

# Using a Trusted Platform Module to enable a Secure Usage of Nodes in Company Networks

## An extension of the AEGIS approach

Carolin Latze and Ulrich Ultes-Nitsche

University of Fribourg, Departement of Informatics, Bd. de Pérolles 90, 1700  
Fribourg, Switzerland  
carolin.latze@unifr.ch, uun@unifr.ch

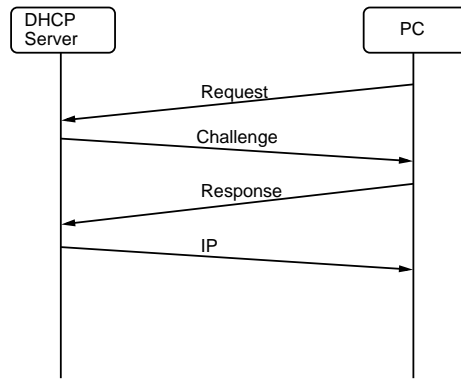
**Abstract.** The secure usage of network nodes in smaller networks like company networks relies on the usage of uncompromised applications. There are many modern approaches to ensure the security - in particular the integrity - of applications, but these solutions assume a secure operating system as base. As this assumption is very critical, there emerged other solutions to ensure a secure bootstrapping, but these do not include applications. Furthermore, most approaches used today just inform the user about a problem but are not able to repair a system automatically. We propose an approach which extends secure bootstrapping by integrity checks of applications. Our approach includes a recovery protocol in the case of compromised applications. The whole process will be secured by a Trusted Platform Module (TPM) to ensure tamper resistance.

## 1 Introduction

Nowadays, most of the attacks on (company) networks come from inside the network and are caused by naïve users, which download malware or open e-mails from unknown senders. Conventional security solutions like an Intrusion Detection System (IDS) may prevent those attacks from network computers. But one huge problem of such solutions is that they are started later than the operating system, which means that they do not prevent attacks coming from a computer in a state before such an IDS is started. To overcome these problems, there are approaches like AEGIS<sup>1</sup> [1] to secure the bootstrapping of a computer. The AEGIS approach is very promising, as it secures really the complete bootstrapping and provides a recovery procedure in case there is some failure. But AEGIS does not provide any checks and recovery procedures for end-user applications. We have extended and modified the approach for the use in company networks. We will introduce more security as we use the new TPM technology to ensure tamper resistance. Furthermore, we will deploy a new recovery protocol for the operating system kernel and the applications, which will be simpler than the one used in AEGIS.

---

<sup>1</sup> Originates from the classical mythology. Aegis is the shield or breastplate of Zeus or Athena.



**Fig. 1.** Flow chart for DHCP request

## 2 The TPM

The TPM as specified by the Trusted Computing Group (TCG) [2], is a module, which provides cryptographic functions and is able to store hashes securely in so called Platform Configuration Registers (PCRs). The specification does not state, that the TPM must be a hardware device which is attached to the motherboard, but this is surely the most secure way to do it. The reason why the specification does not define any deployment details is that the TCG wants to make ubiquity easier. Furthermore the TPM should be a low cost device, otherwise nobody will buy it. This leads to the need of only a minimal subset of cryptographic methods provided by the module and minimal hardware requirements.

The TPM is especially designed to act as root of trust for the collection and reporting of integrity metrics [2] [3]. The idea behind this concept is that a computer can be checked for integrity before usage. The TPM is the instance which stores all the validation data in a secure way to avoid an attack on this validation process.

The module holds confidential information like cryptographic keys in a protected environment. Furthermore, the TPM provides methods to “seal” sensible data to a special state of the platform. This means, that these data may only be released if the platform is in a specified trustable state.

A very important feature of the TPM is that this module can be uniquely identified. It is equipped with a so called Endorsement Key Pair, which is unique. The private part of this key pair is never released from the TPM which ensures the identification.

## 3 Securing the node’s usage using the TPM technology

Our approach for a secure usage of network nodes relies on the fact that there participate only known, trustable nodes. This means that we must provide meth-

ods to identify a node uniquely. Furthermore, our approach must ensure that the nodes, which come online, are in a trustable state.

