Verifying the Integrity of Shared Libraries

Richard Paul
Software Engineering
The University of Auckland
rpau013@ec.auckland.ac.nz

Abstract

Users of modern operating systems are susceptible to malicious programs being executed unknowingly or maliciously modified programs being run under the guise of valid programs. In order to protect computers from malicious agents, the access of programs should be limited. This limiting may only occur reliably if the program authentication method can verify the integrity of the program. The authentication of a program becomes complicated when the programs use shared library routines. The integrity of these program dependencies may be compromised leading to unexpected outcomes when utilised. This paper explores methods for verifying the integrity of programs that utilise one or more library routines and the security each method provides. These methods primarily include the use of cryptographic hashes and the SPEF system.

1. Introduction

Computers in home, business and educational settings are increasingly storing private and sensitive data. This data needs to be protected against the threat of malicious attacks primarily through the Internet.

Such security can be achieved through the use of closed systems. These closed systems do not allow users to add additional peripherals or software to the machine and hence are inflexible and unlikely to be accepted by the public for general computing needs. In order to secure an open system, a computer must provide users with a system of restricting which programs may be executed and what access rights are associated with them. To provide such a secure solution based on this restricted execution environment, programs need to be authenticated by a monitor. To trust the monitor, the monitor needs to verify itself against a reference, the most common solution is that of a secure hardware device that can primarily authenticate the monitor and also allow the monitor to authenticate future execution requests. All security systems outlined in this paper, namely Microsoft's Next Generation Secure Computing Base (NGSCB) [1], Terra [2] and SPEF [3] use hardware to verify the monitors integrity.

Verifying the integrity of a program is relatively trivial when the program is self-contained, i.e. it does not make calls to library routines. It becomes difficult to guarantee that a program is running securely when it utilises shared link libraries as all of these utilised libraries must also be verified.

This paper explores methods for verifying the integrity of programs that utilise one or more library routines. Initially a background is given on trusted computing, the problem domains are then explained. An explanation of two integrity verification methods is given, followed by a discussion of the strengths and weaknesses of the two methods.

2. Background

Verifying the integrity of a program is essential for secure computing. It allows a monitor to detect potentially malicious code and inhibit it from running thus allowing only trusted programs to run.

Due to the complexity of modern programs and the development efficiency achieved through reusing standard libraries, it appears the use of shared libraries will continue to play a significant role in future software development. It is through these shared libraries that complexity arises when trying to verify the integrity of a program. Through the use of libraries the control sequence of a program can no longer be determined by the inspection of just one executable. Instead all library routines utilised by the executable must also be identified and verified. Performing such identifications may prove difficult and time consuming when a program utilises a large number of libraries.

The security risk of compromised libraries is of great concern. A library may be altered to perform entirely different and potentially malicious operations to those originally intended, causing problems with the calling program as well the direct effect of the malicious operations (see figure 1). Possibly a worse effect could be the injection of malicious code in to a library, while still maintaining the libraries functionality. Such injections may allow library

routines to continue with little or no difference in output or performance while also executing malicious operations. Such an attack would be difficult to discover.

In the event that compromised libraries are utilised by a process running with the highest security level of the computer, e.g. root access on a Unix machine, the compromised library may potentially have access rights to alter or control practically all aspects of the computer. Such compromises pose a serious threat to integrity of a computer system.

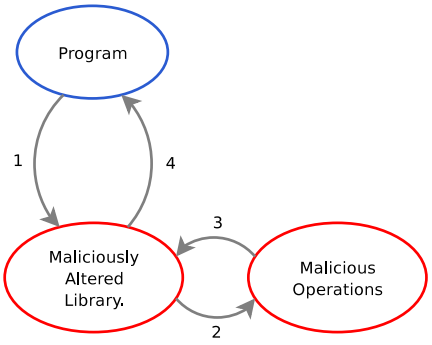

Figure 1. An altered library running malicious code.

3. Integrity Verification Methods

To identify programs accurately a reliable and trusted means of identification needs to be implemented. The Trusted Computing Group [4] *"...develops and promotes open industry standard specifications for trusted computing hardware building blocks and software interfaces..."* It is through this group that the Trusted Computing Platform Alliance (TCPA) standards have been developed. These standards describe the features a trusted computer must incorporate to comply with the open standard for trusted computing. While the TCPA provides a standard for trusted computing, it is not unanimously seen as a complete or perfect solution [5][6].

