Methods of Securing the Bootstrap Process of an Operating System

Akhil Mehra
Department of Computer Science
University of Auckland
ameh010@ec.auckland.ac.nz

Abstract: *A "system is as secure as the foundation it is built upon" [1]. Operating systems have generally trusted the underling bootstrap process that loads the operating system. In order to secure an operating system we must secure the layer that is responsible for loading the operating system i.e. the bootstrap process. In this paper I look at two approaches for securing the bootstrap process of an operating system, A Secure and Reliable Bootstrap Architecture, and Boot Integrity Token System(BITS). The Secure and Reliable Bootstrap Architecture proposes to secure the bootstrap process by proposing a new architecture for bootstrapping which incorporates layered security architecture into the bootstrap process. The Binary Integrity Token System protects the bootstrap by storing important bootstrap information in a secure smartcard. In this paper I propose to analyze the security objectives of both these systems and the techniques and technologies used to achieve these objectives. The paper also aims to provide a comparative overview the two systems based on the security objectives of the systems and the techniques used to achieve these objectives.*

1. Introduction

A computer system is generally viewed in layers. An operating system is as secure as the layer that is built on [1]. It would be naive to believe that an operating system is secure unless we can guarantee the integrity of the layer below the operating system. An operating system is loaded into memory using a bootstrapping process. In order to guarantee the integrity (no unauthorized modifications have been made to the bootstrap process or the operating system [4]) of the operating system it is essential to guarantee the integrity of the bootstrapping process [1].

A larger amount of research has been done to enhance the security of an operating system [1], but few solutions have been provided for securing the bootstrap process of an operating system. This paper looks at two solutions that address the problem of bootstrap security, A *Secure and Reliable Bootstrap Architecture* christened AEGIS and *Boot Integrity Token System* (BITS). AEGIS proposes a new architecture for the bootstrapping process which divides the booting process into several layers. Each layer checks the integrity of the layer above it and only transfers controls when the integrity of the layer above it has been verified [1]. In this way AEGIS ensures the integrity of the bootstrap process. BITS secure the bootstrap process by transferring the important part of the bootstrap process to a smartcard [3]. By booting from a secure device such as a smart card the system aims to provide boot integrity.

The paper gives an comparative over view of the two systems and concludes by stating that sAEGIS (a enhancement of AEGIS) a solution built on top of AEGIS is a good solution for securing the bootstrap because it over comes most of the limitations of AEGIS by incorporating the best feature of the BITS system i.e. the use of a smartcard as a secure storage device.

1.3 Outline

In section two the paper gives an overview of AEGIS and BITS. In section three I will provide my analysis of what the main objective of both the system are and the techniques used by the author to achieve the same. I would further go on to compare the two systems especially the techniques used by the system to establish integrity. In the section 4, the results section I analyze if the two systems meet their respective objectives and discuss the same.

Overview

2.1.1 AEGIS

The main purpose of AEGIS is to secure the bootstrap process. AEGIS does this by "creating chain of integrity checks" [1] for the boot process. AEGIS proposes a new architecture for the boot process. The boot process has been divided in to several layers. The integrity of the very first layer is assumed and forms the root of trust (code that needs

to be trusted in order to secure the system). The AEGIS system requires an addition of a PROM board. The first layer is the initial part of the BIOS and the PROM which contains certificates and verification code to verify subsequent layers. Each layer verifies the integrity of the layer above it. Control is transferred from one layer to another upon successful integrity verification [1]. In the case of integrity failure in any of the layers during the bootstrapping process the system goes into recovery mode. Recovery may come through expansion ROM's in the BIOS itself or through a trusted network source. Once the system recovers it reboots and the whole process is carried out again. This assures that the system will only boot in a secure state. [1]

2.1.2 sAEGIS

AEGIS suffered from two major problems: a) It required a user to trust their system administrator. b) AEGIS was inflexible as it could only load FreeBSD [4]

A new solution was provided to help overcome these problems. The new solution was christened sAEGIS. The main additions by the sAEGIS system to the existing one were the addition of a smartcard and incorporating the GRUB (a flexible boot loader developed by the open source community) into the boot process. The system also added a small program that would be used by the kernel to verify the integrity of essential system files and start system daemons that passed the verification test [4]. "In a nutshell, sAEGIS = AEGIS + GRUB + smartcard + verifier." [4].

