a designer to choose from a range of components that can fulfil specific functions within the security environment" [3]. It works by partitioning the system into a normal and secure world. The secure world is for security sub system whereas the normal world for everything else running in a time-sliced fashion.

III. PROPOSED SYSTEM

In the following section we explain in detail the modules and process involved in the development of Trusted Environment in software in Linux using the important aspects from the previous section.

A. Startup Security

On boot, the database is loaded into memory and this hash value is calculated for the entire image. A SHA512 hash value is generated and this is compared with the hash value of the system before its last shutdown. This value keeps changing as the database is changed during system operation. If the verification is successful, the Trusted Verification is enabled in the kernel loader. Else, a notification is sent to the administrator to manually verify the integrity of the database and restart the system for the changes to be applied.

B. Trusted Verification

This module performs the integrity checks on the ELF files before they are mapped into the memory and executed. The kernel receives and passes the user land pointer of the ELF file which is the absolute path of the file for verification. This path is then verified with the one in the database and only on successful verification, the signatures are verified. The signature again is a SHA512 hash value. In the kernel the hash value is calculated using a crypto API. Only when both the path and the hash value of the executable is verified will the process continue, else he path of the file is written into the logs.

C. Trusted Database

This is the internal database which stores the path and hash value of the executables in a specific format. This database can only be changed by the Trusted Application to add, remove or modify the database. Fig.1 shows the database.

FIGURE-1 TRUSTED DATABASE

D. User Space Processes

User space processes include the Trusted Application, log files and two daemons. The application is the only way to modify the database and provides the user a GUI based method to interact with the database. The log files in question are the kernel log file and the executable log file with the former the place where the kernel outputs the path of the file that failed verification and the latter a formatted file to store the absolute path of the executables not in the database. Also, there are two daemons monitoring the integrity of the database and also monitoring the log files for any update. Any change in these two files cause a notification to the administrator and the opening of the application to manually verify the integrity of the database. On any change the hash value of the database, and the database images in the kernel are all updated.

IV. SYSTEM ARCHITECTURE

The architecture of the system is depicted in Fig.2. It shows the four modules along with their operations. The two kernel modules are connected by the kernel thread where the enabling of the Trusted Verification is dependent on startup security check. The database image is passed to the kernel on startup through the virtual file system. Any changes to the database are made to the image in the user space and this is updated in the kernel space only on reboot. The user daemons interact with the Verification module through the logs. They also monitor the user space database image for any changes and interacts with the application as needed.

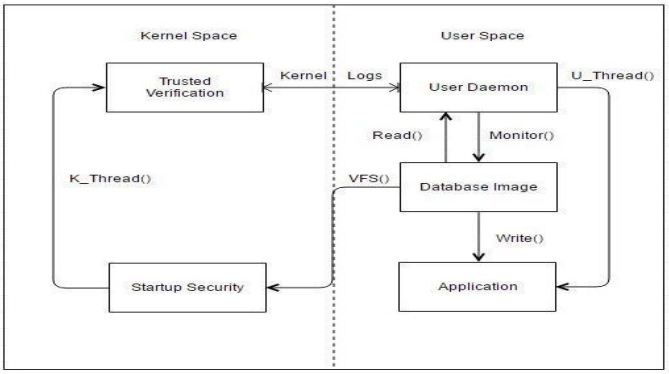

FIGURE-2 SYSTEM ARCHITECTURE OF TRUSTED ENVIRONMENT

V. WORKING OF THE SYSTEM

On boot, the integrity of the system is checked. On failure, a notification is sent to the user informing of the same and post boot, the application is opened so that the administrator can make effective changes and reboot. The Trusted Application is secured by a password known only to the administrator. After the startup security check has been successfully passed, Trusted Verification is enabled. Any ELF file now being executed is checked before being executed. On failure of the check, the absolute path along with the current timestamp is written into the kernel log file. Subsequently, this file is check to find ELF file paths as the kernel can log anything. These selected paths are written to the executable log file and as soon as this file is changed the application is opened. Any changes made to the database are reflected on their images and also on the signatures only after boot. The state where the verification was a success is the secure state and the other being the unsecure state. The Trusted Environment ensure that the system is always in the secure state or tries to reach the secure state.