3.1. Cryptographic Hash

Both Microsoft's NGSCB and Terra are implementations based on the TCPA standards. Using NGSCB terminology, a cryptographic hash is created on the executable file. This hash is known as the *Code ID* and allows for code-based access control. This code ID is stored with in a secure environment on the operating system and the security mechanism can query the code ID repository to see if the program requested to run matches its predefined code ID. Using this method, known as *authenticated operation*, a secure system can allow or deny a program's execution. As shown in the figure 2 any change to a programs file results in a change in the cryptographic hash, or code ID of the file. This resulting executable no longer matches the allowable code ID from the repository and hence is denied access to run as a trusted program or assess any content sealed under the original executables code ID.

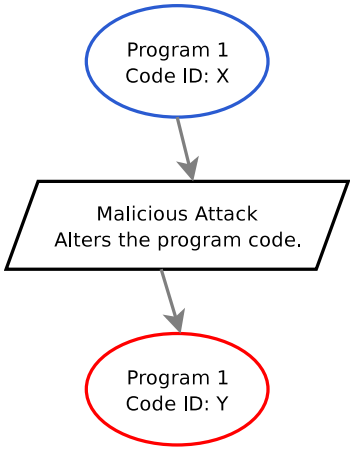

Figure 2. Authenticated Operation.

Neither Microsoft's paper titled 'A trusted open platform', nor the paper on Terra title 'Terra: a virtual machine-based platform for trusted computing' discuss how they plan to address the issue of library calls or interpreters. The following methods of verification are purely speculative and designed to provide a basis for the discussion and analysis of different methods. The assumption is made that utilised libraries are checked in a similar fashion to the programs, whereby they are verified using a cryptographic hash and controlled by an access-control list pending the results of the verification.

The next question is when a library should be verified. With dynamic libraries a call to a library is not known until the call is invoked, this process is known as lazy linking and is most common [7] , hence the only option for verification is at runtime. Due to this constraint calculating a code ID representing both the program and its called library routines requires upfront knowledge of which libraries will be called. In the case that the library has been verified previously, the trusted call to the library invokes a check on the library before it is executed. This is similar to the behaviour outlined by the NGSCB for handling regular program executables. The system becomes complex when a library is updated or modified, either legitimately or maliciously.

Below are two possible methods for dealing with the verification of shared libraries.

3.1.1. Dependency Listing. If a program identifies which libraries it utilises, a code ID may be generated based on

the combined cryptographic hashes of the program and its libraries. The dependency listing would need to state the code IDs of the required libraries at program install time to generate the combined hash code. Problems may arise when different programs require different versions of the same library to function or the library itself is updated.

3.1.2. Separate Code IDs. Each library has its own code ID. When a library is verified it is checked against the code ID for the library. Hence there is no dependencies between the program and the libraries. This ensures simple upgrading of libraries, however it does not allow programs to detect modifications of libraries.

An alternative to this approach is to allow the calling program to include the code ID of the library with its own code ID. This way the program can identify if the library has been altered and either the user or an automated process can determine if the new library is authentic and update its reference to the library's code ID is appropriate.

Although it is unclear exactly how NGSCB and Terra plan to implement library verification, it can be fairly certain it will involve some form of cryptographic hash. It may be difficult to create a code ID system that allows the flexibility of library updates while not complicating the process of updating the verification method for these libraries.

3.2. SPEF

The Secure Program Execution Framework (SPEF) [3] provides a radically different approach to program execution control. Similar to NGSCB and Terra, SPEF requires a hardware module to provide a trusted authentication method. However the way in which execution is limited is drastically different. When a program is installed, SPEF manipulates the program binary in a way that creates a set of constraints specific to the processors unique key. To quote the paper, "*SPEF embeds encrypted, processor-specific constraints into each block of instructions at software installation time and then verifies their existence at run-time.*"

This manipulation of the program binary is merely a shifting of operations in a trivial manner so as not to change the logic of the program. This manipulation is performed at runtime and allows certain constraints specific to the processors unique secret key to be introduced, thus making trusted code recognisable by the hardware module. Any code that does not conform to the constraints will be identified as non-trusted. Generation of a program binary that conforms to the processors constraints is claimed to be computationally intractable. The secret key is stored within the processor itself and is never written to any memory location. Hence with out knowledge of the processor's secret key, generation of binary code that complies to the constraints would prove difficult.