Inculcating a smartcard into the bootstrap process yielded two main advantages. It provided a means to authenticate a user and allows a user to have control of have possession of important verification code. Verification code was previously stored in the BIOS and made attacks on the system easier especially attacks by system administrators [4]. By giving a user the possession of certificates and verification code the system claims to prevent attacks by administrators.

For the reminder of the paper both sAEGIS and AEGIS will be referred to as AEGIS unless explicitly stated.

2.2 BITS

BITS aims to guarantee integrity by providing an access control mechanism to the bootstrap process and performing integrity checks on essential system files. The booting

process of the BITS can be better understood with the help of Figure 1. A user is required to be authenticated by a smartcard before the host is allowed to gain access to data on the smartcard. The host and the smartcard also authenticate each other by using a shared secret. Once authorized the host has access to the boot sector on the smart card. Essential information required to complete the booting process is retrieved from a smartcard. Integrity checks are made on the system executable that reside on the host computer by comparing executable checksum values with those stored on a smartcard. On successful completion control is handed over to the operating system and the user may proceed to use the operating system in a normal fashion [3]. Using the above mentioned booting sequence the BITS system aims to achieve boot integrity.

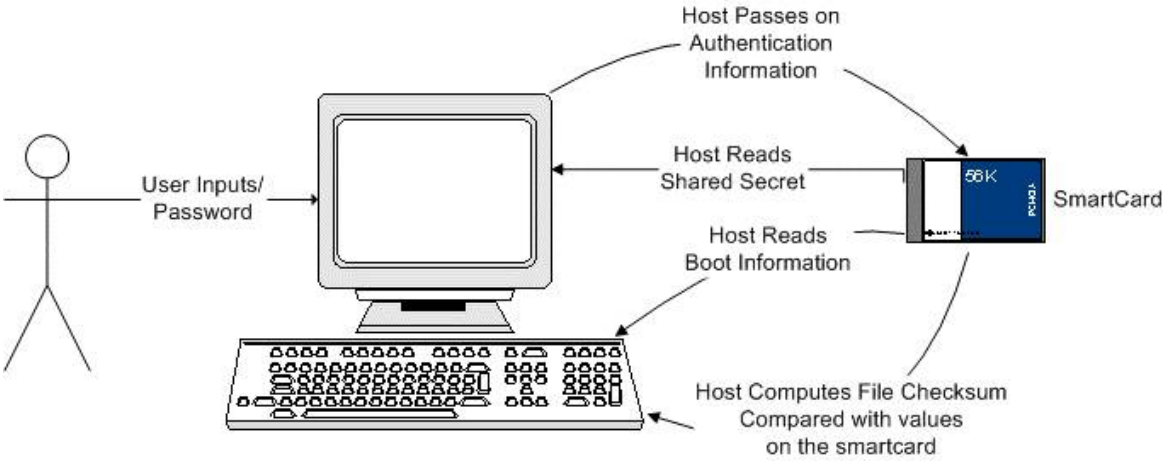

Figure 1: The BITS System

Discussion

3.1 Security Objectives

A well know security principle is that a system is as secure and it weakest link [6]. The main objective of both AEGIS and BITS is to secure the weakest link in a current computer system i.e. the bootstrap. Most computer systems today make no effort in securing the bootstrap. Even trusted systems have assumed the integrity of the boot process [1] to avoid the sheer complexity of dealing with boot strap security. Both AEGIS and BITS identified the aforementioned problem as a weak link in terms of

security in current computer systems and presented respective solutions to secure the same.

