Digital Signature for Signed Executable

Lokesh[1] Basvaling Swamy[2]
[1,2]Assistant Professor
[1,2]Department of Computer Science & Engineering
[1]GNDEC College [2]LACE Bidar

Abstract— This paper describe the design and implementation of signed executable module in Linux Platform where it allows only to load and run trusted application/code and deny the execution of untrusted execution of application/code. The signature shall be provided using the Asymmetric Key Cryptographic Infrastructure. A careful study will be taken up to design an efficient signing framework which can manipulate the existing object file formats recognized by the native operating system platform, to introduce the digital signatures, as part of the object files themselves. It ensures that a piece of code is not altered once it has signed and also identifies the code emerging from a specific developer or signer. It is based on the idea that the author signs the application/code using private key and the end user uses public key to verify the signature.

Key words: Hashing, Public Key Encryption, Authentication, Integrity, Trusted Execution Technology, Digital Signature

I. INTRODUCTION

Today's Operating System is built mainly for general purpose usage. Security is also built into Operating System but priority is given to its usage. Using today's Operating System for machine critical application where security of environment its high priority is a big challenge. Security has become a prime factor of concern, with the advent of the Internet technologies. Rapid developments in the field of computing have yielded enormous potential for the applications to be run on devices ranging from massive super-computing clusters to tiny handheld devices. However with the advancement in the applications, there has also been a tremendous outburst in the development of malicious software and intrusion systems. These software range from a simple script which could get executed on a normal web browser of any desktop system, to a whole framework with abilities of self duplication and self learning, that can bring down massive networks of computing systems at a single instance. Intruder may add scripts that do not change the website's appearance. These scripts may "silently" redirect you to another web site without you even knowing about it. This redirection to another web site may cause infected programs to be downloaded to your computer. These infected programs are generally designed to allow remote control of your computer by the attacker and to capture personal information, often related to obtain the information of credit card; banking information and data used for identify theft. Thus newer and newer methodologies have to be evolved and deployed to prevent the systems being attacked and taken down by malicious software. Rather than a detection system that may detect suspicious activity, protection is needed from fast acting attacks, in the realm of Internet computing. A prevention system must thus identify and stop malicious attacks before they do damage and have a chance to infect a system. Hence, executables must be authenticated and protected against integrity attacks and is conveniently achieved by digitally signing it. This is implemented in Linux operating system where signed executables help users overcome the problems and threats. A signed executable will allow trusted software to be run in any system.

Digital signing of the object code or the executable helps to ensure that application/code is not tampered. This proposed framework attempts to bridge the gap in the existing security technologies and provide an integrated solution to overcome the challenges mentioned herein [1] [2].

II. DESIGN ISSUES

Signed Executable in Linux Operating System is primarily focused on two integrity guarantees:

- Prevent the modification of authorized executables, and
- Prevent the addition of unauthorized executables.

This paper is mainly concerned about the implementation of the digital signature mechanism. The design is to digitally sign individual executables rather than using extended file system attribute [3] or a signature database. The advantage of using extended attributes is to allows all files to be signed (i.e. configuration files, database, C programs), on the other side the disadvantage is it does not work on file systems that do not support extended attributes or remote file systems. As remote file systems are problematic because we need an additional mechanism to ensure that the remote server is presenting the true extended attributes.

The signature database mechanism has the advantage that it does not require the modification of the executable itself, but on the other side disadvantages are it needs to be updated every time when a new executable is added or modified and also the entire database has to be signed which requires invoking the authorization process on every update. The other is that the system administrator has to manage two separate files, the executable content and its signature, instead of one.

In the proposed design the signature is attached to the executable content. The advantage of this is it deals with the single container, and the implementation requires small kernel modifications. To allow flexibility, a standard format called Executable and Linkable Format (ELF) [4] is used for executable binaries, the PKCS#7[5] formats for storing our signatures, and the X.509 [6] format for storing public key certificates.

The proposed system is vulnerable to two attacks. The first attack replaces the public key which is used for verifying signatures by the attacker public key. This is possible as the public key is stored in a file, /etc/certificate. This attack can be tackled by using a secure boot mechanism [7].

The second attack is about overwrite or downgrade attack, the intruder in this attack has gained access to a machine and copied, for example glFTPd which is the current signed version of the ftp daemon. When the next glFTPd bug is discovered after a month or so, the system administrator installs a new and improved signed version of the ftp daemon. The attacker at this stage replaces the new signed version with the older signed version, which has the know bug and the intruder presumably can exploit it, as the attack is undetected by the current system since it does not keep any state on individual files.

