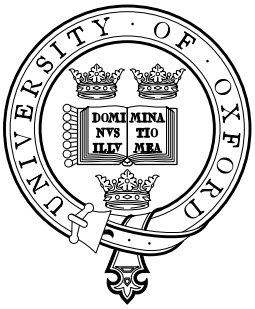
Master's Thesis

End-To-End Application Security Using Trusted Computing

Michiel Broekman

August 18, 2005

University of Oxford
Software Engineering Programme



University of Nijmegen
Security of Systems Group

Preface

This thesis is the result of my research project at the Software Engineering Programme of the Oxford University Computing Laboratory, from January till August 2005. This project has been done under the supervision of Dr. Andrew Martin from the University of Oxford and Dr. Jaap-Henk Hoepman from the University of Nijmegen. I would like to take the opportunity to thank Andrew Martin, Andrew Cooper, and Jaap-Henk for their help and support. Also, I would like to thank my family for supporting me throughout these years and for giving me the opportunity to fully develop myself.

Michiel Broekman

Abstract

This thesis describes the implementation of Trusted Computing in end-to-end application security. It focuses on the services that are provided by application layer protocols in end-to-end communication. There are many security issues related to application layer protocols and some of these are explored. Application layer firewalls are often used to protect against these security issues. Unfortunately they are application specific, which means that only recognised protocols can be examined. It is practically impossible to distinguish between all possible application layer protocols. Therefore an abstraction to application layer protocols is proposed by providing a classification that tries to capture a large subset of the full spectrum of protocols. The security issues involved are discussed and addressed by Trusted Computing. A more thorough investigation is performed by examining the architecture of some distributed application components. Spam and secure message store in SMTP are researched and discussed in the context of Trusted Computing. Access control in FTP is enhanced to enable policy enforcement of an object on both the server and client platform. Furthermore, some ideas are proposed to enhance the privacy and security of web cookies. The research offers a new perspective on how Trusted Computing can improve end-to-end application security.

Contents

List of Figures

Chapter 1

Introduction

This thesis describes the outcome of the research project that has been carried out at the Software Engineering Programme (SEP) of the Oxford University Computing Laboratory (OUCL). This chapter will first provide some introductory information on the problem definition. Then it will formulate the exact problem definition which will be the basis for the rest of this thesis. After the reader has gained a clear understanding of the motivations behind this project, chapter 2 will provide some information about Trusted Computing to show what trust mechanisms it can provide and how they are realized. Chapter 3 will give some background information on application layer protocols and present a classification of application layer protocols, which will make it easier to look at security issues from the perspective of end-to-end application security. Chapter 4 will discuss the security requirements of the various classes and the application of Trusted Computing to these security requirements. In chapter 5, Trusted Computing will be used in the implementation of the architecture of some distributed application components. At the end, chapter 6 will summarize the performed research and give an answer to the problem definition stated in the beginning. It will conclude by proposing topics for future research.

1.1 Application Layer Protocols and End-To-End Security

From the perspective of the Open Systems Interconnection (OSI) model application layer protocols are part of the top layer of the network stack. The OSI stack does not focus on how the various network protocols exactly accomplish their task. Instead, it describes different layers of which each layer performs services for the next higher layer, and makes request to the next lower layer. The OSI model is a hierarchical structure of layers which defines the requirements for communication between two computers. In this thesis the emphasis lies on the services that are provided by the application

layer protocols in end-to-end communication. This approach is completely in agreement with the objective of the OSI model.

There are many security issues related to application layer protocols. Various web servers, mail servers and other internet service software contain bugs that let remote users do things that are harmful. For example remote users can gain control of the machine and can do whatever they want. The exposure to these kind of threats can be minimized by running only the necessary software and getting the latest patches, and using software that has a good reputation [RCR04]. However, there remain many security issues that cannot be easily dealt with. Firewalls are often used to cope with these kind of issues.

Application layer firewalls are hosts running proxy servers that do not permit direct traffic between networks and perform logging and auditing of traffic that is going through them. These proxy applications are software components that run on the firewall and are application specific, which means that only recognized protocols can be examined. However, it is difficult to distinguish between all possible application layer protocols. Therefore a higher level approach must be chosen to cover all protocols without having to deal with the protocols one by one. This can be achieved by looking at how information is shared and disseminated in end-to-end communication between hosts. In the next section a justification is presented for this end-to-end approach.

1.2 End-To-End Argument

The end-to-end argument that was raised by Saltzer, Reed and Clark in 1984 [SRC84] has been one of the most influential in the area of communication protocol design. The paper discusses the placement of functions in a communication network. Certain functions can be placed in both the network and the end systems, like for example error control, security and routing. The end-to-end argument however, suggests that these functions should be implemented in the endpoints. The end-to-end argument states that:

"The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible."

As an example of the end-to-end argument the secure transmission of data is presented. Firstly, if the data transmission system encrypts and decrypts data, it has to be trusted to manage the required encryption keys in a se-

cure manner. Secondly, if the data is transferred to the target node and sent to the application, the data is not encrypted and vulnerable to attacks. Thirdly, the authenticity of the message has still to be checked by the application on the end system. However, if the application performs end-to-end encryption, then it can obtain the required authentication check and it can handle key management itself. Furthermore, the data is never exposed outside the application.

So in order to meet the requirements of the application, the communication subsystem does not have to provide automatic encryption of all data. However, automatic encryption of all the data by the communication subsystem may be used to ensure that a user or application does not deliberately transfer information that should not be exposed. However, neither mechanism can completely satisfy both requirements.

1.3 Trusted Computing Platforms

The ability to protect a computing platform by using only software has some inherent weaknesses [Pea03]. Security solutions based on software depend on the correct installation and execution, which can be easily affected by other software that has been executed on the same system. Even the most robust software does not have complete control over its own integrity. Malicious software can bypass the mechanisms of the OS and corrupt the behavior of the OS.

Experts in information security say that some security issues cannot be solved by software alone and therefore trusted hardware is required. For example, the increasing e-business activities on the internet demand more security than is given at this moment. This has led to the Trusted Computing Group (TCG, which was formerly called the Trusted Computing Platform Alliance) that designs the specifications for computing platforms that are supposed to create trust for software processes, based on some extra hardware within the platform.

Basically a Trusted Platform is a computing platform that makes use of a trusted component. This trusted component is in the form of built-in hardware and is the basis of trust for software processes. The security functions of the security hardware in a Trusted Platform must be trusted. The hardware is a root of trust and is able to measure both the hardware and the software environment of a specific system. If the software is decided to be trustworthy for some purpose, then all other security functions and software can operate as normal processes. These roots of trust are the core TCG capabilities.

There are all kind of interesting applications that can be realized with TCG technology. One interesting application is that of distributed firewalls. On a Trusted Platform a distributed firewall is significantly more powerful. Another application is rate limiting for the prevention of distributed denial of service attacks (DDoS). There are a few papers that describe possible OS architectures which support Trusted Computing. One of these architectures, called Terra, will be briefly discussed in the next chapter.

1.4 Research

The introductory information given in this chapter is meant to familiarize the reader with some basic concepts that will be further investigated and used in the following chapters. In this thesis the emphasis lies on the end-to-end communication between applications. The general problem definition can be formulated as follows:

What are the benefits and drawbacks of using Trusted Computing in the implementation of end-to-end application security?

From the information provided above, it can be concluded that Trusted Computing may significantly improve platform security and guarantee specific properties of applications. Therefore it is very interesting to research the possibilities of enhancing end-to-end application security using Trusted Computing.

Chapter 2

Trusted Platforms and TCG Technology

This chapter provides the reader with relevant information about Trusted Platforms and TCG technology in order to create a basis of understanding for the rest of the thesis. The first section is intended to give the reader a summary of Trusted Platforms and their context. In the second section some applications of Trusted Platforms (TPs) are brought together. The last section describes a few relevant properties of Trusted Platform technology in more detail. These properties are used further on in this research.

Much of the information given in this chapter is gained from the book "Trusted Computing Platforms, TCPA Technology in Context" [Pea03]. This book presents an overview of the 1.0 specification which is a bit out of date, as the 1.2 specification was already released in February 2005. However, the general ideas of Trusted Computing Platforms are still the same and version 1.0 of the TCG specifications provides the basic building blocks to design secure applications and services. For more comprehensive and up-to-date information about Trusted Computing Platforms the reader is referred to the official web site of the Trusted Computing Group (TCG)[1].

2.1 An Overview of Trusted Platform Technology

This section introduces the reader to the concept of Trusted Platform technology. It describes the reasons for Trusted Platforms and what they can provide. Trust is an important notion in the development of Trusted Platforms and therefore a short discussion about its definition is given. After that, a summarization of the main features of Trusted Platforms is presented.

[1] Web site can be found at http://www.trustedcomputinggroup.org

2.1.1 The Reasons for Trusted Platforms

A platform is a computing device like a PC, server or mobile phone that is capable of computing and communicating electronically with other platforms. Nowadays computer platforms can be found everywhere and they are used for electronic business and commerce. This means that information protection is becoming increasingly important, especially on client platforms. Secure operating systems have been developed for server platforms but no corresponding improvement has been realized for client platforms. The reason for this is the ad hoc manner in which client platforms are developed, the large number of them and the cost.

The flexibility and openness of PC platforms have created a remarkable business growth and efforts to counter this flexibility and openness have proven to be unsuccessful. People have always preferred convenience to security and this makes confidence in client platforms (PCs in particular) even more difficult.

There is no single company responsible for the design of the architecture of all platforms. Most client platforms on the internet are PC-based and the variety of hardware and software continues to increase. As businesses increasingly rely on PCs and the internet in order to meet their objectives, the trustworthiness of platforms and PCs becomes an increasingly important issue. Furthermore, the same computer platforms are used for both personal use and business which means that users must be able to store and use sensitive data on their platforms.

Unfortunately a computing platform cannot be protected by using software alone because of its inherent weaknesses. The amount of confidence a user can get of the software largely depends on the correct installation and execution of it, which can be influenced by other software that is running on the same platform. Malicious software can bypass the security mechanisms of an OS and corrupt its behavior without the OS noticing anything. In general it is not really difficult to find a particular change made to the software if it is clear what to look for. However, conventional computing platforms do not enable a remote user to check whether the platform is appropriate to store and process sensitive information. It is certainly possible to identify a remote user accessing a corporate network by using a Virtual Private Network (VPN) gateway, however nothing can be said about the trustworthiness of the remote machine.

Some security issues cannot be solved by software alone and therefore trusted hardware is required as a basis for security mechanisms.

2.1.2 Limitations of Existing Security Technology

In the previous section it was stated that software alone is not enough to protect a computing platform because of its inherent weaknesses. Although this may be true, a few other important and more specific limitations of existing security software should be discussed as well. Existing security infrastructure includes the following technologies:

- **Firewalls** provide boundary protection for computer networks, but when much traffic is going through them they can become a bottleneck [RCR04]. Moreover, to add new functionalities and services it has become normal to increase the number of 'holes' in firewalls through which dynamic content and programs can be pushed. So there is the choice to either prohibit the new traffic or to adapt the firewall to deal with the new situation.

- **Software security programs** are available to provide a spectrum of security functions. These programs may run in a cryptographic co-processor or on the main platform processor. Software that runs on the main platform processor assumes that it runs in a safe environment. So confidence can be gained only if the software is installed and executed properly. However, data that is stored can be exposed to malicious programs and is therefore not safe.

- **Cryptographic accelerators/co-processors** are other types of security products. They are very good at creating a trusted environment. They may even be preferred over a Trusted Platform, because the co-processor can do bulk encryption in a physically protected environment. However, the disadvantage is that this kind of specialized hardware is too expensive and therefore cannot be used in all platforms. Therefore it must be clear that ubiquitous platform security cannot be based on conventional cryptographic co-processors.

- **Other specific technologies** can be used to increase the levels of confidence in computing platforms. A large number of security protocols (like IPsec) can be implemented in either hardware or software. All these techniques are executed on the main CPU and therefore rely upon the correct operations of the host computing platform.