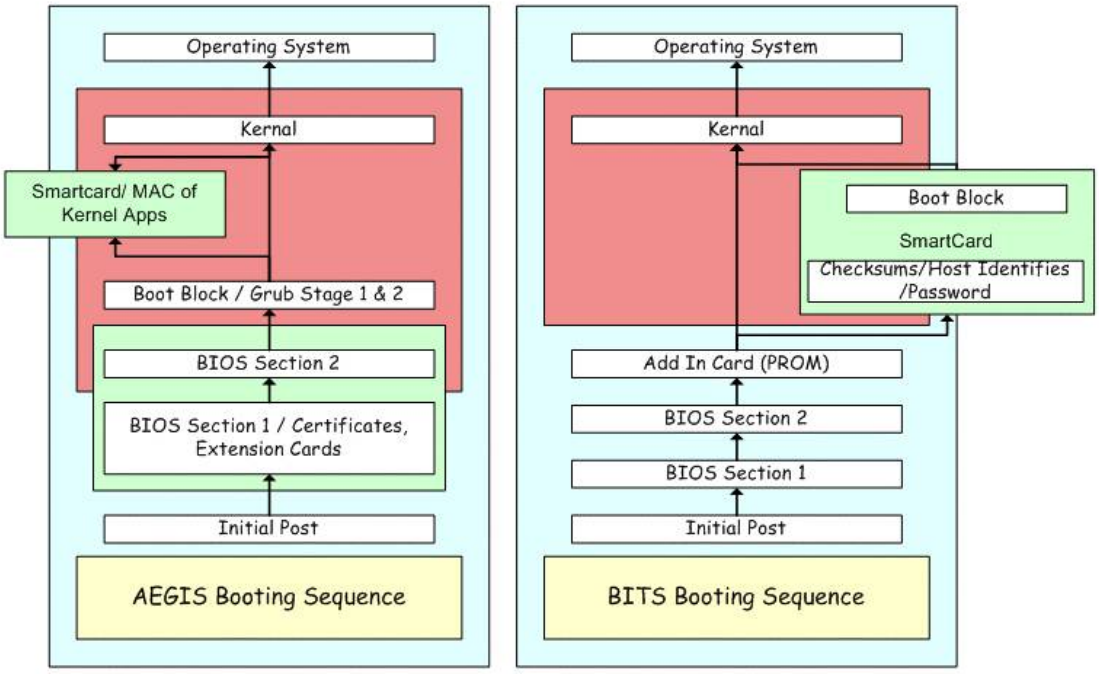

Figure 2: Security Boundaries of the Two Systems

(Adapted from [1, 4])

Figure 2 shows comparative overview of the booting process for both the AEGIS and the BITS system. The red part (darkish part for black and white printouts) of the diagram shows us the security boundaries established by the respective systems. AEGIS has a larger security boundary than BITS. It begins establishing integrity one level lower than BITS by trying also to secure the latter half of the BIOS (BIOS Section 2). Both the systems make sure that the operating system has started in a secure state but do not deal with the security of the operating system once control has been handed to the operating system [1,3,4]. AEGIS and BITS have also incorporated a smartcard into the booting process. The main purpose of the smart card in both systems is to provide access control to the system and act as a secure storage device.

While the primary objective of both the systems is to secure the bootstrap and ensure bootstrap integrity the secondary objective of the systems seems to be to provide access control to the bootstrap. Access control in both the systems seems to have two major objectives:

A. Ensure that a user is authorized to use a particular computer system.
B. Guarantee that a user is authorized to use a particular operating system on a certain system.

Objective A (ensuring a user is authorized to use the system) is achieved in BITS by making the user authenticate them selves to the smartcard before using the system. A user is required to supply a password to the host computer which is then provided to the smartcard to authenticate the user (as shown in Figure 1). The author suggested a number of additional ways such authentication could take place, e.g. biometric authentication, normal user name password etc [3]. AEGIS initially did not require to a user to authenticate themselves, but this was recognized as a limitation and an appropriate solution was provided. The AEGIS solution also incorporated a smart card for authentication purposes. AEGIS currently uses a PIN to authenticate users; however in my opinion a more complicated and secure authentication solutions can easily be built into the system.

Access control provided in AEGIS and BITS also tries to ensure that a user only boots an operating system which he/she is authorized to use (Objective B). In a multi user operating system environment an authorized user may wish to gain access to unauthorized data [3]. This may be done by making a certain system into a dual boot system. A dual boot system is one where two or more operating systems exist on a single machine**.** When the system boots a user is given the option of choosing the operating system he would like to use.