The overwrite or downgrade attack can be prevented by creating a new key-pair and resigning all executables or by keeping a signed revocation list. As this list will grow arbitrarily, records from the list cannot be deleted; the list should be purge by creating a new key and resign all executables. In the current implementation none of this is supported.

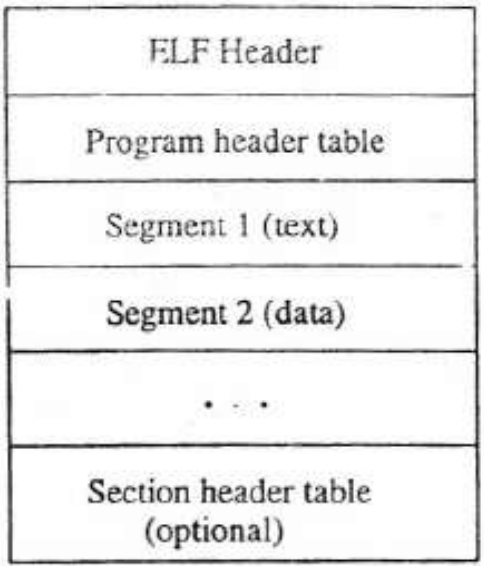

Fig. 1: ELF object file format (execution view) [1].

III. IMPLEMENTATION

The signature checking for executables is done by using the Executable and Linkable format (ELF) [4]. The ELF is the standard file format adopted by different operating systems and different platforms for executables, object files. It distinguishes three types: executable files, shared object files, and reloadable files. Since the implementation is dealt with program execution the focus is done on executable files, shared object files. There are two views of ELF object files format: linking, and execution. The proposed implementation deals with the signature on a per executable granularity, so the focus here is only on the execution view. The ELF object file format (execution view) consists of four parts, as shown in Figure 1: an ELF header gives global information on the object file, like its type and intended platform. The program header table lists the segments, and provides their characteristics, such as the type of the segment, size, and offset. The segment contains the actual code and data that will be loaded into memory when executables started. The files end with the optional section header tables that store information used during the program creation.

In the proposed mechanism the signature checking is implemented using digital signature as shown in figure 2. Here the developer or signer will take the one way hashing of the data and text segment part of ELF where the actual code lies using md5[8] method, next it is encrypted using RSA method with the help of private key forming the digital signature. The digital signature is added to the option section header table of ELF forming the signed executable. At the run time the kernel module will verify and execute it, before executing the kernel module will decrypt it using RSA method [9] with the help of public key. It compares the result i.e decrypted value with the hashed value. If it matches then the control is transferred to the loader where the execution of the executables happens.

IV. FUTURE OUTLOOK

In the future system cache mechanism can be implemented which will increase the performance of the program execution. As the signature verification slows down the startup time of the program. To avoid the loss of performance the signature verification should be avoided whenever it is possible as the signature does not need to be computed every time whenever it is executed. Suppose the kernel knows it has verified an executable file before and has not changed afterwards, then it can reuse the previous result.

Digital signature module can be added as a part of the kernel module to form an additional feature in Linux operating system, thereby forming the trusted platform module to increase the security of the system.