This does certainly not mean that Trusted Computing technology will replace existing security technology. Both technologies can be combined to realize new applications and to provide a higher level of security. Examples of applications that use both existing security technology and Trusted Platform technology are described in for example [GRB03], where Trusted Computing technology is combined with SSL. A recently published book [Smi05] treats various application case studies in which existing security technology

is merged with Trusted Computing technology. More information about this will be presented later on in this thesis.

2.1.3 Definition of Trust

First if all it should be noticed that trust is a complex notion. With respect to trust one usually thinks in terms of entity A trusting entity B for something. However due to a number of reasons it is a more complex notion than expected. For example, trust is dynamic which means there exist differing phases in a relationship such as building trust, ongoing trust and declining trust. Further, trust levels vary in the degree and scope of trust: entities trust (or do not trust) each other to fulfill specific tasks, rather than for everything. At the same time, trust in certain areas can transfer to trust more generally.

Trust can be categorized in terms of behavioral and social components. These two different approaches help to get a better understanding of how Trusted Platforms can improve trust.

- The behavioral definition of trust is about collecting evidence of behavior and providing evidence of behavior. The obtained information is used to decide whether a platform can be trusted.
- The social definition of trust focuses on what it is to be trustable. Social trust in a Trusted Platform is an expression of confidence in behavioral trust, because it is an assurance about the implementation and operation of the specific Trusted Platform.

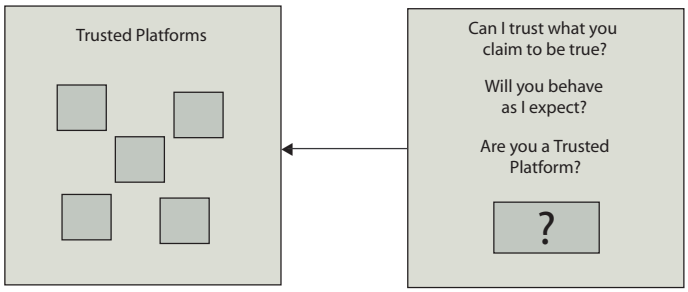


Figure 2.1: Questions addressed by TPs.

Trusted Platforms use social trust to get confidence in the mechanisms that collect and provide evidence of behavior. They also use social trust to provide confidence that certain values of evidence represent a platform that is in a “good” state. It should be noted that a platform itself cannot decide

whether it can be trusted because trust is dependant on how the platform is to be used. So ultimately only a user can decide if the platform should be trusted for the intended purpose. The platform must report the necessary information to the user so that a decision can be made. Figure 2.1 shows what kind of questions play a central role in obtaining trust.

2.1.4 Trusted Computing Group and the TCG Specification

The security issues described above are addressed by the Trusted Computing Group (TCG), formerly known as the Trusted Computing Platform Alliance (TCPA). This group is responsible for the design of the specifications for computing platforms that can provide a basis of trust to software processes, based on some hardware within the platform. The TCG specification is independent of the type of platform. More information about the Trusted Computing Group can be found on their official web site.

2.1.5 Definition of a Trusted Platform

A Trusted Platform is a computing platform with a trusted component. Creating a Trusted Platform out of a conventional platform requires TCG *roots of trust* to be embedded in the platform, so that local and remote users can trust the platform. These TCG *roots of trust* are realized by cost-effective security hardware. The security functions that are enabled by this hardware must be trusted as the hardware is the root of trust in the process that measures the platform's hardware[2] and software environment. The focus will be on the software environment because knowing what the computing engine is doing is of most importance. If the software environment turns out to be trustworthy then all other security functions and software can operate normally. These *roots of trust* are core TCG capabilities.

As can be observed in the figure 2.2, within each Trusted Platform there is a Trusted (Platform) Subsystem which contains a Trusted Platform Module (TPM), a Core Root of Trust for Measurement (CRTM), and support software (the Trusted platform Support Service, TSS). The TPM is a hardware chip that is separate from the CPU(s). During the boot process the CRTM is the first piece of software that is allowed to run. The TSS is responsible for all kind of functions that are necessary for communication with the rest of the platform or other platforms. So called Certification Authorities (CAs)

[2] [HvD04] claims that TCG technology does not measure the hardware environment, whereas [Pea03] claims that it can do both. This contradiction is the result of the fact that at this stage the CRTM is part of the BIOS. However, the BIOS can be altered so that the CRTM cannot be trusted anymore. Therefore future BIOS should have both a writable and a read-only part. Then the CRTM can remain unaltered in the read-only part of the BIOS.

are used to testify that the TP is genuine and are therefore very important in gaining trust.

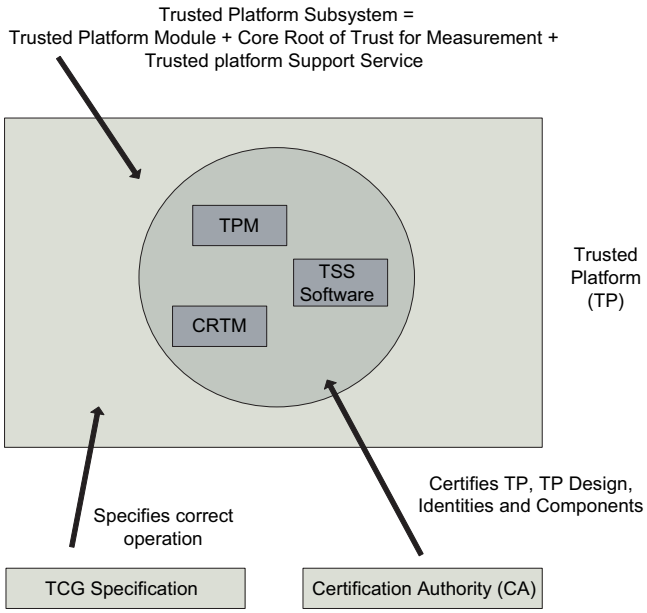


Figure 2.2: Trusted Computing Platform model.

In information security the Trusted Computing Base (TCB) is the set of functions that is responsible for the security properties of a platform. With respect to a Trusted Platform the TCB is the combination of a Trusted Subsystem (mainly dealing with secrets) and some additional functions that mainly deal with the use of those secrets, like bulk encryption. The Trusted Subsystem first shows that it can be trusted and then it shows that the rest of the TCB in the Trusted Platform can be trusted as well. Certifications from trusted entities are used to vouch for the platform in all kind of configurations.

Now that the Trusted Platform model has been described, some specific trust mechanisms of Trusted Platforms can be introduced. In the following subsections the characteristics of the most important ones are summarized.

2.1.6 Cryptographic Capabilities

Cryptography plays an important role in the realization of a Trusted Platform. Therefore the cryptographic capabilities that are provided by the TPM are listed below.

- Hashing (SHA-1) can be used on both data and secret keys and pro-

duces a fixed-size output. The use of SHA-1 is explained later on in the thesis.

- Random number generation (RNG) is used for the security of many cryptographic mechanisms. Examples are the primes in the RSA encryption and digital signature schemes and the *nonce* (the concept of a *nonce* will be explained later on in the thesis). In all these cases the number that is generated must be of sufficient size and be random in the sense that an adversary cannot construct the generated number.

- Asymmetric key generation (RSA) is used to create asymmetric key pairs for asymmetric encryption/decryption.

- Asymmetric encryption/decryption (RSA) is used to wrap and unwrap secrets because this gives better functionality than would be provided by symmetric encryption/decryption. By using an asymmetric algorithm, public key operations can be performed outside the TPM on the CPU or at another platform, while the private keys are left inside the TPM. If a symmetric algorithm would be used then both encryption and decryption must be performed in the TPM because the single symmetric key has to remain secret.

Besides asymmetric encryption/decryption the TCG specification also requires symmetric encryption/decryption (3DES) for purposes that are not further described here. At the moment AES, 3DES's replacement, is not part of the TCG specifications, however in the future it may be required as well. For more information the reader is referred to [Pea03].

2.1.7 Digital Signatures

On a Trusted Platform digital signatures are created by encrypting data with the private key that is protected by the TPM. Digital signatures will become more important as their legal status is increasing. Trusted Platforms can support and enhance the use of digital signatures. The benefits that can be achieved are listed below.

- A Trusted Platform can protect signature keys by using the TPM. It never releases these keys outside the TPM. The Trusted Platform uses these keys to digitally sign data that is submitted to the TPM.

- A Trusted Platform can improve the value of digital signatures by adding integrity metrics that show the software state of the platform when the data is signed.

- Depending on the implementation of the TPM, a Trusted Platform can further improve the value of signatures to guarantee that what is signed matches with what was seen by the signer.

2.1.8 Integrity Measurement and Report

A Trusted Platform executes a series of measurements that record the summaries of software that has executed or is executing on the platform. This is illustrated in the figure 2.3. There is a boot-strapping process where Trusted Subsystem components measure each next component in the chain and save the value in the TPM. This procedure enables the measurement and record of a set of software instructions (binary code) before it is executed. This means that malicious software is recorded and therefore cannot hide. Cryptographic techniques are used to send the measurement to an interested party without enabling a malicious party to change it in transit (in most literature this principle is known as *remote attestation*).

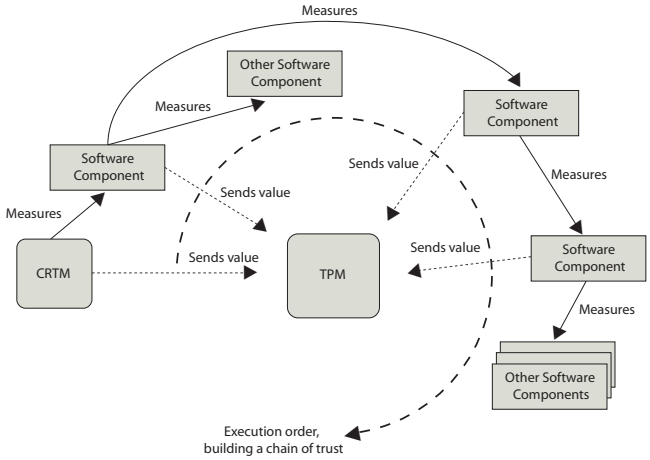


Figure 2.3: The measurement process in a TP.

2.1.9 Creation of Trusted Identities

To prove that the measurements are reliable it is necessary to prove that a platform is a genuine Trusted Platform. This proof is realized by cryptographic attestation identities. Each identity is created on the individual Trusted Platform, with attestation from a PKI Certification Authority. A randomly generated asymmetric key and an arbitrary string that is used as an identifier for the pseudonym (which can be chosen by the owner of the platform) are part of each identity. In order to get an attestation from a CA, the platform's owner sends the CA information that shows that the identity was created by a genuine Trusted Platform. This attestation process makes use of signed certificates from the manufacturer of the platform and uses a secret that is installed in the TPM. This secret is only known to the TPM

and is not released to other parties. Cryptographic attestation identities are used so that these secrets do not have to be made public to arbitrary third parties.

2.1.10 Privacy

Attestation identities provide a prove that they correspond to a Trusted Platform. A specific identity always identifies the same platform. The Certification Authority (CA) is the only party that can track the origin of the specific identity, which assures privacy. So an appropriate selection of CAs enables the owner to control traceability from an attestation's identity to the certificates that attest to a specific TPM and a specific platform. It is important to mention that identities can only be correlated with other identities by the CA that certifies these identities and that the owner has the choice of that CA. The owner is able to choose a CA that has the policy not to correlate identities or to correlate the identities. Different purposes can get different identities and separate identities are given to different users of the Trusted Platform. However, platform privacy can still be at stake because of identification of platforms from for example MAC and IP addresses.

In [Arb02] it is claimed that Trusted Computing does not provide a reasonable degree of privacy. According to this paper there are two major problems with the approach described above: (1) if a user requests several anonymous identities then the trusted third party can link all of the anonymous credentials to the user because they have complete knowledge of the user's identity and (2) if the trusted third party ever conspires with a true name CA then it becomes easy to relate the user's real identity with their anonymous identities by matching public keys.