It is very easy to ensure, that there are only known nodes in the network as we limit our approach to company networks, which have limited size. Usually when buying a new computer, this new machine will be registered by the administrator. In our case, the administrator stores the public endorsement key of the machine's TPM. Every time, a machine wants to connect to the network and retrieves its IP via the Dynamic Host Configuration Protocol (DHCP), the server determines the machine's authenticity using a challenge response protocol as shown in Figure 1. This mechanism allows the exclusion of unknown nodes. Furthermore, this method provides a proof that a TPM is working properly.

Before a machine in the network is able to send the DHCP request, it has to pass several tests on boot up to ensure it is in a secure state. If these tests cannot be passed successfully, the Network Interface Card (NIC) will be blocked by the TPM.

For the remainder of this section, we will explain a concept to ensure secure bootstrapping, which results in a secure state of a machine and what to do if the machine is not in a secure state anymore.

### 3.1 Secure Bootstrapping

In 1997, Arbaugh et.al. proposed a new scheme called AEGIS for integrity checks<sup>2</sup> on boot-up [1], which was extended by themselves by an automated recovery protocol in case of a failure [4].

In [1], the authors describe a secure bootstrapping protocol, where they establish a chain of trust. The chain starts with a small set of code, which is assumed to be valid. This first little program checks the Basic Input/ Output System (BIOS) for integrity. Afterwards, the BIOS validates the operating system kernel. If the hashes are valid, the operating system gets booted. Otherwise, a recovery kernel will be booted, which connects to a so called "Trusted Network Recovery Host", which contains valid images of the desired code. The whole recovery process as described in [4], consists of two steps:

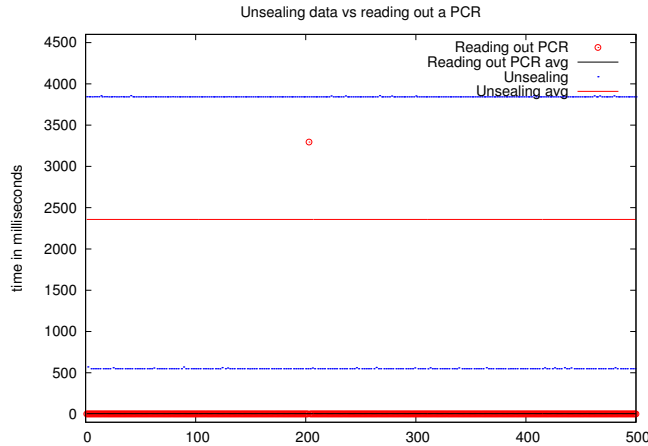
1. Initial authentication, and
2. System recovery.

The initial authentication consists of a key exchange protocol to establish a secure connection between the client and the repository. During the recovery process, mainly two protocols are used. The DHCP is used to tell the server which components have to be replaced and the Trivial File Transfer Protocol (TFTP) is used to transfer the desired component.

We think there is too much overhead, if we equip every normal computer with a recovery kernel. In our approach the boot loader will be responsible for the recovery of the operating systems kernel. If there are some problems

---

<sup>2</sup> With integrity checks we mean calculating the hash value of a component and comparing this value with an older value of this component.



**Fig. 2.** Comparison of unsealing data and reading out a PCR

with the integrity checks, the boot loader switches to network boot and loads a new kernel from the administration server. This means, that the boot loader is allowed to bring up the NIC in such a case but it has to stop the interface when the transmission of the new kernel was successful. This mechanism reduces the overhead introduced by AEGIS.

In the next section, we will present a scheme to check applications, if the operating system is running properly. We will also present an easier recovery protocol only for the applications, which will cause a small overhead compared to the manual recovery by a human administrator.

### 3.2 Measuring the Applications

In our approach there is the additional need to validate and recover applications. First, we have to think about a method to determine the initial correct hash value of an application and their components.

Usually, applications are installed via the internet or a smaller network as installation CDROMs are often not up to date. Nowadays, there is nearly always a file in addition to the program's image or installer, which contains a hash value for the archive to download. Our idea is to extend this file with the following information:

- Which files will be created by the installer?
- Which files will be modified by the installer?