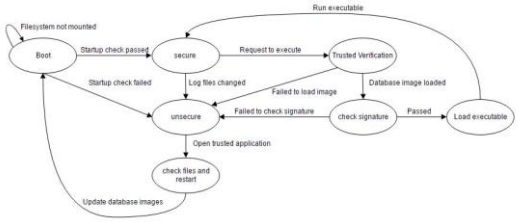

FIGURE-3 TRANSITION DIAGRAM OF TRUSTED ENVIRONMENT

VI. VERIFICATION OF SYSTEM INTEGRITY

System integrity can be defined as using the state of the system that is termed as good and one which cannot be changed as a reference to check the state of the system periodically. According to [4] the requirements of a good integrity tool are "Integrity Measurement", "Lockdown" and "Monitor and Protect". Integrity Measurement is to provide the administrator tools to detect changes to the system. In case of any change there should be a mechanism of locking down the baseline information to prevent any intruder of modifying the system. The last requirement should provide means to monitor files identified as critical files which are being monitored.

Changes to the system can be measured using hash values. The [1] SHA algorithm proves superior to any other algorithm. In [4], SHA256 hashes are calculated and used. In this implementation, SHA512 hashes are used to increase the size of the digest and to prevent any imminent collisions. [1] and [6] help in analyzing the SHA512 and MD5 hashes respectively. The fact that SHA512 has a bigger message digest means that there are many more possibilities as compared to the MD5 sums and this leads to fewer collisions [9]. Also it takes less than 64 bits to crack the MD5 digest and many attacks have been reported but it takes more than 120 bits to crack the SHA512. No successful attacks on the SHA512 have been reported yet.

TABLE I. COMPARISON BETWEEN MD5 AND SHA512

Keys for comparison	MD5	SHA512
Output size in bits	128	512
Block size in bits	512	1024
Rounds	64	80
Max message size	Unlimited	2^{128}-1
Speed	Faster with only 64 iterations	Slower with 80 iterations
Attacks to find original message	2^{128} operations required to break	2^{512} operations required to break
Successful attacks	Attacks reported	No such reported attacks yet

VII. PERFORMANCE ANALYSIS

A performance analysis for Trusted Execution was conducted by running an executable for a certain number of times on both the kernels. Fig.4 shows how long it takes the generic kernel to execute the command "LS" 10 times and the same criteria is carried out on the kernel with Trusted Execution in Fig.5. It can be seen that the latter has an average execution time of 0.006 seconds more than the generic kernel. This was true for all executables tested with a range of 0.002 seconds higher or lower.

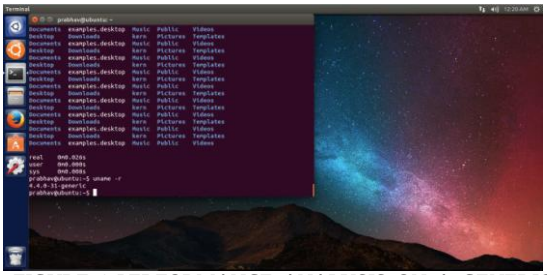

FIGURE-4 PERFORMANCE ANALYSIS ON A GENERIC KERNEL

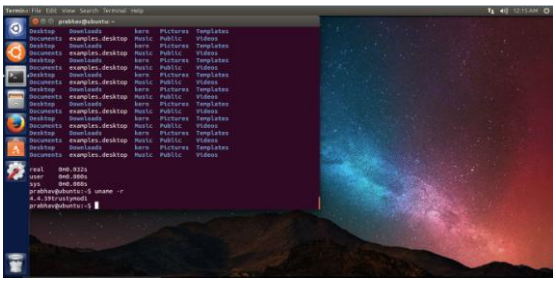

FIGURE-5 PERFORMANCE ANALYSIS ON A CUSTOM KERNEL

ACKNOWLEDGMENT

We are grateful to BMS College of Engineering for having provided us with the facilities needed for the successful completion of this paper. We are also grateful to ISRO Satellite Center for their constant help and support. The work reported in this paper is supported by the college through the TECHNICAL EDUCATION QUALITY IMPROVEMENT PROGRAMME
[TEQIP-II] 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] AIX V6 Advanced Security Features Introduction and Configuration, 2007.

[3] ARM, "Security technology building a secure system using trustzone technology (white paper)." ARM Limited (2009).

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

[5] Intel Trusted Execution technology: White paper 2012.

[6] FIPS, PUB. "180-4-Federal Information Processing Standards Publication-Secure Hash Standard (SHS)-National Institute of Standards and Technology Gaithersburg." (2012): 20899-8900.

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

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

[9] "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