This claim should not be taken too seriously because there is no reason to talk about Trusted Platforms if CAs cannot be trusted anymore. A CA is an organization that vouches for an entity and verifies if a TP is genuine. Therefore CAs are centrally involved in the manufacture and usage of TPs. Furthermore the owner is able to choose a CA that does not correlate identities. Nonetheless, there is definitely some truth in this claim because preventing potential conspiracies between different parties is a serious issue that has to be taken care of.

2.1.11 Protected Storage

A TPM can be seen as a secure portal to protected storage, however the time to store and retrieve specific information can become large. The TPM is intended for keys that encrypt files and messages, keys that sign data and for authorization secrets. It is possible for a CPU to get a symmetric

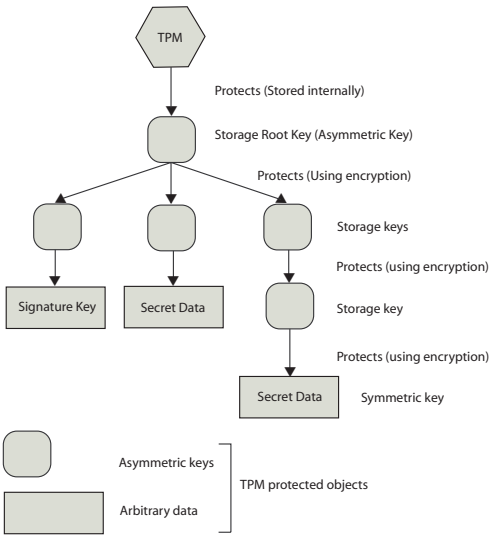


Figure 2.4: A storage hierarchy.

key from a TPM for bulk encryption or a CPU can send data to a TPM and ask the TPM to sign that data. The TPM acts as a series of separate operations on individual secrets. These operations together form a tree (hierarchy) of TPM protected objects, each of which contains a secret that is encrypted (wrapped) by the key above it in the hierarchy. It is important to mention that the TPM itself does not know anything about this hierarchy and only receives a series of commands from untrusted software that manages the hierarchy. An example of a possible hierarchy is shown in figure 2.4.

An essential characteristic of Trusted Platforms is that an object that is protected by a TPM can be sealed to a particular software state in a platform. If the object is created the creator specifies the software state that has to exist when the secret is to be revealed. If a TPM decrypts the TPM protected object, the TPM first checks if the current software state equals the stated software state. If they match, the TPM grants access to the secret and if they do not match, the TPM denies access to the secret.

2.2 Applications of TCG Technology

Trusted Platform technology can provide the basic features that are required for current applications. The protection of data and digital signing can be achieved by using secrets that are protected by the TPM. In this section some basic enhancements that are enabled by Trusted Computing are given first. Then a brief description of Virtual Machine-Based Platforms for Trusted

Computing is presented. After that a few possible applications are shown where Trusted Platforms promise to have significant impact.

2.2.1 Improvements Realized by Trusted Platforms

By providing improved methods of storing secrets and signing data Trusted Platforms can attest that a specific secret is being used on a TP and they can report the state of a TP. The enhancements that are enabled by TPs are described below.

- **Improved Protection of Secrets** TCG technology provides a way of encrypting that enables the caller to define the software environment that must be fulfilled in order to get the TPM decrypt a secret. A decrypting software environment is described by defining values of some or all so-called Platform Configuration Registers (PCRs) which are saved with the secret. So when a TPM tries to retrieve the secret, it first compares the saved PCR values with the current values of those PCRs. Only if the stored and current PCR values match then the TPM reveals the secret (in the case of data) or it uses the secret (in the case of a key). This TCG feature can be used quite easily, however the application should know how to use it.

- **Improved Signatures** TCG technology improves digital signatures by (1) using keys that can prove they belong to a genuine TPM and (2) incorporating the current value of some or all PCRs into the value of the signature. Only specific data structures generated by the TPM are signed directly by a TP identity key. Some structures that originate outside the TPM, but are constructed by the TPM and thus cannot be tampered with, are included as well. Examples of these kind of structures are messages concerned with identity creation, signatures that include PCR values and signatures that vouch for other keys. A TP does not directly use its identity keys to sign homogenous data. In fact, an identity key vouches for another TP key which then signs the homogenous data. This mechanism is important as it prevents a malicious party from signing data with the identity key. Otherwise it would be impossible to distinguish between data generated by the TPM and data that is generated outside the TPM.

- **Improved Protection of Data on Remote Computers** The aforementioned improvements increase the confidence for the owner of data that resides on a remote platform. So it is possible to store data on a remote platform and to define the conditions under which the data can be used.

2.2.2 Examples of Improved Conventional Services

Conventional services can be improved by using the storage and signature mechanisms that TPs provide. A few scenarios are described below.

- **Checking Client Integrity** A server can ask client TPs for integrity information and then use this information to check that unauthorized modification has not been made to the client platforms. Figure 2.5 visualizes this process.

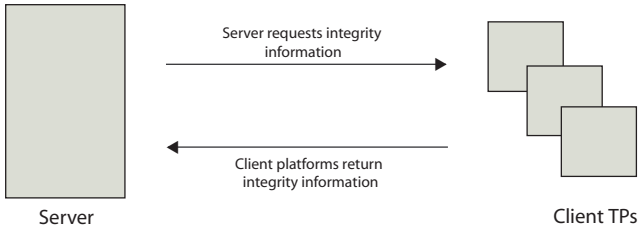


Figure 2.5: Integrity-checking.

- **Authorized Network Connection** A corporate server allows services to be provided only to remote platforms that are TPs and that can provide trustworthy information about the software that is installed on the platform. This makes it impossible that sensitive information is sent to computers that cannot be trusted.

- **Remote Attestation** Trusted Platforms can be used in a Virtual Private Network (VPN), so employees that are not at their home office can check if an unknown platform is trustworthy before using it. So employees can use an unknown TP at an arbitrary physical location to work on company information. The employees insert a smart card into the unknown platform. The smart card then asks the employer's server to challenge the unknown platform and to decide whether it can be trustworthy. The server challenges the unknown platform after which the unknown platform responds by digitally signing and sending its PCRs and other integrity information. The set of data that is sent should be enough to show that a TP will not reveal sensitive information to unapproved platforms. The server then checks the identity signature and determines whether the integrity metrics are acceptable. If the server decides that everything is well then the server gives the smart card the TPM identity and PCR values of the approved platform. So at the end the server sends sensitive information to the approved platform and the approved platform will not use the information in the wrong way.

For a comprehensive list of enhanced conventional services the reader is referred to [Pea03].

2.2.3 Virtual Machine-Based Platforms for Trusted Computing

It is claimed that the vision of Trusted Platforms cannot be achieved with today's operating systems because they offer poor assurance and implement a security model that contradicts the one that is required for trusted computing [GRB03]. Therefore [GRB03] and [GPC+03] outline the design of implementing a new OS architecture based on the idea of a *Trusted Virtual Machine Monitor* (TVMM). In this model traditional applications and OSes can run side-by-side on the same platform in either an *open box* or *closed box* execution model.

A Trusted Virtual Machine Monitor is a high-assurance virtual machine monitor that splits up a single tamper-resistant, general-purpose platform into multiple isolated virtual machines. Existing applications can run in a standard virtual machine (open-box VM) which is quite similar to today's open platforms. Applications can run in closed-box virtual machines as well so that the functionality of a dedicated closed platform is provided. The TVMM is responsible for protecting the privacy and integrity of the content of a closed-box VM. At the same time the TVMM can cryptographically authenticate the running software to remote parties by attestation.

Closed-box VMs are completely isolated from the rest of the platform. Because of hardware memory protection and cryptographic protection of storage, the content is protected from observing and tampering by the platform owner or some malicious party.

A good example of an OS architecture that enables these kind of features is *Terra* [GPC+03]. In figure 2.6 the TVMM isolates and protects the independent VMs. The two closed-box VMs that are shown in dark gray are protected from eavesdropping and modification by anyone but the remote party that supplied the box. The SETI@Home client is put in a closed-box VM so that its server can verify that it has not been modified and claims that it has run checks that it actually has not. The online game is in a closed-box as well in order to deter cheating. The TVMM can identify the contents of the closed box to remote parties so that they can gain trust in the platform. In each VM another OS can be run. The management VM is responsible for the configuration of how much storage and memory is assigned to the VMs.

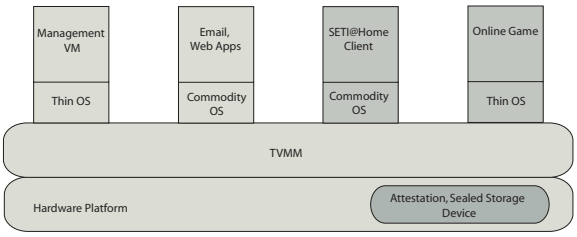


Figure 2.6: Terra's architecture.

2.2.4 Example Applications

There are several areas in which Trusted Platforms can have a significant impact. The use of Trusted Platforms can have a big influence on the functionality of today's client side technology. There will be no focus on applications that are related to Digital Rights Management (DRM) of which the public policy consequences have been discussed quite thoroughly [And03]. Some interesting examples are presented in [GRB03] and are briefly described below.

- **Regulated Endpoints and Distributed Firewalls** Conventional firewalls assume that everyone on the inside of the network is trusted and that everyone on the outside is untrusted. However, due to the deployment of wireless access points, tunnels, VPNs and dial-ins the distinction between the inside and the outside becomes more vague. Because of increasingly dynamic network topologies, distributed firewalls simplify the task of implementing network security policies a lot. By using a distributed firewall the security policy is defined centrally and enforced at every endpoint of the network. This enables a larger variety of policies and greater scalability than the conventional centralized firewalls [IKBS00].

  On a normal host, distributed firewalls can help protecting a host from others, but they are of little use for protecting others from the host: it is impossible to tell if the host does not tamper with or bypass the firewall. However, on a Trusted Platform a distributed firewall is a lot more powerful because it can prevent packets that violate the central security policy from ever reaching the network. A distributed firewall can for example prevent port scanning and IP spoofing as the packets will never reach the network. At the same time the distributed firewall can make sure that all VMs on the machine have implemented the connection rate limits properly. This means that distributed firewalls on Trusted Platforms enable well-regulated endpoints for a all kind of network types.

- **Third-Party Computing** It is increasingly common to borrow, lease or donate computing resources. A nice example of this is the use of donated cycles for massively parallel scientific and mathematical computations by SETI@home. Another example, which is quite popular at this moment in the United Kingdom, is the emerging field of grid computing which allows heavy users of scientific computing resources to pool and share their computing resources. The problem with this third-party computing is trusting the machines that are doing the computation to (1) produce the correct results and (2) keep the contents of the computation secret. Trusted Platforms offer a solution to these two problems by providing attestation. Using attestation, remote machines can show and prove that they are running the right executable image. The results of the computation can be signed and encrypted so that privacy and authenticity is guaranteed.

- **Civil Liberties Protection** Law enforcement increasingly needs the use of network surveillance devices that can potentially violate civil liberties. These kind of devices are certified not to exceed their legal boundaries by getting together a select group of experts to review their design. Unfortunately, there is no certainty that the system that is reviewed by the experts is the one used in the field. Attestation is the technique that makes it possible to verify if a system can be trusted.

2.3 Key Components of Trusted Computing

This section focuses on some of the key components that are part of Trusted Computing technology. They will be discussed in more detail as they will be used later on to address the security issues of some application layer protocols or to enhance specific security mechanisms. Each of the following subsections scrutinizes a specific component. First, the concept of *platform configuration registers* is explained, because they are a prerequisite for the functioning of the other key components.

2.3.1 Platform Configuration Registers

An important part of a Trusted Platform are the *platform configuration registers* (PCRs). They are used in all kind of trust mechanisms. The integrity metrics are stored in such a manner that misrepresentation of the presented values or the sequence in which they were presented is prevented. The PCRs are used to store summaries of integrity metrics. If values of integrity metrics would be individually stored and updates of integrity metrics would have to be individually stored as well, then it is difficult to define an upper bound on the size of memory that is necessary to store them. There

is an unknown number of integrity metrics that have to be measured in a platform and an integrity measure can change at every moment so that a new value has to be stored. The authentication of the source of measurements of integrity metrics is practically impossible, and adding a new value of an integrity metric should not overwrite an existing value. If this would be possible then a malicious party can easily erase an existing value that represents a subversion and replace it with a value that looks harmless.