Furthermore, this file has to be signed by the server to ensure its integrity. When the installation process has been finished successfully, the new hashes for this application have to be calculated locally. It is not possible to do this before the application has been installed since most of the binaries are compiled or at least

configured for a special system. But there is no security problem when calculating the hash since the computer has been checked for integrity on boot time. Software updates will run similar to new installations. The computer connects to a trusted server to download the update. This archive will be checked using the hash provided by the server in a signed manner. Furthermore, the server contains a file containing information about newly created and modified files on the client machine. After having installed the application, the hash needs to be newly generated.

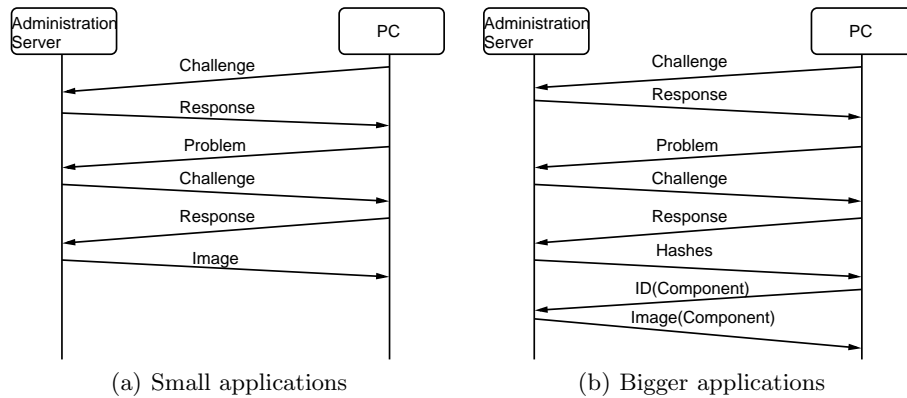
This newly generated hash will then be stored in the TPM. But the number of applications, which may be used on such computers may be rather high and the TPM contains only 16 free usable Platform Configuration Registers (PCRs). One solution for this problem is to concatenate all the hashes of the programs and calculate a new hash over the result as proposed by the TCG Main Specification [2]. This mechanism reduces the need for memory, since we need only one PCR per result. In case of a software update, the old hash needs to be removed from the concatenation and the new one will be added.

But this approach has one important disadvantage: it allows the operating system's validation agent only to decide whether all applications are in a trustable state or not, but it does not help to decide which application is compromised if the test fails. To solve this problem there will be a file on the harddisk, sealed to the bootstrapping integrity, which contains the individual hashes of the applications. This file will be used to identify the compromised application. Due to performance reasons, it is not desired to use this file to determine, whether all application are in a trustable state since reading out a PCR is significantly faster as can be seen in Figure 2. The points around zero represent the time, which is needed to read out a Platform Configuration Register (PCR), whereas the dots show the time needed to unseal a file. Reading out a PCR as nearly a 2000 times faster than unsealing.

### 3.3 The Recovery Protocol

If the validation agent of the operating system kernel detects a problem with the applications, it uses the per application hashes to determine which application caused the problem. After having identified the application, the agent has to establish a connection to the administration server, which serves also as recovery server. This connection is a security problem since the NIC has to be activated, although we know that at least one application is compromised. Therefore, it is very important, that the compromised application cannot be started before the recovery process has finished. Furthermore, no other process should run to avoid problems with dependency resolution. Therefore, we forbid any additional process to start. The validation agent has to monitor the list of running processes to shut down the NIC if there is any disallowed process. The operating system kernel should be configured in a manner, that it is not possible to start an application process until the validation agent signals a successful integrity check.

The blocking of the NIC avoids the direct infection of other computers in the network, but it does not avoid delayed communication through hidden channels.



**Fig. 3.** Recovery Process

There are two types of delayed communication, which have to be handled in our scenario:

1. The virus/worm tries to communicate with an external server.
2. The virus/worm tries to send a copy of itself to other computers.

The first type will not be a problem, since we replace the compromised application before releasing the NIC again. This means, that the server may know, that there was a virus/worm on this computer, but it is not able to use this infection anymore.

The second type can easily be handled. While recovering the application, the old compromised version will be copied to a server, which extracts the virus/worm. When the computer comes online again, the network traffic has to be analyzed for this pattern.