In a dual boot system access to unauthorized data may be achieved by booting another operating system and using it to access data that he/she is unauthorized to view [4]. For Example: User hacker is not allowed to access directory "Confidential" when using WindowsXP operating system on machine "Public". A user trying to gain unauthorized access to directory "Confidential" may load another operating system E.g. Linux and they use it to gain access to directories "Confidential". This may be done by mounting the Windows partition on the Linux operating system by using the mount command. Limiting the user to a certain type of operating system helps solve this problem.

In the BITS system making sure that a user could only boot a particular operating system was a major goal and must have been taken into account during the design phase. The AEGIS system however making sure that a user only boots an operating he is authorized to use is inherently part of the system. By making sure that the bootstrap process of a system cannot be modified the system makes sure that a user will only be able to boot a particular operating system. In the AEGIS system the author does not state this as an objective but it seems to be an outcome as a result of implementing the system.

Currently the only form of access control available to the bootstrap process of a computer is the BIOS password. The BIOS password only provides a rudimentary form of access control. A BIOS password limits a person to weather he is able to use the system or not, but in no way can control what type of operating system the user may use. More over a BIOS password is quite insecure as there are a number of ways to bypass it. The most popular ways of bypassing a BIOS password are using default password often provided by the manufactures and removing the system battery that wipes out any existing password [13].

The author of both the systems also wants to make sure that their implementations are practical and can be easily adapted to work in current systems. A major goal of both the systems is to make sure that the bootstrap architecture proposed by both can be easily adapted into popular computer architectures. The AEGIS paper states that its solution is meant for the "real world" [1].

AEGIS has a further objective of recovery which enables it to cope with a denial of service attack. If a certain layer of the bootstrap process failed an integrity test, that layer would be recovered from a trusted source. The trusted source may have been an expansion ROM in the system itself or a trusted network host. The whole recovery process takes place without user intervention. This ensured that the system would eventually boot in a secure state [1]. A major advantage of AEGIS over BITS is that AEGIS not only tries to secure the bootstrap process but it also provides a recovery mechanism in case of failure. The BITS system does not address recovery in any way.

3.1 Techniques Used

AEGIS and BITS are quite similar in terms of the techniques used to secure the bootstrap even though at first glace both the protection mechanism look quite different. Both the system use some form of integrity checks on essential system files and provide some sort of access control to the bootstrap where none previously existed.

It is quite interesting to note that techniques used by AEGIS and BITS to secure the bootstrap are quite similar to virus detection programs. A virus detection program identifies virus using virus signatures or some form of patter matching [5]. Similarly AEGIS and BITS identify integrity failures by comparing hash values / checksums (kind of patter/signature matching) of important bootstrap related files with ones stored in trusted sources. The major difference (apart form the obvious difference that a virus detection program runs when the operating system has been loaded into memory and the AEGIS and BITS system run before loading of the operating system) between a virus detection program and the two systems is best described using the principle of "Fail-safe defaults" [12]. A virus detection program usually assumes the integrity of all files and searches for those that have been maliciously modified. The AEGIS and BITS system use a different design approach. They consider all (AEGIS) or a group (BITS only important system executables) of file to be malicious and use only files whose integrity has been established. Thus while a virus detection program does not implement the principle of "Fail-safe defaults" [12] AGEIS and BITS do.

AEGIS system mainly focuses on securing the bootstrap process by establishing integrity of major components of the bootstrap process starting from the BIOS Section 2(Ref to Figure 2), on the other hand the BITS system's main emphasis for securing the bootstrap process is by using access control. This is obvious from the fact that AEGIS verifies the integrity of every file used in the bootstrap process while the BITS system only relies on access control to secure the bootstrap and puts in additional checks by verifying the integrity of important system files.

Access control is quite an obvious measure to help secure a system. By putting in an access control mechanism to the bootstrap, the security of bootstrap certainly increases. BITS focus on access control is made obvious by the fact that the system incorporates two types access control. A user has to be authenticated by a smartcard and

the host and the smartcard authenticate each other using a shared secret (see Figure 1). I fell that this is a good technique to defend as the bootstrap the author has makes use of the design principle of *defense in depth [12]* (two kinds of access control is better than one incase one is breached) [7]. Although the BITS system makes use of the defense in depth principle it does not adhere to the principle of open design [12]. The system would have been much better of if it had utilized asymmetric encryption instead of using symmetric encryption for the authenticating that takes place between the smart card and the host.