The solution that is presented in the TCG specification is based on storing sequences of integrity metrics instead of individual integrity metrics. So values of integrity metrics are appended to a sequence of which the representation has a fixed size. At power-up the states of all the sequences in the TPM are set to a specific value. Then each new metric must change the value of the sequence. The value of a new integrity metric is concatenated with the existing value of the sequence. The digest of the concatenation will then be the new representation of the sequence. The digests are saved in the Platform Configuration Registers inside the TPM.

2.3.2 Integrity Recording and Reporting

Integrity measurement is one of the most important components of a Trusted Platform. The host platform uses integrity measurements to provide protected storage: the disclose of secrets is only performed if the platform is in the correct state. Third parties use integrity measurements to verify if the target platform is in the correct state. A third party believes the measurements because the platform signs the measurements with the TPM identity which provides guarantee that the measurements are coming from a Trusted Platform.

Integrity challenge and *integrity response* are the TCG processes that are responsible for reporting the current hardware and software configuration of a computing platform to local and remote challengers. *Integrity verification* is the process of verifying that the configuration is the one that is desired. So if some computing platform wants to know the state of the computing environment inside a TP then there is a strong dependance on the values of the integrity metrics. Therefore the integrity metrics should be reported by a trusted mechanism which in this case is the TPM. The TPM creates a certain amount of trustworthiness by signing data using one of its identity keys.

A challenger, who wants to know the state of a target platform, creates a nonce and sends an *integrity challenge* to the target platform. The TP does not have to respond to the challenger, which depends on the policy of the TP. If the TP decides to respond, the Trusted Platform Agent coordinates the integrity response. The TPA sends the nonce to the TPM so that the

TPM signs the nonce and the current PCR values, using a TPM identity. After that, the TPA gets the logs of the measured software from the Trusted Platform Measurement Store and gets certificates from the relevant repositories. The TPA collects all this information and sends it to the challenger. This is the *integrity response*.

The challenger must verify the integrity response in order to decide if the target platform can be trusted. In practice, the challenger must investigate the following aspects:

- The certificate that is signed by the Privacy CA that attests to the TPM identity (to know whether the TPM identity is a genuine TPM identity).
- The nonce signed by the TPM identity.
- The PCRs signed by the TPM identity.
- The PCR values and the certificate signed by the entities that vouch for the platform (to determine whether the platform is in the expected state).

The challenger then makes a decision whether there is enough reason to trust the target platform and to interact with it. In most cases the challenger only wants to know if the target has the correct identity, if it is running the right OS with the latest security updates and if it has a virus checker that is using the latest virus definitions. If the challenger wants to communicate with the target platform then the platform should sign all information that it sends to the challenger for the rest of the session. If the data would not have been signed there is a chance of a man-in-the-middle attack. If a specific signing command, which includes PCR values, is issued then a TPM identity signs the information. Arbitrary data is signed by an ordinary signing key that is certified by a TPM identity, because TCG does not allow TPM identities to sign arbitrary data as there is a risk of forgery of special data structures that are generated by the TPM and signed by TPM identity keys. It is important to notice that there is a time interval between the beginning and the end of a trusted transaction. The TCG specification tries to overcome this problem by asking the integrity metrics both before and after the transaction, although this approach clearly does not really offer a satisfactory solution.

Fine-Grained Remote Attestation In [SPvD05] an idea is proposed to address two important problems that are inherent to the attestation technology described above:

- Due to the great diversity of software versions and configurations the verification of the hash is quite difficult.
- The time-of-use and time-of-attestation discrepancy is still to be addressed, because the code may be correct at the time of attestation, but it may be compromised by the time of use.

BIND, a fine-grained attestation service for secure distributed systems, promises to deal with these issues. It offers the following properties to realize its aims:

- BIND accomplishes fine-grained attestation, by only attesting to the part of the memory that is relevant. So instead of attesting to the entire memory content. This makes verification a lot easier.
- BIND makes the gap between time-of-attestation and time-of-use a lot smaller. It measures a piece of code directly before it is executed and uses a sand-boxing mechanism to protect the execution of the attested code.
- BIND ties the code attestation with the data that the code produces, so that it can figure out what code has been run to generate that data. Further, the verification of input data integrity is incorporated into the attestation which enables a transitive integrity verification. So by using one signature it is possible to vouch for the entire chain of processes that performed transformations over a piece of data.

Semantic Remote Attestation In [HCF04] remote attestation is scrutinized and a another approach is chosen. The kind of attestation that is proposed is called *semantic remote attestation*. It is claimed that the kind of remote attestation, which is described in the TCG specifications, is static, inexpressive and fundamentally incompatible with today's heterogeneous distributed computing environments. Current techniques of remote attestation do not really investigate program behavior as they only attest to a specific binary. However, it is possible that an attested binary has bugs and does not confirm to the security policy the server was expecting. At the same time it is quite difficult to keep pace with the continuing release of software patches and upgrades. Therefore a language-based virtual machine is proposed to enable remote attestation of complex, dynamic and high-level properties in a platform-independent way. This principle is known as semantic remote attestation.

Semantic remote attestation uses a Java Virtual Machine to create a language-based virtual machine (for the sake of clarity, a VM is not the same as a VMM which was described earlier in this chapter) that executes platform-independent code. The requests for remote attestation are now made to

the TrustedVM which enforces or checks the specific security policies on the code that is run by the TrustedVM. See also figure 2.7 for an overview of the top-level architecture.

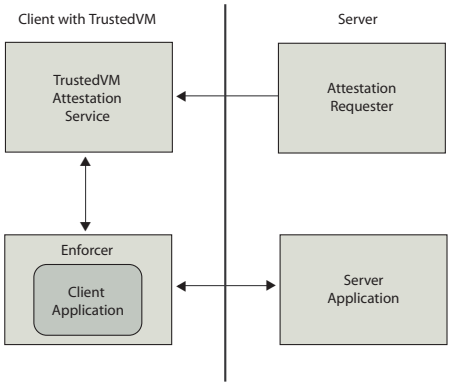


Figure 2.7: Architecture for dynamic checking in a TrustedVM.

Secure Boot Integrity measurements can be used in *secure boot* which is about checking that a boot process is proceeding as expected. A *secure boot* process checks if the values of the PCRs match with the expected values that are stored in the data integrity register (DIR) and activates an exception-handling routine if the values do not match. So a platform boots in a predefined way or not at all. However, [HvD04] states that secure bootstrap is not enough and that secure boot standards should verify the firmware of all devices in the computer as well, so not just only devices that are accessible by the host CPU. Therefore a simple extension to secure bootstrap is proposed which prevents attacks on firmware by adding some extra functionality to the devices so that they can prove their trustworthiness. Figure 2.8 shows that hashes of bootstrap code, operating system and applications are stored in the PCRs which can be queried. For a more detailed description about PCRs and integrity measurement the reader is referred to [Pea03].

2.3.3 Protected Storage

Protected storage is a service to the platform in which the TPM is used as a portal to confidential data that can be stored on all kind of unprotected storage media. The service creates and uses TPM protected objects that exist outside the TPM. The objects can be either data protected by the TPM (TPM protected data objects) or TPM keys protected by the TPM (TPM protected key objects). The protected storage service needs help from the host platform software to manage those objects. The management

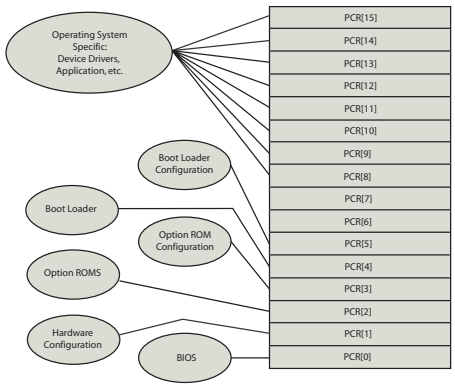


Figure 2.8: Hashes of bootstrap code stored in PCRs.

process itself is not security sensitive and therefore the TCG specification does not include this. The protected storage service can work without using any integrity measurements, but complete functionality requires access to the data that is stored in the PCRs. The values of these PCRs have to be known in order to decide if the platform is in the right state and if access should be granted to the secrets. Important points of interest for users are:

- Protected storage makes it possible to store signature keys in such a way that the TPM can use them without exposing them to the host platform.
- Bulk encryption keys or arbitrary authorization data can be stored in such a way that cooperation of the TPM is needed to reveal them to the host platform.
- Protected data can be stored in such a way that either permits duplication of the data by TCG features (which is under control of the owner of the data) or prevents duplication of the data by TCG features.
- Protected data can be stored in such a way that prevents use of the data if the platform is not in an expected state (this functionality requires access to the data stored in the PCRs).

Cryptographic Protection The TPM uses cryptographic functions to make sure that the TPM protected objects that are created by protected storage have the necessary properties of confidentiality and integrity. The focus here will not be on the cryptographic capabilities, for further information on this topic the reader is referred to [Pea03].

To provide confidentiality of TPM objects asymmetric cryptography is used instead of symmetric cryptography. One important reason for this is the fact

that some plain text originates outside the TPM and asymmetric cryptography permits encryption of that data outside the TPM without revealing the decryption secret. This can improve overall platform throughput as the TPM becomes less of a bottleneck. At the same time it permits encryption to be done at another platform. Unfortunately the drawback of using asymmetric cryptography, compared to symmetric cryptography, is that it takes a longer time to generate keys and perform encryption/decryption. Another drawback is that the size of data in the TPM protected object is limited by the size of the asymmetric key that is used to encrypt it. So the amount of data that can encrypted by the TPM and stored as a TPM protected object is limited. The Trusted Computing Group came with the idea to make all Protected Storage structures fit in a single block.

The integrity of data objects should be checked as well. Usually integrity checks are performed by checking the self-consistency of data. Protected storage already has a built-in consistency check for decrypted data in the form of the authorization check. A TPM protected object includes the encrypted authorization data so that a comparison can be made with the authorization data that is sent to the TPM to prove sufficient rights to use the TPM object. The probability that there is an authorization match after decryption of a corrupt TPM protected object is very small, because authorization data is the size of an SHA-1 digest. So no other explicit integrity check is required.

However, an explicit integrity check is certainly needed for every associated plain text data, because all TPM protected objects correspond with plain text data. At the same time all TPM protected key objects are associated with the public part of that key and other descriptive information, whereas all TPM protected data objects are associated with descriptive data. In order to enable an integrity test of the plain text data that corresponds with the TPM protected object, every TPM protected object contains a digest of the associated plain text data. So if the authorization for the TPM protected object is successful then the plain text integrity digest must be correct and can then be used to check the integrity of the plain text data.

Migratable and Non-Migratable Migratable TPM protected objects can be infinitely replicated by its owner and can be moved to another platform. Non-migratable TPM objects however, are locked to one particular TPM and never duplicated. As a result, migratable TPM key objects are allowed to be created outside the TPM and are only protected by the TPM, but they can also be created by the TPM itself. Non-migratable keys always have to be created by the TPM. More detailed information can be found in [Pea03].

Object hierarchy A TPM protected object that is stored outside a TPM will not be lost or corrupted. A TPM protected object includes all kinds of secret attributes of the key or data it is protecting, and it is associated to plain text attributes that are not secret. All TPM objects are encrypted by an encryption key. Such an encryption key is stored outside the TPM as another TPM protected key object. The consequence of this is that a set of TPM protected objects form a tree hierarchy, where each child TPM protected object is encrypted by the encryption key in the parent TPM protected object. The only encryption key that is stored in the TPM is the storage root key (SRK). It is generated inside the TPM and is non-migratable. The intermediate TPM object nodes in the tree are encryption keys (storage keys) and they are used to decrypt child nodes. It is not very wise to use the same key for both signing and encrypting, so a signature key will never be used as a storage key and vice versa. Therefore a storage key can exist anywhere in the tree, but a signature key will always be a leaf key. Further, a migratable storage key will never be the parent of a non-migratable TPM object, as this would result in a non-migratable object being migratable. Of course a non-migratable node can be the parent of a migratable node without any bad consequences. Figure 2.4, which was already shown before, gives an example of a storage hierarchy.