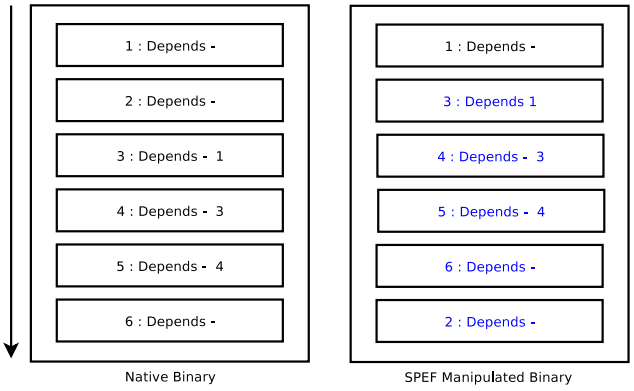

Figure 3. Program code before and after SPEF installation.

The example shown (figure 3) illustrates the logically trivial manipulation performed by the SPEF system to introduce the processor specific constraints. We can see that operation 2 has been moved down the flow and operation 6 has been moved up one. Operations 3,4 and 5 have maintained their order due to their dependency on operation 1. All operations that relied upon a previous operation have retained their ordering, while operations that are not reliant on other operations are shifted to create the desired constraints. This example encapsulates a very simplistic program, in practise the complexity of the dependencies is much higher.

The implications of a successful implementation of the SPEF system, as outlined in [3], protects more than just trusted applications and libraries. Differing attacks performed on a host often inject malicious code into the stack through exploits similar to a buffer overflow. In the case of a buffer overflow the code may still be injected, however the injected code would not conform to the required constraints of the processors unique key. Hence any form of attack that attempts to inject malicious code will fail.

4. Discussion

The two integrity verification methods described provide program execution security in very different ways. The cryptographic hash method verifies the integrity of the executables and libraries before they are run, while the SPEF system validates the executables at run time. Both methods provide a secure base for limiting which programs may be run on the trusted computer.

In the cryptographic hash system an access-control list is stored based on the hashes of trusted executables, these executable are trusted at install time if the programs code ID matches the code ID provided by the distributor. With the SPEF system a similar install time method is used to

created a trusted executable. A program is installed through the SPEF installer that creates processor specific program code. Both methods requires the administrator of the system to trust the distributor of the program. A malicious attack on the distributors website or a 'man in the middle' attack may trick the administrator in to trusting an installation that is not provided legitimately through the distributor. The NGSCB and Terra provide attestation which allows the administrator to be very sure they are interacting with the legitimate distributor.

Library routines often require upgrades to remove vulnerabilities and faults or to add new functionality. Updating a library routine on both systems requires a similar procedure to the initial installation. With the SPEF system the new libraries are installed through the SPEF installer which can immediately be utilised. However with the cryptographic hash methods the program is installed and checked against the provide code ID. Due to the change in code ID of the library, all programs using the library routine now no longer see a library that matches their recorded code ID.

Using the *dependency listing* technique described earlier the resulting code ID will no longer match and hence the program will be denied execution and access to its sealed storage. It may be possible for the system to inform the administrator that the library has changed, and ask them if they wish to update their code ID. This is a very dangerous security mechanism as an administrator my agree to the update hastily if they wish to use the program resulting in the introduction of potentially malicious code that is now trusted.

If the *separate code ID* technique is used the upgrading of the program library will not effect the code ID of the calling program as each program and library has its own code ID. This technique allows program libraries to be easily upgraded, but does not provide any notification to the calling program that the library has changed. In order to inform the calling program of a library change the calling program may contain a reference to the libraries code ID. What the calling program does with this notification is unclear, it may advise the user that the library has changed or more usefully, certain programs may only be classified as trusted given certain versions/code IDs of a library dependency. This would allow critical programs to be only usable with libraries that have been thoroughly tested and verified by the distributor.

In contrast to the code ID based access-control mechanism used by NGSCB and Terra, the SPEF system has a relatively simple upgrade procedure that is identical to the SPEF installation procedure. As with the installation procedure the initial integrity of the program/library can not be verified through the SPEF system itself.

Many virus and Trojans including Backdoor.Emcommander [8] use vulnerabilities in a program to cause a buffer overflow allowing complete control over the compromised computer. The attacking Trojan injects malicious code into the overflowed buffer on the stack allowing the Trojan to run arbitrary code. Neither the NGSCB or the Terra system inhibit such attacks, but rather ensure the library or program is trusted. With the SPEF all code running on the machine in trusted mode is checked against the processor's unique key. Therefore when a Trojan attempts to create a buffer overflow the injected code does not conform to the processor specific constraints and hence the execution will terminate before any malicious code is executed.