In the beginning of the recovery session, the (compromised) client PC has to connect to the administration server, which is well known in a small network. This server also needs a backup server to avoid an architecture, where this server is the single point of failure. The authentication between client and server is done via a challenge response protocol. In the beginning, the client sends a challenge to the well known server to ensure, that the server has not been compromised. The server replies with a response, which is verified by the client. Only if the response corresponds to the client's expectation, she sends the problem report. Before replying to this specific problem, the server sends a challenge to the client to ensure that the client's TPM is working properly. If the client sends the desired response, the server sends a reply to the client's first request. This may be for instance a new image of the compromised application if it is small enough as shown in Figure 3(a). If the application is too big, the server may send the hashes of the different components of the application. The validation agent then has to determine, which component(s) are compromised. The idea behind this is that the images of some components may be smaller than those of the whole

application, which leads to shorter transmission delays and a shorter recovery process. The client has to check the components of the compromised application and request again only the needed components (see Figure 3(b)).

After having finished the recovery, the computer will be rebooted to initiate the integrity checks again to ensure that everything is now working properly.

## 4 Related Work

The first approach for integrity checks was proposed by Yee in 1991 [6] [7]. He wanted to equip every computer with a secure co-processor, which calculates hashes over the running programs. Even if the co-processor approach is similar to our approach using a TPM, there are some problems. First of all, Yee does not propose any mechanism for the validation of the BIOS. And second, he does not provide any recovery procedure if the integrity checks fail.

Lampson et.al. proposed their idea in 1992 [8]. They calculate message digests over the applications, but do also not provide any BIOS checks and recovery methods.

In 1994, Clark proposed an approach where he used smart cards to enable a secure booting of the operating system [9]. In this approach, the computer is only booted, if the smart card provides the correct authentication information. Furthermore, the card contains checksums for important system executables. But he again does not provide any method for BIOS validation and recovery.

The university of Bochum implemented a trusted version of the GRUB [10] boot loader [5] in 2006. In their approach, they check the BIOS and the operating system kernel, but till now, they do not provide any recovery mechanism and as it is not the task of the boot loader to check the applications, they do not provide any application checks.

## 5 Conclusion

We propose a mechanism to ensure a secure usage of network nodes by validating the integrity of the participating computers and checking their authenticity. During the integrity checks, we validate the BIOS as well as the operating system kernel and the applications. All the checks are secured by the TPM, which provides a tamper resistant storage for the results. Furthermore, the TPM allows to block the NIC in case of a failure. This feature has never been proposed by any of the other approaches. Furthermore, we provide an automated recovery mechanism especially for the operating system kernel and the applications. The usage of the TPM allows us to use a very simple authentication mechanism like a challenge response protocol during the recovery process.

## References

1. Arbaugh W.A., Farber D.J., Smith J.M.: A Secure and Reliable Bootstrap Architecture, IEEE Security and Privacy Conference, Oakland, CA, pp. 65-71, 1997

2. Trusted Computing Group: Trusted Computing Platform Alliance. Main Specification Version 1.1b, 2003, available at <https://www.trustedcomputinggroup.org/specs/TPM>
3. Pearson S., Balacheff B., Chen L., Plaquin D., Proudler G.: Trusted Computing Platforms. TCPA Technology in Context, Prentice Hall, New Jersey, 2003
4. Arbaugh W.A., Keromytis A.D., Farber D.J., Smith J.M.: Automated Recovery in a Secure Bootstrap Process, Internet Society 1998 Symposium on Network and Distributed System Security, San Diego, CA, pp. 155-167, 1998
5. Selhorst M., Stueble C.: Trusted GRUB, University of Bochum, 2006, available at [http://www.prosec.rub.de/trusted/\\_grub.html](http://www.prosec.rub.de/trusted/_grub.html)
6. Yee B., Tygar J.: Dyad: A System for using physically Secure Co-Processors, Technical Report CMU-CS-91-140R, Carnegie Mellon University, 1991
7. Yee B.: Using Secure Co-Processors, PhD Thesis, Carnegie Mellon University, 1994
8. Lampson B., Abadi M., Burrows M.: Authentication in Distributed Systems: Theory and Practice, ACM Transactions on Computer Science, v10:265-310, 1992
9. Clark P.C., Hoffman L.J.: BITS: A Smartcard Protected Operating System, PhD Thesis, George Washington University, 1994
10. GNU GRUB, available at <http://www.gnu.org/software/grub/>