Arbitrary data will be contained in a leaf TPM object node as the TPM never uses arbitrary data as a key. The time that is required to access the contents of a particular TPM protected object node depends on its position in the tree. Only the SRK is always available as it is stored as plain text inside the TPM. In order to get the content of some TPM protected object, the management software outside the TPM has to look for a path from the SRK to the target TPM object. First the SRK is "loaded" and used to decrypt the child TPM key object which is next in the path to the target. Then the key from that child object is used to decrypt the child TPM key object that is the next in the path to the target. This process continues and at the end target object is decrypted by the TPM and can be immediately used. There are ways of decreasing the delay in accessing a node and this can be read in [Pea03].

Sealing and Unsealing Keys and Data If arbitrary data or a key are saved by protected storage it is possible to define the future software environment that has to be satisfied before a TPM will release the data or key. This adds some extra functionality to access control. The desired software state is defined by the values of platform configuration registers (PCRs).

Sealing is the process of locking a secret to a certain software configuration. So a TPM will not unseal the secret unless the platform is in the approved software configuration. Both TPM data objects and TPM key objects can

be sealed to a specific software configuration. Two important advantages of sealing data and keys are listed below.

- By directly sealing arbitrary data the use of that data is prevented unless the platform is in the right software state.
- By sealing TPM signature keys a signature will not be performed unless the platform is in the right software state.

In figure 2.9 the sealing and unsealing is visualized [ELM+03a]. A secret S is sealed and a program N is named that is allowed to access the secret. If a program calls unseal then the secret is revealed only if the identity of the requester is the same as the identity specified in the sealed data block. The successful unseal requests are represented by dashed lines.

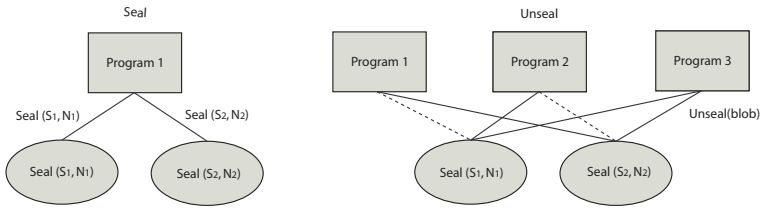


Figure 2.9: Sealing and unsealing.

Authorization There are two authorization types related to objects in protected storage. One proves rights to use the contents of the object and the other proves rights to migrate the contents of the object. There are two values used, so a protected storage object can be created and an entity can get the right to use to object but it may not have the right to copy (migrate) the object, for example.

Chapter 3

End-To-End Application Properties

This chapter gives an introduction to application layer protocols and describes their fundamental properties from the perspective of end-to-end communication between applications. Some security issues are briefly addressed by Trusted Computing technology. Then an abstraction to application layer protocols is proposed by providing a classification that tries to capture the full spectrum of protocols. The obtained classification is of great importance as it it will be used in the next chapter to address the involved security issues.

3.1 Introduction

The term "application layer" refers to the top layer of the Open Systems Interconnection (OSI) model[1] which is a layered abstract description for communications and computer network protocol design. This model divides the functions of protocols into seven layers of which each layer only uses the services of the layer below and exports functionality to the layer above. The lower layers are usually implemented in hardware and the higher ones in software. This separation of seven layers makes it easier to discuss the behavior of protocol stacks. The application layer interfaces directly to and performs common application services for the application processes. It also sends requests to the presentation layer. Furthermore, it is the application layer that contains the protocols used by applications to exchange data.

In the beginning application layer protocols were quite simple. Basically, a program on one host that wants to connect to a program on another host is able to communicate by the transfer of simple textual commands. Figure 3.1 visualizes the messages that are required to send an e-mail message from

[1] The OSI model is described in ISO 7498.

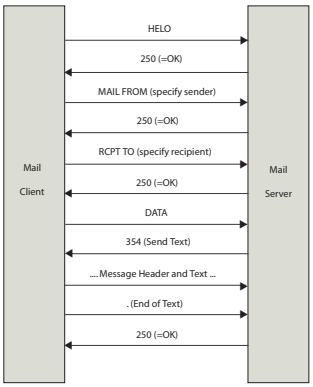


Figure 3.1: Message interactions in SMTP.

a client to a server, using the SMTP protocol. An e-mail is transmitted by sending a few lines of text back and forward between the client and the server.

Some widely-known application layer protocols are those that are used for the exchange of user information like HTTP which is used to transfer files that form the web pages of the World Wide Web, FTP which is used for interactive file transfer, SMTP which is used for the transfer of mail messages and attachments, and Telnet which is used for logging on to network hosts remotely. There are other application layer protocols that help facilitate the use and management of TCP/IP networks like DNS which is used to resolve a host name to an IP address and RIP which is a routing protocol that routers use to exchange routing information on an IP internetwork.

3.2 Architecture

There can be several processes at different computers that communicate with each other. The term architecture refers to the way applications are organized into different modules and how these modules communicate. Figure 3.2 shows a very simple architecture where a few clients communicate with one server. The server is usually responsible for a database. The protocol is a client-server protocol.

A more complex architecture that is used for example to transfer email and Usenet News can be observed in figure 3.3. There are two types of connections to be distinguished: (1) connections between a user agent and a service agent and (2) connections between two service agents. Because of the nature of these connections the protocols are different. In the case of email the protocol used for sending email (from a user agent to a service agent)

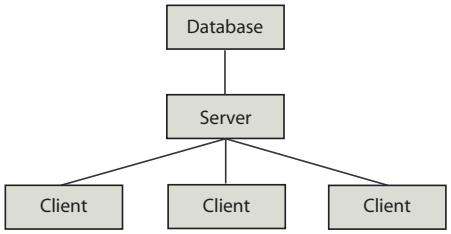


Figure 3.2: Simple client-server architecture.

is another one than the one used for receiving email (from a service agent to a user agent). The boxes are labelled "user agent" and "service agent" instead of just "client" and "server" because a service agent can be both a client and a server, depending on who opens the connection and requests the service.

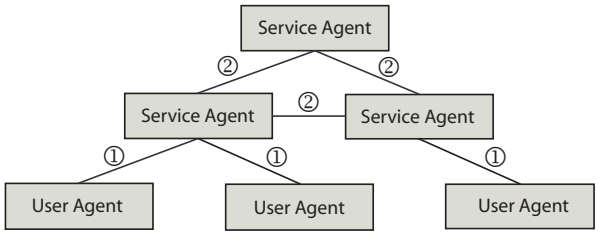


Figure 3.3: Architecture with interconnected servers.

3.3 Transfer of Responsibility

The interaction between a client and a server may include a number of interactions that are sent back and forward. An example of a common type of interaction is the transfer of responsibility. This transfer of responsibility can for example be found in email where a store-and-forward technique is used. The original sender sends the email to the local mail server which is closest to the original sender. On its turn the local mail server forwards the email to the local mail server which is closest to the final recipient. Finally, this local mail server delivers the email to the final recipient. It is very important that an email is forwarded in such a manner that it does not get lost in transit. At each moment a computer can crash or the connection between two computers can get lost. Therefore each agent stores the email in non-volatile memory so that it can always be retrieved, for example after a reboot. This implies that there is a certain transfer of responsibility

between the consecutive agents, see also figure 3.4. In chapter 5 Trusted Computing technology is used to enhance the storage of emails in such a chain of responsibility.

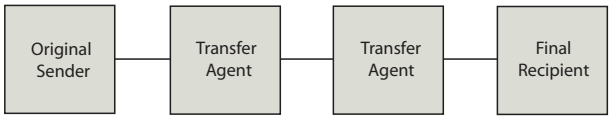


Figure 3.4: Store-and-forward of email.

3.4 Identification

Another part of the interaction is the identification between two agents. For example, assume that the server wants to tell its name to the client so that the client knows for sure that the servers is who it claims to be. This can be realized by using certificates. Knowledge about the identity of the other party may help to gain trust, however one could ask the question what assurance the client has that the server, even with that identity, is actually providing that service.

For example, assume that Bob is shopping for a video graphics card and he finds the cheapest price at an online merchant he has never heard of. Then SSL assures him that his transaction cannot be eavesdropped and it also provides Bob with the name of the merchant. However, what Bob really wants to know is if he can trust the merchant to carry out the transaction and if his credit card number is kept secret. Just the name of the merchant does not tell him this. This kind of security issue is addressed by Trusted Computing technology. In [Smi05] a way is proposed to solve this problem by:

- binding a server's SSL private key not only to the server, but also to the application which the server says it provides.
- putting the binding beyond the reach of anyone, so that even the operator of the server cannot manipulate the data.

This can be achieved by "connecting" the server end of the tunnel to the advertised application and placing both in a Trusted Computing Platform at the server site. The application then creates the SSL key pair and asks the Trusted Computing Platform to prove to a standard SSL CA that the Trusted Computing Platform binds the public key to the application at the server. The CA then issues a special kind of certificate that indicates the

binding. However, the approach discussed in [Smi05] does not discuss the trustworthiness of the application itself.

In the example above, Bob wants to be sure that the shopping software used by the online merchant only does what it is expected to do. This assurance can only be gained if the shopping software is acquired from and attested by the official party which releases that software. Then the official party can attest to the software and transmit a certificate to the online merchant. Bob is able to trust the software as the certificate vouches for its trustworthiness.

3.5 Intermediaries

In some cases the communication between two agents is passed through intermediary computers like proxies, gateways and tunnels. A proxy is a server that caches data which means that requests from a client can be directly taken care of by the proxy. Proxies are also used for all kind of regulatory or security reasons. These kind of regulating proxies are also known as firewalls.

Firewalls can block inbound or outbound traffic. Personal firewalls are only effective at blocking incoming traffic. Once a policy has been set up to block outbound traffic, the policy can be altered if a Trojan horse or malicious user is able to gain access to the firewall policy file. For example, in Microsoft Windows it is possible for any process to execute arbitrary code in the context of another process. However, this is only the case if both processes are running in the same user context[2]. For example, a Trojan horse running in the context of the logged on user can execute code as Internet Explorer. Personal firewalls that block outbound connections based on program identity fail because of this reason.

All of these attacks rely on weaknesses that are solved by Trusted Computing. In Trusted Computing, it would be impossible for a Trojan horse or malicious user to alter the firewall policy, because it is protected under sealed storage. Even a kernel vulnerability would not bypass the access controls. Only a vulnerability in the personal firewall process itself would be sufficient to alter the policy. Secondly, it is possible to run the firewall process in curtained memory, and a Trojan horse could not run code in another processes memory space.

In [GRB03] it is said as well that a proxy or firewall is significantly more powerful on a Trusted Platform as it can prevent packets from ever reaching the network in the first place. This is completely in agreement with the discussion above: a personal firewall based on Trusted Computing can be

[2]See also http://www.windowsecurity.com

effective at blocking both inbound and outbound connections. Therefore it can block attacks at the source, before the attack ever reaches the target.

3.6 Old Versus New Protocols

Many different types of computers can communicate in a network by using protocols. However when an upgrade to a protocol is required then all computers need to be upgraded as well. With standard protocols like HTTP and SMTP there are millions of agents that have to be upgraded and this would be practically impossible. If different companies develop the various agents it will be even more difficult as some companies want to support other features than other companies. It is possible to design protocols that can be quite easily extended in the future. Unfortunately, many internet standards only have a few features to add extensions and this forced developers of newer versions to use less tidy solutions. There are some methods to overcome this problem. Two of these are described below.

- **Version Number Method** By using the version number method the communication between a client and a server starts with exchanging version numbers. After this, both agents are assumed to adhere to the lowest version any of them use. This means that an agent should support both the new protocol and many or all older versions, which may be used by the other agents.
- **Feature Selection Method** By using the future selection method the communication starts with the server and client listing which features they support. After this, the communication continues and only the features are used that are supported by both agents. An advantage of this method is that an agent can choose which features it wants to support and still co-work with another agent that supports other features.