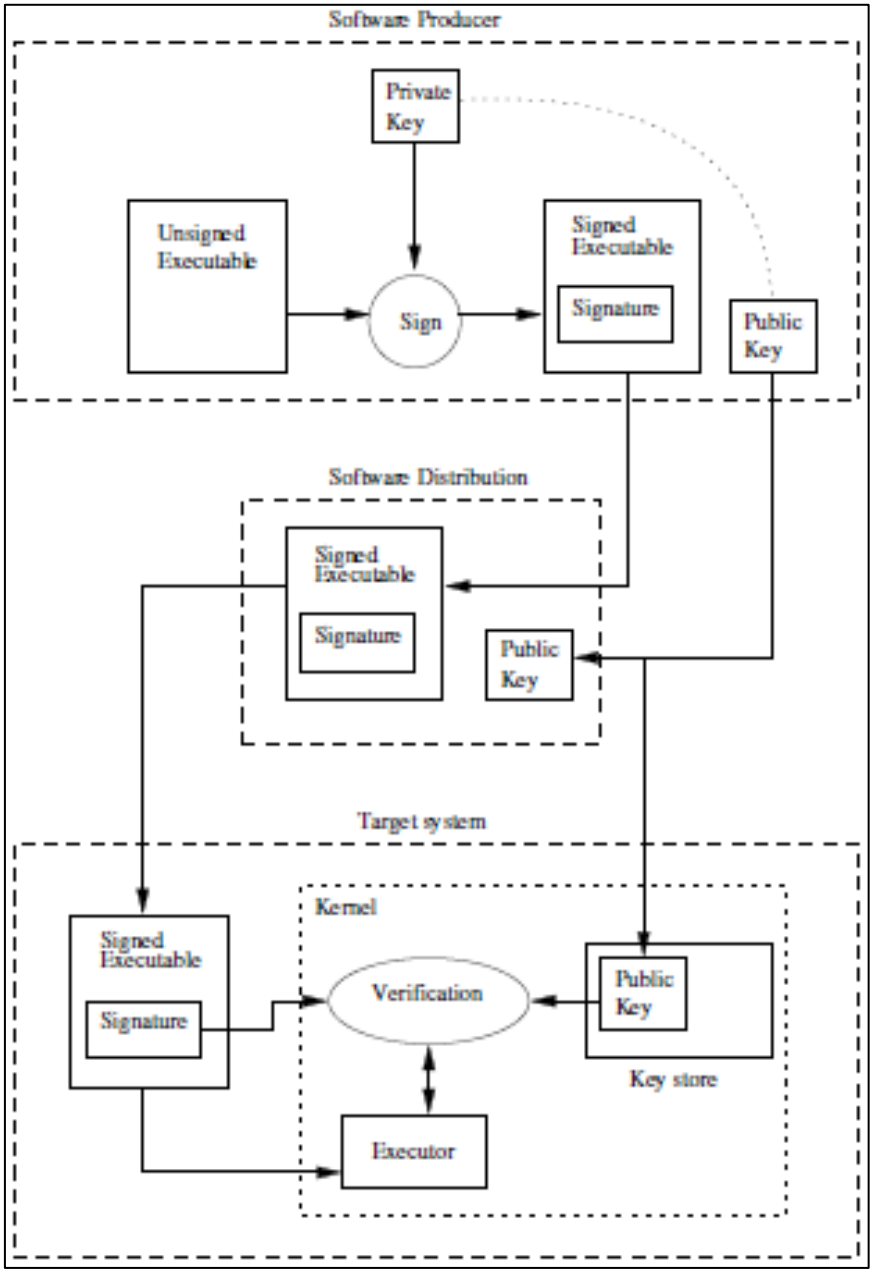

Fig. 2: Verification at the run time of executables [10].

V. CONCLUSION

In this paper a new model named Signed Executable in Linux Platform has been developed. It focuses on strong integrity guarantee: Inability to tamper the executables once it has been signed. It also focuses on the prevention of execution of malicious code during the run time by checking and matching the signature of each executable. The signature checking takes very small over head at load time. It is therefore does not impact machine performance from the end user point of view. The signature is created from secure hash function MD5 and RSA methods.

REFERENCE

[1] L. van Doorn, G. Ballintign, and W.A. Arbaugh. Signed executable for linux. Technical report CS-TR-4259, University of Maryland, 2001.
[2] Sailer, R., Y. Zhang, T. Jaeger, and L. van Doorn, Design and Implementation of a TCG-based Integrity Measurement Architecture, IBM Research Report RC23064, January 16th, 2004.
[3] IEEE. POSIX 1003.le Draft Standard, Access Control Lists, October1997.Withdrawn,http://www.guuge.de/winni/posix.le/download.html.
[4] Intel. Tool interface standard portable format specification (version 1.1), October 1993. Intel order number 241597.
[5] B. Kaliski. PKCS #7: Cryptographic Message Syntax (version 1.5). In Internet Request for comments (RFC) 2315. March 1998.

[6] X.509. ITU-T Recommendation X.509 (1997 E): Information technology- open Systems Interconnection – The Directory: Authentication Framework, June 1997.
[7] W. Arbaugh, D. Farber, and Smith. A Secure and Reliable Bootstrap Architecture. In 1997 IEEE Symposium on Security and Privacy, Pages 65-71. IEEE, 1997.
[8] H. Dobbertin. Cryptanalysis of MD5 compress, May 1996. Presented at the rump session of Eurocrypt '96.
[9] A. J. Menezes, P. C. Van Oorschot, and S.A. Vanstone. Handbook of applied cryptography. CRC Press,1997.
[10] Luigi Catuogno, Ivan Visconti: An Architecture for Kernel-Level Verification of Executables at Run Time. Compute. J. 47(5): 511-526 (2004).