The SPEF system verifies code based on blocks. The size of a block is dependant on the implementation. Two examples given in [3] include the 'size of the instruction cache line' or the 'pre-fetch buffer'. Each block is verified before execution begins so no malicious code will ever begin to execute. When SPEF encounters non-conforming code the execution is interrupted, this provides a high degree of safety for the system, however it may cause the program or library with the vulnerability to fail, causing a program error. While this is the preferred approach to an attack as the system is not compromised it may lead to problems. For example if a library routine is attacked in the early stage of system boot, the system becomes unusable.

An issue with the SPEF system not outlined in [3] is the reverse engineering of the processors unique key through the analysis of the SPEF installed executables and libraries. While it may be difficult to discover the unique key, the size of programs and libraries allows for many blocks of code to be examined which may eventually result in the discovery of the key. If the key is compromised the entire system becomes compromised as all installed programs use the same key and programs may be installed without the SPEF installer that will run as trusted code.

The two methods for verifying the integrity of shared libraries are mutually exclusive. It may be possible to integrate the two solutions to provide a system that is both resistant to vulnerabilities in trusted code and provides attestation to verify the authenticity of the installed programs. Systems like the NGSCB provide more than just attestation and program identity verification, additional features including the encryption of sensitive data to restrict its access to a particular program provide users of such a system with a highly secure system. Through the combination of the hardware components and integration of SPEF in to a trusted computer, as defined by TCPA, authorised programs and libraries could be installed that are resilient to attacks that attempt to exploit their vulnerabilities, and can utilise all features provided by NGSCB, Terra or similar system.

5. Conclusion

Both identity verification methods (cryptographic hash and SPEF) outlined above provide a strong base for running programs in a trusted environment. While NGSCB and Terra do not specifically outline how they protect and manage libraries, some logical speculative ideas based on the methods used for protecting regular executables have been discussed. In contrast the SPEF provides an in-depth view of how all program code is manipulated to comply with the specific constraints of the unique processor key.

It can be seen that while the NGSCB provides a more complete security mechanism providing support not only for file execution rights, but also encryption of sensitive data and restricted access to programs and data, its code ID based executable and library verification method may cause unwieldy library upgrades. The SPEF system on the other hand does not guarantee the library version is identical, but rather that the library was installed through a trusted measure. This allows for simple library upgrades but a lack of library version control.

When analysed under the threat of malicious code injection, it can be seen that the cryptographic hash method will not provide any protection against security vulnerabilities within software programs, but rather it protects the files stored on the disk. Under the same threat the SPEF will detect that the injected code does not conform to the processors constraints and disallow the attack.

It can be seen that through a combination of cryptographic hash systems, like NGSCB and Terra, and SPEF, a system that protects the file system and reduces the risk of software vulnerabilities, could be created to enhance the security of a trusted computer.

Acknowledgements

The author would like to thank Jason McCamish for helping with the proof reading of this paper.

References

[1] England, P.; Lampson, B.; Manferdelli, J.; Willman, B, *A trusted open platform*, IEEE Computer, Vol.36, Iss.7, July 2003, Pages:55-62

[2] Garfinkel, T.; Pfaff, B.; Chow, J.; Rosenblum, M.; Boneh. D., *Terra: a virtual machine-based platform for trusted computing*, ACM Symposium on Operating Systems Principles, 2003, Pages:193-206

[3] Kirovski, D.; Drinic, M.; Potkonjak, M., *Enabling Trusted Software Integrity*, Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, 2002, Pages:108-120

[4] Trusted Computing Group, Available on: `https://www.trustedcomputinggroup.org/`, 2004-10-22

[5] Arbaugh, B., *Improving the TCPA specification*, IEEE Computer ,Volume: 35, Issue: 8, Pages:77-79, August 2002

[6] Huang, A., *The trusted PC: skin-deep security*, IEEE Computer, Volume: 35, Issue: 10, Pages:103-105, Oct. 2002

[7] Dean D., *The Security of Static Typing with Dynamic Linking*, ACM Conference on Computer and Communications Security, Pages:18-27, 1997

[8] Symantec Security Response - Backdoor.Emcommander, Available on: `http://securityresponse.symantec.com/avcenter/venc/data/backdoor.emcommander.html`, 2004-10-24