Trusted Platforms provide a mechanism called attestation, which enables remote machines to prove that they are running the expected software. As stated before, the Trusted Computing Group specifies coarse-grained attestation: the attestation is performed over the entire software platform. By using fine-grained attestation hash verification is simplified, which means that software upgrades can be performed more easily because the expected hash for each process can be updated independently [SPvD05]. Especially when protocols are frequently updated fine-grained attestation is definitely preferred.

3.7 Security Issues

There are many security issues related to application layer protocols. Various web servers, mail servers and other internet service software contain bugs that let remote users do things that are harmful. For example remote users can gain control of the machine and can do whatever they want. The exposure to these kind of threats can be minimized by running only the necessary software and getting the latest patches, and using software that has a good reputation. However there remain many security issues that cannot be easily dealt with. Firewalls are often used to deal with these kind of issues.

In the beginning of this chapter the OSI model and its layers were described. Conceptually, there exist three types of firewalls: (1) Network Layer, (2) Application Layer and (3) Hybrids [RCR04]. However, they are not as different as one may think: in today's firewalls they are combined and therefore it is no longer clear if either one is better or worse. An important thing to notice is that the lower-level the forwarding mechanism, the less examination the firewall can perform. In general, lower-level firewalls are faster, but they are easier to fool into doing things that are wrong.

The focus here will be on application layer firewalls which are hosts running proxy servers that do not permit direct traffic between networks and perform logging and auditing of traffic that is going through them. These proxy applications are software components that run on the firewall and are application specific which means that only recognized protocols can be examined. The examination techniques used in proxies are *anomaly detection* and *attack signature detection*.

In anomaly detection these proxies examine application layer protocols for anomalies and their failure to conform with standards. For example, protocol anomaly detection can recognize HTTP packets with fields that contain more bytes than the standard specifies, which probably indicates of a buffer overflow attack. The drawback of the protocol anomaly detection is that protocols may be slightly differently implemented by each application developer. The consequence is that an anomaly detector may alert in the case of a harmless difference between the two implementations. In attack signature detection the incoming packets are investigated to check if there are specific code strings that represent a known attack.

In figure 3.5 an application layer firewall, called a *dual homed gateway* is depicted. A dual homed gateway is a highly secure host that runs proxy software. It has two network interfaces, one on each network, and blocks all traffic that passes through it.

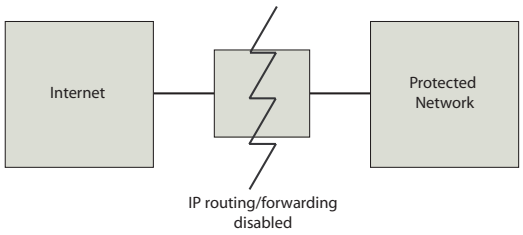


Figure 3.5: Dual Homed Gateway.

Firewalls that incorporate encryption, which protects traffic passing between them over the internet, are increasingly used. If these firewalls provide end-to-end encryption then organizations with multiple points of internet connectivity do not have to worry about their data or passwords being stolen.

Because proxies must have knowledge about the application layer protocol being used, they can also implement protocol specific security. For example an FTP proxy may be configurable so that incoming FTP is permitted and outgoing FTP is blocked. A disadvantage is that proxy servers are application specific: in order to support a new protocol, a proxy must be specially developed for it.

It is difficult and practically impossible to distinguish between all possible application layer protocols. Therefore a higher level approach must be chosen to cover all protocols without having to deal with the protocols one by one.

3.8 Classification of Application Layer Protocols

A somewhat higher level approach to application layer protocols can be realized by adopting another perspective. From the level of application layer firewalls it is all about what the actual protocol messages look like. However one can also look at how information is shared and disseminated in end-to-end communication between applications. This perspective allows an abstraction from the application layer protocols and makes it easier to look at security issues from a more general point of view.

The classification outlined below is inspired by the Domain Based Security Model described in [Goh03]. This carefully designed model helps a defence-related organization to estimate security issues at different levels of detail. The classification gives an abstraction from the protocols without focusing on implementation issues, as these are constantly subject to change. Furthermore, it enables an interesting perspective on end-to-end application

security issues that will be discussed in the next chapter.

- **Message connection** is a connection where messages are sent from one place to another. Transfer from a member of one domain to a member of another is realized through the explicit action of the former sending the message. An important characteristic of this class is that information is received without any explicit request. An advantage of this type of connection is that if the recipient is temporarily off line data will be delivered later on when the recipient is available again. Application layer protocols like SMTP and POP are designed to be used in the message connection class.

- **Shared data repository connection** is a connection where a person from a particular domain publishes data so that others can view and possibly change it. This principle is also known as the publish/subscribe method, see figure 3.6. In contrast to the previous class the information is explicitly requested beforehand. A disadvantage is that if the recipient is temporarily off line data will get lost, whereas it would be stored in the previous class. The following subclasses are contained in this class.

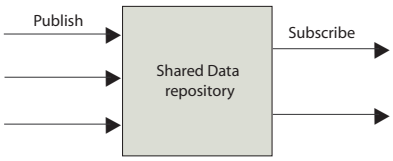


Figure 3.6: Publish/subscribe mechanism.

1. **Filestore connection** is a traditional way of file sharing where a hierarchy of files and file containers are involved. The structure enables tight control over the sharing of data. An application layer protocol like FTP is designed to be used in the filestore connection class.
2. **Web connection** is a connection where data is shared by publishing on a web server. The advantage is that more flexible structures can be created with unlimited cross-references. However, the control over the sharing of data is difficult to achieve. An application layer protocol like HTTP is designed to be used in the web connection class.
3. **Database connection** is a connection where data from an application specific database is shared to people who have access to the database. Data can only be requested by sending queries which must be in accordance with the security requirements in

order to retrieve the requested data. The advantage is that the database structures are well defined. However, the obvious disadvantage is that detailed security requirements have to be adapted to the specific applications.

- **Conferencing connection** is a connection that allows people to interact in a real-time manner and passes large quantities of data in a publish/subscribe model similar to a repository. Although it gives people the real-time advantage it is really difficult to control the information that is sent. The protocols that are contained in this class are based on a peer-to-peer model. Video-conferencing is an instance of this class and it is unlikely that many checks can be made on the data as there is a requirement to view it in real-time and slow downs are not acceptable. Other examples are VoIP and whiteboarding and the same counts for them.

It should be noted that a specific application layer protocol may not exclusively belong to only one class or subclass. For example, SOAP can belong to both the message connection class and the shared data repository class. This is due to the fact that SOAP can deliver data in two different ways: (1) SOAP can deliver data in the form of a message when the recipient is temporarily unavailable or (2) SOAP can deliver data by pushing it (publish/subscribe principle) towards the recipient when the recipient is available.

The obtained classification enables an interesting perspective on security issues, without focusing on the individual application layer protocols. The nature of the classification ensures that it will be valid for a long time, even when the protocols have undergone some changes or when new protocols are invented. The following chapter will address the involved security issues.

Chapter 4

Classification and Security Issues

In the previous chapter a classification of application layer protocols was presented which makes it easier to look at security issues from the perspective of end-to-end application security. The classification looks at how information is shared and disseminated between end hosts, without focusing on the individual application layer protocols. The underlying mechanisms used by application layer protocols to achieve end-to-end communication are not of importance. This approach is completely in agreement with the OSI model described earlier.

Each of the following sections describes the security requirements for the classes contained in the classification. The sections also provide some brief discussion on how Trusted Computing can be used to address the various security requirements. A more detailed and technical approach can be found in the next chapter where some concrete scenarios will be investigated. That chapter will also clarify which benefits and drawbacks can be encountered in using Trusted Computing in end-to-end application security.

4.1 Message Connection

In the message connection class the transfer from a member of one domain to a member of another is realized through the explicit action of the former sending the message. The fact that messages are received without an explicit request is an important feature of this class. A consequence of this is that hosts might be flooded by unsolicited messages and therefore cannot provide their services anymore. Messages that cannot be directly delivered to the recipient are usually stored in non-volatile memory which creates the risk of an adversary gaining access to the messages. In this section these kind of security issues are further discussed. They are also briefly addressed

by the trust mechanisms provided by Trusted Platforms.

Denial of Service Attacks and Rate Limiting A denial of service attack is realized by the explicit attempt of a hacker to prevent legitimate users of a service from using that service. Hackers usually try to flood a network in order to prevent legitimate network traffic. In order to tackle denial of service attacks it is important to invent a mechanism that deals with flooding. By rate limiting the amount of messages that can be sent from one host to another the roots of the problem are directly addressed.

Messages can be stamped by a rate limiter and then sent into the network at most once every time period. It is important that the rate limiter is run as an isolated process and that all messages are first delivered to the rate limiter. The rate limiter will then decide whether the message will be stamped or not. The receiving host(s) can discard any incoming messages that are not stamped or invalid. There already exist Trusted Computing technologies like LT[1] that support isolated processes and therefore enable the implementation of a design like the one described here. In the next chapter a more detailed design is presented which addresses the problem of spam.

At present, one of the best known methods to realize rate limiting is using client puzzles [DN93]. The client is forced to do some costly computation for every request that is made. The Trusted Computing solution presented above provides a few important advantages over client puzzles: (1) there are no resources wasted in order to stamp messages (which is quite relevant on mobile devices where computing client puzzles can cause a significant power drain) and (2) users do not have to wait for messages to be stamped and (3) client puzzles vary a lot in their impact because of the type of platform (processor, memory speed, etc.).

Protected Storage of Messages A characteristic of the message connection class is that if the recipient is temporarily unavailable, data will be delivered another time when the recipient is available again. This feature is only possible if data that cannot be directly delivered is temporarily stored in non-volatile memory.

In the previous chapter the concept of transfer of responsibility was mentioned which is based on the store-and-forward principle. Current message infrastructure provides no means to verify the integrity and security of server machines, so an adversary may read or modify the content of a

[1]LaGrande Technology contains a set of hardware enhancements that will be part of Intel processors, chipsets and platforms. The capabilities of LT include protected execution and memory spaces, sealed storage, protected input, protected graphics and attestation. More detailed information can be found at http://www.intel.com.

message stored on disk. Trusted Computing, however, can protect messages by using a mechanism like protected storage. A big advantage of protected storage is that data can be stored in such a way that prevents disclosure of the data if the platform is not in the expected state. In the next chapter a more detailed design is presented which addresses the problem of protected storage of email messages.

Digital Signatures Digital signatures can be used to digitally sign messages so that both tampering and impersonation of messages can be prevented. As already mentioned before, Trusted Computing can enhance and support digital signatures. A couple of security issues involved in using digital signatures are listed below.

- Normally, it is possible that somebody can steal the signature key from a computer. A Trusted Platform however, can protect signature keys by using the TPM. It never releases these keys outside the TPM and uses these keys to digitally sign data that is submitted to the TPM.

- There are some issues with sending the public key to another computer. If Bob sends his public key to Alice, then there is the risk of a man-in-the-middle attack. The directory service[2] needs to be trusted. Trusted Computing may help to bootstrap such a service, but it does not solve the problem by itself. This issue is left for future research.

- There are some legitimate security concerns with sending genuinely anonymous mail. Just forging a return address will always leave some clues about the origin of the email. A Trusted Platform can improve the value of a digital signature by adding integrity metrics that show the software state of the platform when the data is signed. So by looking at the particular software state of the sender, the recipient can decide if it should trust the signature. This feature increases the confidence in a signature. In [Pea03] it is claimed that, because TP identities can be pseudonymous, Trusted Computing provides a high level of confidence in the signature while simultaneously increasing the anonymity.

  This advantage can be used to create a user group identity without disclosing the true identities of the specific users contained in that group. Imagine a group of employees who have to complete an online poll which asks them to rate the performance of their colleagues. Information like this is of course privacy-sensitive and in order to obtain

[2] A directory service belongs to a CA and provides the public keys of the certificate holders. This service makes it possible that anyone has access to these keys and can use them for the secure communication with they key holder.

honest opinions the employer should guarantee their anonymity. Because of anonymous TP identities, the employer can be sure that the poll has been completed by its employees and the employees can be sure that their true identity has not been revealed.