AEGIS did not provide any from of access control in initial version but was recognized as drawback and was incorporated into sAEGIS. The AEGIS system requires authentication based on a PIN number which is stored in a smart card. The authentication is based on a simply PIN and thus may be easily bleachable, but in my view one easily corporate a better authentication techniques using a smartcard. [4]

AEIGS also incorporates recovery. But adding recovery the AEGIS system accounts for denial of service attacks. The authors suggest two sources for recovery: expansion ROMS, or a trusted network source. The authors of AEGIS system seem to be motivated towards network recovery [1, 4]. The BITS system does not address recovery from failure in any way and I fell that for the system to be viable recovery has to be addressed. It is important for any protection mechanism to ensure that it has some means of recovery in case of failure.

2. Results

I feel that both the papers have done a wonderful job of identifying a weak link in operating system security and provided pre-emptive solutions for the problem. Till date there have been very few solutions presented for securing a bootstrap process of an operating system. The most popular solution for an infected or corrupted bootstrap has been a virus detection program. Virus detection programs are a means of recovery rather than prevention and thus are inappropriate solutions for securing a bootstrap.

Securing the bootstrap process in a personal computer (x86 architecture) is a daunting task [14]. The authors of both the papers have made it clear that their systems are breach able if the user has access to the hardware of the system. This makes it clear

that the authors are trying to prevent modification of the bootstrap wire software means. There are various attacks that are possible because of the sheer complexity of the bootstrapping process but both the systems make a huge leap forward by providing an architecture or process to help secure the bootstrap process.

Both the systems have also provided good access control mechanism. Although the technologies used may be bleachable both the systems provide a good process to help secure the bootstrap. I feel new and better technologies can be easily incorporated into the two systems.

While the objective of securing the bootstrap and access control are fairly well address the objective of the solutions being viable or practical for the real world is questionable. The systems provide bootstrap security but at the cost of flexibility. A major disadvantage of the AEGIS solution is the fact that the modification of the BIOS makes the whole system rigid in terms of what bootstrap loader may be used with the system. The whole idea keeping the BIOS small is to enable one to be flexible in order to be able to incorporate any operating systems easily [13]. AEGIS restricts the user to a certain bootstrap loader.

The author suggests in sAEGIS the use of a flexible bootstrap loader such as GRUB to over come this problem. I feel that is a good solution in order to enable loading any operating system, but the author is unclear on how exactly this would work. GRUB has two methods of loading an operating system direct loading and chain loading.

In the direct loading method the operating system kernel would be loaded directly once GRUB reaches stage two. There is no intermediary between the boot loader and the kernel. The AEGIS system addresses the verification of this kind of code very well. In the sense we assume that the kernel is verified and booting continues.

GRUB also provides a secondary way of loading an operating system called chain loading. Chain loading is provided in GRUB in order to enable the boot loader to load any kind operating system. “Under this method, the Master Boot Record simply points to the first sector of the partition holding the operating system. There it finds the files necessary to actually boot that operating system [11].” The AEGIS system does not address what all will be verified under this circumstance and how will the verification be addresses. In this situation one would need to verify the master boot record which inter

would verify the operating system kernel. But if we are loading a new operating system then the author leaves us in confusion on how verification will be carried out. Will the new operating system some how conform to the AEGIS booting process i.e. the Master Boot Record (MBR) will ship with verification code that conforms to the AEGIS architecture or both the MBR and the kernel will be verified by GRUB, thus moving away from the concept of "creating chain of integrity checks" [1].

The BITS system seems to be a bit more practical for the real world. In my opinion it would be much easier to incorporate the BITS solution into current computer system. I do not think flexibility is compromised in the BITS system as one may easily configure the smartcard to load and boot various operating systems.