4.2 Shared Data Repository Connection

In the shared data repository connection a person from a particular domain publishes data so that others can view and possibly change it. In contrast to the message connection class, information is explicitly requested beforehand. This means that a client has to establish a connection to a server and usually has to authenticate itself in order to request specific information.

Client Authentication Basically, client authentication can be seen as the process in which a server checks the identity of a client before it allows access to its data. The server wants to get assurance that a client platform is who it says it is and that it can be trusted. Of course there exist various ways of client authentication, however these will not be discussed here. Instead, there will be a general discussion on how Trusted Computing can provide a more secure authentication process.

By changing standard client platforms into Trusted Platforms, client authentication can be improved. The Trusted Platforms all have a unique TPM identity key which is bound to the platform. If this identity key is later presented to the server, then the server can be sure that the client is who it says it is, regardless of the origin of the connection. Furthermore the server can be sure that it is talking to a Trusted Platform. No machine in which a TPM identity key is installed can be impersonated, as copying of TPM identities is practically impossible.

Machine identification alone is often not enough. For example assume that a client machine is known to be part of the internal network of a company. If the user on that client machine wants to have access to the shared data resources on a per-employee basis, then machine authentication must be extended with user authentication. In an internal network, machine identification is not always required (as all computers in the internal network can sometimes be trusted) and only reliable user identification is needed in order to provide access control.

User identification is another problem than machine identification, but it can also be improved by using Trusted Platforms [Pea03]. User authentication can be delegated to the client platform, instead of to a centralized server. This delegation can be performed if it is known that the software

environment on that platform will perform proper user authentication. This assurance can be gained if for example an IT department administers the platform or by checking the software state of the platform and identifying that the platform is a genuine Trusted Platform. Verification of the software state can provide confidence that the platform can be trusted to perform user authentication locally. So the role of the server is just to check that a correctly identified machine is trying to access a resource and that this machine provides credentials for the use of this resource. This approach also reduces the complexity of the distribution of user credentials.

As a result, Trusted Platforms can improve client authentication and access control by improving confidence in the authentication process and by simplifying user credentials management.

4.2.1 Filestore Connection

In a filestore connection the way of file sharing and the structure of the hierarchy of files and file containers enables tight control over the sharing of data. The principle of access control was already mentioned once above. The filestore connection is probably most suitable for the implementation of enhanced access control using Trusted Computing.

Access Control Trusted Computing can enhance access control in ways that have not been possible before. First some general introduction to access control is presented below. Then a brief discussion on how Trusted Computing can provide enhanced access control is given.

Basically, access control is a method of restricting access to resources by allowing only privileged entities. Access control mechanisms may work, but past experience has shown that access control can be easily overcome. Most of the time access control is nothing more than an attribute for a set of data. If the managing software (the OS or an application) respects the attribute, then the access control mechanism is secure. However, when another OS or application tries to access the data, then the access control mechanism does no longer provide protection. It is claimed that improvements in access control mechanisms do not improve the level of security, so another technology is required. The solution to this problem is encryption and is considered as the ultimate level of data protection. The things that are required are the keys that are used for encryption and decryption. The only security issue is the proper handling of the keys so that a malicious party is not able to get access to these keys.

Trusted Computing provides a way of keeping these keys at a safe place: the

keys are protected by the protected storage key hierarchy (the root of this hierarchy is the Storage Root Key which does not leave the TPM). Another mechanism that can be used to enhance access control is protected storage. Secrets can be sealed to a particular operating system or to a platform configuration. This means that data can be stored in a way that prevents use of the data unless the platform is in the expected state. In the next chapter enhanced access control is discussed in a lot more detail.

4.2.2 Web Connection

In a web connection data is shared by publishing it on a web server. At present, the web is often used in information services. Basically, a client (that is, the browser) sends a request to a remote server, which then responds with data. Without any security measures, the exchange of information takes place in plain text which results in data getting exposed to potential adversaries. Another security issue is that the browser user cannot be sure whether a connection has been established to the intended server, or whether the adversary (by performing some DNS attack for example) is impersonating the other end [Smi05]. The secure sockets layer (SSL) is normally used to address these security risks.

Web Security A problem with using SSL to solve the "web security problem" is that it only protects the traffic between the browser and the server and in the standard instantiation it only authenticates the server identity. As already mentioned in chapter 2 there are still some security issues to be addressed:

- What happens to the data if it arrives at the server?
- What assurance does the client have that the server with that specific identity is actually providing that service?

In chapter 2 a way to solve this problem was described. In [Smi05] a trusted co-server is proposed, which is basically formed by the Trusted Platform combined with the application that has to be attested. The result is that an authenticated and secret shelter for the particular application is provided. The client, server and maybe even other stakeholders can then trust this shelter for maintaining a high level of security. The use of a trusted co-server enables many possibilities in the field of web applications. Below, a couple of interesting applications of a trusted co-server are described to give some idea of how hardened web servers can be realized using Trusted Computing. The following examples are taken from [Smi05].

Integrity of Server At present there is no way for the client to check the integrity and site security of a server. For example, a server may run on a

machine where a hardened operating system is run and/or other good security practises are performed. The question here is how a client can get to know this.

This problem can be solved by using a trusted co-server which can witness that the appropriate security tool or secure boot was actually applied to the host. The tool may have been applied from the trusted co-server itself or from a companion trusted machine. In this way the client can trust all kind of assertions it receives through the SSL-authenticated communication channel between the trusted co-server and the client.

Privacy of Sensitive Web Activity Currently there are no ways for a server operator to deny that he is monitoring all client activity. In the same way the operator cannot deny that an adversary who comprised his machine is monitoring this data.

In order to solve this problem for the data retrieval case, the trusted co-server can implement a variation of a RAM algorithm which treats encrypted storage in the server's file system as the "RAM". By making use of an SSL-protected channel, the client can send a request to the trusted co-server which then retrieves the record via the algorithm, re-encrypts it, and returns it to the client. In this way, the server operator will not notice anything (it only sees ciphertext) except the fact that a query has been made.

Non-repudiation of Client Authentication Without the public key infrastructure (PKI) web users are forced to authenticate themselves by using a username and password. However, a user can never know the integrity of the server to which it sends the authenticators to. This implies that an adversary or a malicious server operator can impersonate the user at this site and at all other site where that user can use its authenticators. This also prevents legitimate server operators from being able to judge if it really was a particular client which opened a particular session.

This problem can be addressed by using a trusted co-server that can retain the password and authenticate the client. After a successful authentication it can send a signed receipt to the server with which the client wants to have a session.

Safety of Downloadable Content Currently there are no ways for the client to make sure that an executable content downloaded from a server is safe. Safety depends on the clients running the latest anti-virus software. Unfortunately, most people do not take this issue seriously and are therefore at risk.

This problem may be solved by moving the anti-virus software (and the problem of maintaining the latest updates of virus signatures) to the server. The trusted co-server could run the anti-virus software with the latest signatures. This can be done either dynamically, as the co-server is transferring data back to the client or offline. In this latter case the trusted co-server would check that it had indeed scanned this data before. In this way, clients can trust that downloaded content via the SSL-authenticated channel from the trusted co-server has been scanned. This process has to be optimized of course, otherwise the anti-virus software might ask too many resources of the TCP.

4.2.3 Database Connection

In general a database connection uses a non-standard protocol that is specific to the database software in use. For example, MySQL, which is quite a well-known DBMS, uses an extremely simple protocol. It sends the SQL statement to the DBMS in a network packet and returns the results in a comma separated format[3].

A database typically uses access controls that can restrict the tables and columns that a specific user can access and whether they can read, update or create new entries or database structures.

The security requirements for a database connection are somewhat more complicated than the filestore or web connection. Databases are mostly used to store large quantities of sensitive information and each class of user should have a different view of that data. Often, a lot of information can be collected indirectly. For example, if a database tells a user that a record does not exist, they may be able to infer information from the database even if they do not have access rights to it.

It is difficult to implement databases by using isolation countermeasures like compartmented operating systems (mandatory access controls) like SELinux or Trusted Solaris. Typically the DBMS must have access to the entire database in order to perform global searches. A vulnerability in the DBMS then gives the attacker access to the entire database. The same problem occurs using virtualisation, where you might want to put a public database in one operating system, and a more sensitive database in another.

Most of the time it is necessary to restrict queries to prevent a malicious party from downloading the entire database. For example, imagine an online dictionary. The web site may allow users to query on particular words. The

[3]See also http://dev.mysql.com

owners of the dictionary are happy to do this because a human user will only query on particular words. Now, imagine a competitor of the online dictionary that wants to steal the online dictionary. The competitor can write a program that queries every word, at a rate of several thousand per minute, and will then be able to download the entire dictionary. One way to prevent this kind of attack is to rate limit queries from a particular connection. The same kind of approach, described in denial of service attacks and rate limiting above, can be taken to limit the amount of queries that are sent into the network.

4.3 Conferencing Connection

A conferencing connection allows people to interact in a real-time manner. As said before it gives people the real-time advantage but it is really difficult to control the information that is sent. For example, a video connection passes large quantities of data in a publish/subscribe model similar to a repository. It is unlikely that many checks can be made on the data as there is a requirement to view it in real-time and slow downs are not acceptable.

There has already been done considerable work in secure routing and network connections in a peer-to-peer model. However, in this thesis the focus is on the end-to-end communication between platforms. Since each stream can be encrypted by the initial platform, the security problem in the intermediate nodes will be be ignored. So the main security concern lies on the end platforms. An important requirement is the real-time protection for video-conferencing and whiteboarding, for example.

In recent peer-to-peer VoIP applications like Skype, audio streams are routed and delivered to the recipient by making use of active peers, which is a similar mechanism as that in traditional peer-to-peer file sharing systems[4]. The main difference is that in the case of Skype it is people (userIDs) instead of files that are searched and after a successful search a real-time communication is started rather than a file transfer.

Besides the security considerations in routing and network connections, the real-time protection of audio data in a platform is a new issue. Real-time protection prevents a malicious party from eavesdropping or illegally recording a conversation. In general, an audio stream is encrypted so that intermediate nodes cannot have access to the data. To ensure that in an end platform an audio stream is not illegally accessed by other applications or processes, the initiator of the conversation needs to verify the integrity and state of the platform, including the peer-to-peer client application and the

[4]See also http://www.skype.com

audio output channel between the sound card and the VoIP application.

Trusted Computing can be used to enforce these kind of security policies on a platform. There are already existing Trusted Computing technologies like LT (LaGrande Technology) that enable curtained memory space, process isolation and secure channels between processes [SZ05]. There should also be a trusted path between an application and audio, video and I/O channels. For example, the channel from an application to an audio card and driver should be protected so that no other process can read or change the content. This feature can also be realized by a Trusted Computing technology like LT.

Chapter 5

Trusted Computing Applied

In the preceding chapter the application of Trusted Computing to the classification of application layer protocols was discussed. The approach was quite general and can be used in the future as it does not focus on the individual protocols themselves. However, the information provided in the previous chapter was still quite abstract.

Therefore it is time to get more concrete and to investigate the possibility of applying Trusted Computing to the architecture of some distributed application components. There are four candidates that will be scrutinized: FTP, whiteboarding, SMTP, and web cookies. The first section introduces an access control architecture on which the enhancements of FTP and whiteboarding are based. An important feature of FTP, called access control, will be discussed in the light of Trusted Computing. Then the end-to-end communication of a whiteboarding session is discussed. In another section, several aspects of SMTP that can be enhanced by Trusted Computing are examined. At the end, the privacy and security issues of web cookies are investigated.

5.1 Architecture for Access Control Using Trusted Computing

Basically, access control is a method of restricting access to resources by allowing only privileged entities. There exist two interesting papers about access control and Trusted Computing: [SZ05] and [SJZvD04]. Both papers describe access control architectures and mechanisms based on Trusted Computing. In this thesis the architecture proposed in [SZ05] is chosen as the flexibility it provides for distributed computing systems is relatively higher. Below, the new requirements of the security model and architecture are listed.

- **Change of Trust Relation** Traditionally, sensitive objects and policy enforcements are located on the server-side and a client usually trusts a server. Once the information is released to the client there is no further control. However, in a many of today's circumstances a server needs to trust the client as well with respect to both the platform and the user authentication. The server here distributes objects to other platforms.