The only doubt I have about my claim is that the BITS system has not been described very well. The author is quite vague about how much of the code is to be moved to smartcard. From the conclusion provided in the paper it would seem that the author wants to move all operating system related code to the smartcard but is limited by a smartcard capability. The system seems to heavily rely on the premise that a smartcard is an extremely secure device and thus making a system much more secure if it boots from a smartcard.

3. Conclusion

In conclusion I would like to state that both the systems AEGIS and BITS have done a wonderful job by identifying a weak link in operating system security and providing solutions to secure the same. Both the systems have aimed provided a protection mechanism for the bootstrap process.

Although at first site both the systems look very different the objectives and techniques used in the system seem to be the same. In my opinion sAEGIS solution seems to be the best of all the three systems because it not only build on AEGIS but overcomes it draw back by incorporates the best features of the BITS system i.e. the use of a smartcard as a secure storage device.

4. References

[1] Arbaugh WA, Farber DJ, Smith JM. A Secure and Reliable Bootstrap Architecture. [Conference Paper] Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097). IEEE Compute. Soc. Press. 1997, pp.65-71. Los Alamitos, CA, USA.

[2] Arbaugh WA, Keromytis AD, Farber DJ, Smith JM. Automated Recovery in a Secure Bootstrap Process. In Internet Society Symposium on Network and Distributed System Security, San Diago, CA, March 1998.

[3] Clark PC, Hoffman LJ. BITS: A Smartcard Protected Operating System. [Journal Paper] Communications of the ACM, vol.37, no.11, Nov. 1994, pp.66-70, 94. USA.

[4] Itoi N. Arbaugh WA. Pollack SJ. Reeves DM. Personal Secure Booting. Information Security and Privacy. 6th Australasian Conference, ACISP 2001. Proceedings (Lecture Notes in Computer Science Vol.2119). Springer-Verlag. 2001, pp.130-44. Berlin, Germany.

[5] Kephart J, Sorkin G, Chess D, White S, "Fighting Computer Viruses", VX Heavens, November 1997. June 5th 2003 <http://vx.netlux.org/lib/ajk01.html

[6] McGraw G, Viega J "Software security principles: Part 1, The chain is only as strong as its weakest link", developerWorks Security, October 1, 2000. June 5th 2003 < http://www-106.ibm.com/developerworks/security/library/s-link.html?dwzone=security>.

[7] McGraw G, Viega J "Software security principles: Part 2, Defense in depth and secure failure", developerWorks Security, November 1, 2000 June 5th 2003

< http://www-106.ibm.com/developerworks/security/library/s-link.html?dwzone=security.>

[8] McGraw G, Viega J "Software security principles: Part 3, Controlling access: Least privilege and compartmentalization", developerWorks Security, November, 2000. June 5th 2003 < http://www-106.ibm.com/developerworks/security/library/s-priv.html>.

[9] McGraw G, Viega J "Software security principles: Part 4, Keep it simple, keep it private", developerWorks Security, November 1, 2000. June 5th 2003<http://www-900.ibm.com/developerWorks/cn/security/s-simp/index_eng.shtml>.

[10] McGraw G, Viega J "Software security principles: Part 5, On keeping secrets, trusting others, and following the crowd", developerWorks Security, December, 2000. June 5th 2003 < http://www-900.ibm.com/developerWorks/cn/security/s-simp/index_eng.shtml >.

[11] "Red Hat Linux 8.0: The Official Red Hat Linux Reference Guide, Chapter 4. Boot Loaders" Red Hat, Inc. 2002. June 5th 2003 <http://www.redhat.com/docs/manuals/linux/RHL-8.0-Manual/ref-guide/s1-grub-whatis.html>

[12] Saltzer J. H., Schroeder M.D., The Protection of Information In Computer Systems. [Journal Paper] Proceedings of the IEEE, vol.63, no.9, Sept. 1975, pp.1278-308. USA.

[13] Subhraveti K D, "Making of an Operating System - A Bottom Up Approach" June 1, 2002 June 5th 2003 <http://www.cs.columbia.edu/~dinesh/papers/book.pdf>

[14] Thomberson Clark <ctho065@ec.auckland.ac.nz> Email Communication Mon, 21 Apr 2003.