- **Location of Policy Enforcement** An important requirement of the security architecture proposed here is that the security policies are enforced on the client platforms. The policy enforcement depends on the trust between the two platforms.

- **Trust of Platform and Application** In many traditional security systems (like mandatory access control) security policies mostly consider the properties of subjects and objects. However the integrity and state of the platform and running software is not taken into account. In modern open systems mechanisms to guarantee their integrity are required.

- **Trusted User Authentication and Authorization in Client Platform** Subject authentication is necessary to enforce a security policy. In traditional security systems, a centralized server usually provides authentication mechanisms. However, in distributed and decentralized systems, a particular object or policy owner has to trust that the valid user is authenticated and authorized in a client platform before he is allowed to access a protected object.

- **Trusted Path from User to Applications and Vice Versa** Spoofing and man-in-the-middle eavesdropping or modification attacks are a few possible software attacks. Trusted path technology can help guarantee that input from the user to the application software (and vice versa) is not tampered with.

By using the mechanisms provided by Trusted Computing, [SZ05] claims that an architecture can be realized to support these kind of security requirements.

5.1.1 A Platform with Trusted Reference Monitor

The platform contains trusted components like a TPM, a secure kernel, and a trusted reference monitor (TRM) in user space of the operating system, see also figure 5.1. The hardware works together with the kernel and provides the important functions to the TRM from basic cryptography functions to platform and program attestation, sealed storage for sensitive data, and

protected running environment. The sensitive data of the TRM includes the secrets and policies. A secret can be an encryption key for an object, which is originally from the object owner and distributed to the platform. A policy is made by the object owner and controls the access to this object in this platform. A policy specifies the integrity state of an application on a genuine platform where the object can be accessed. The security attributes that belong to a subject may also be specified in a policy, like a security clearance or a role name. Secrets and policies are sealed by a TRM so that they can only be read in a valid integrity state. This guarantees that the policies are well enforced by the TRM and that the secrets are not revealed outside the TRM. The policy enforcement is realized by attestation, which will be explained later on.

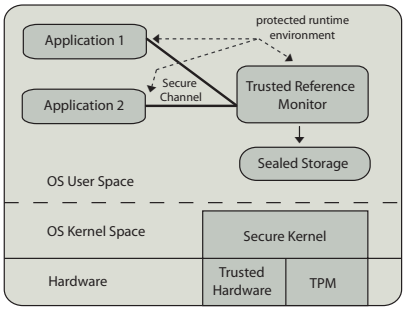


Figure 5.1: The platform architecture.

A relevant security requirement is the integrity and confidentiality of the protected runtime environment for every application, including the TRM. This involves two aspects: first, the memory space of an application must be private, so that it cannot be accessed by other applications, this also accounts for devices with direct memory access (DMA) capabilities. This means that an application which is able to access an object cannot release it to other applications without the explicit opening of a secure channel. Secondly, the communication between applications must be protected by secure channels which are only available to the corresponding applications. There are already existing Trusted Computing technologies like LT which provide curtained memory space, process isolation and secure channels between processes.

A trusted path between an application an graphics and I/O channels is another important security requirement. For example, the channel from an application to the graphic card and driver should be protected so that no other process can have access to the data. At the same time inputs from a keyboard and a mouse have to be protected as well and Trusted Computing

technologies like LT support these functions.

The secure kernel's function is to separate execution between upper layer applications and related services like TCG Core Services (TCS) and TCG Service Providers (TSP), for more information on this the reader is referred to [SZ05].

Credentials The set of credentials and the corresponding certificate authorities presumed are listed below.

- TPM attestation identity key (AIK) pair $PK_{TPM.AIK}$, $SK_{TPM.AIK}$. The AIK is created by a TPM and used to sign PCR values and to present to a challenger in an attestation protocol, or to sign a public key of an application running in the platform for authenticity. The private part of an AIK is protected by the TPM by the storage root key (SRK) and the public key certificate is issued ba a trusted third party such as a privacy CA. The TRM can have a number of AIKs with certificates from different CAs.
- TRM asymmetric key pair PK_{TRM}, SK_{TRM}. Every TRM has this key pair for signature and encryption. The private key is protected by the TPM in the platform so that only the TPM on the platform is able to make use of it (this is realized by checking the integrity value). The public key is in the form of a certificate signed by an AIK of the TPM.
- Application asymmetric key pair PK_{APP}, SK_{APP}. This is similar to the TRM, each application has an asymmetric key pair. The private key is protected by the TPM and the public key is in the form of a certificate signed by an AIK of the platform.
- TPM storage key(s) to protect TRM's credential and other sensitive data with sealed storage, like secrets and policies. This key has to be either the SRK of a TPM or a key protected by the SRK.

Primitive Functions of a TRM By using the capabilities of a TPM, a TRM has the following primitive functions.

- *TRM.Seal($\mathcal{H}$(TRM), x).* This function seals data x by a TRM which has integrity measurement of $\mathcal{H}$*(TRM).* The x can only be unsealed under this TRM when the corresponding PCR value is $\mathcal{H}$(TRM). The actual key that is used in the sealed storage is a TPM storage key.
- *TRM.UnSeal($\mathcal{H}$(TRM), x).* This function unseals x provided $\mathcal{H}$(TRM) was the value that was used to seal x.
- *TRM.GenerateKey(k).* This function generates a secret key k.

- *TRM.Attest($\mathcal{H}$(TRM), PK_{TRM})* = $\{\mathcal{H}(TRM) \mid PK_{TRM}\}_{SK_{TPM.AIK}}$. This function generates an attestation response by returning a certificate of the TRM's public key concatenated with its integrity value, signed with an AIK private key of the TPM.

5.1.2 Architecture

In order to control access to an object in a client platform, first the the policy and the object encryption secret have to be generated and distributed. Figure 5.2 shows the case where a server (Alice's platform) attests a client (Bob's platform) and distributes policies and secrets to the client platform. The general process is listed below.

1. The TRM of Bob's platform issues an access request message to the TRM of Alice's platform in the case it has not requested this object before. This request may originally be from an application APP_B in Bob's platform: Bob causes APP_B to access *OBJ*. The integrity measurement $\mathcal{H}(APP_B)$ signed by one AIK of the TPM in Bob's platform may be included in this message, which can be available to the TRM by an attestation challenge in the same platform. At the same time, an object identity *OBJ_ID* may be included in this message.

2. The TRM of Alice's platform checks the integrity of the requesting application. If Alice trusts APP_B to enforce some basic policy (for example, APP_B will not change an object or save in it in plain text to persistent storage) as well as additional object-specific policies that Alice can specify, then Alice's TRM sends an attestation challenge to Bob's TRM.

3. Bob's TRM executes the *TRM.Attest($\mathcal{H}$(TRM), PK_{TRM})* function, which returns a certificate of Bob's TRM running hash and its public key, signed by the private key of his platform's AIK and sends it back it Alice.

4. Alice checks the attestation. If Alice trusts the platform (which means that the platform has a genuine TPM and trusted booting) and the running hash of the TRM, then Alice's TRM creates a secret key k_{OBJ}, encrypts this key together with the policy information using the public key of Bob's TRM, and sends it to Bob's platform.

In order to ensure confidentiality of the policy and secret, the TRM in Bob's platform seals the items with its own integrity measurement value. Secrets and policy information never leave the TRM. The TPM forms an application domain, which is isolated from other application domains and communication between these domains is encrypted.

Secrets that are created by a server platform and sent to a client platform are used to encrypt protected objects during distribution. For example, Alice encrypts the target OBJ with k_{OBJ} and distributes it to Bob's platform. An object can be distributed together with a policy or without. Because it is encrypted and the key is only available to a TRM, the security of the object is maintained during distribution.

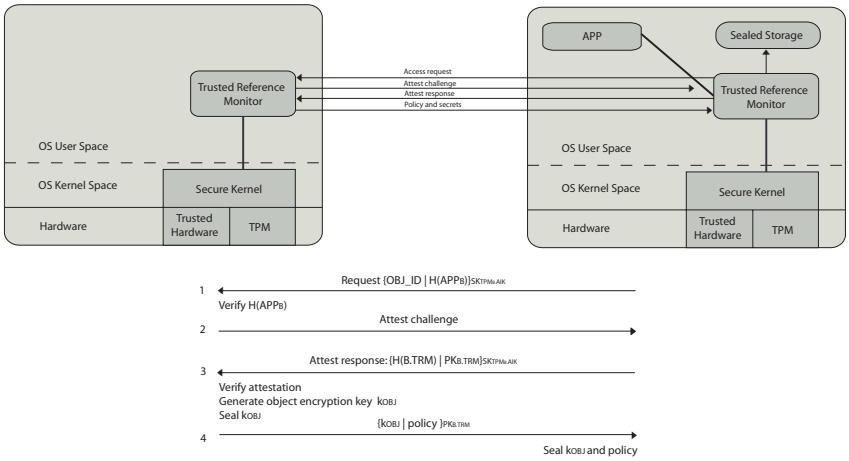


Figure 5.2: Architecture for client-side policy enforcement.

An essential property of secrets and policies is migratability. A migratable object can be re-distributed from a platform to another platform while a non-migratable object cannot be re-distributed. Another option is whether the key is kept or deleted on the original platform after a distribution. The implication of deleting a key after a migration is that the object is only accessible in a single platform at any time. The application and policies are responsible of determining and specifying these options. Fine-grained policies can be designed to specify complex situations, such as Alice can re-distribute an object to Bob, who cannot re-distribute it to others.

Policy Enforcement When a secret and a policy are distributed, a subject on the client platform can issue an access request by invoking an application or process. Then the TRM in the platform verifies the application's integrity state based on the policy information of a policy when an access to OBJ is generated from an application APP_B in Bob's platform. This procedure is presented in a high-level description below.

1. APP_B sends the request *view* OBJ to the TRM, with the object encrypted by the secret key k_{OBJ} which is distributed from the object owner.

2. The TRM then sends an attestation challenge to APP_B.

3. APP_B responds with its running integrity measurement (which consists of one or more PCR values) and its public key signed by the TPM's AIK.

4. The TRM compares the integrity measurement with a list of expected values according to the policy. If APP_B can be trusted, the TRM creates a session key k_s and encrypts it with the public key of APP_B and then sends it to APP_B. At the same time, the TRM unseals k_{OBJ} and decrypts OBJ with k_{OBJ} and encrypts OBJ with k_s and finally sends this back to APP_B. If necessary, the TRM updates the policy, for example to update a usage count, see also figure 5.3.

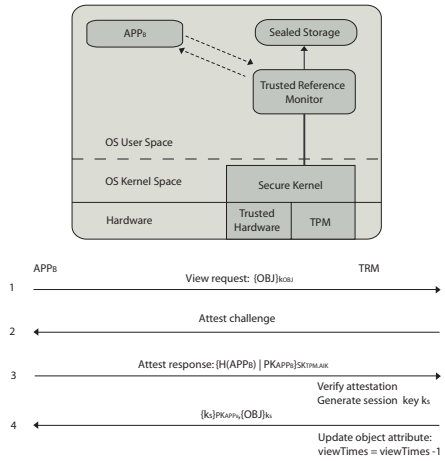


Figure 5.3: Policy enforcement in a client platform.

5.1.3 Policy

A policy is created by an object owner, distributed to a client platform, and enforced by the TRM in that platform. In general a policy is sealed in a TRM and it can be logically assumed that each object is related to a specific policy.

Basically, a policy specifies under which conditions a subject can access a specific object. In reality the condition consists out of one or more platform configurations. A platform configuration consists of a trusted platform attestation identity name (which is represented by an AIK certificate) and an application's integrity measurement with expected properties. A policy can

specify that a specific property of the accessing application must be satisfied. For example, an application is not allowed to save an object in plain text to any persistent storage. This can be certified by a trusted third party.

Figure 5.4 shows an example of a policy specified in XML. The "migratable" attribute with boolean value of the <policy> entry defines whether this policy can be migrated to another platform or not. A <subject> entry specifies the attributes that are needed for the accessing subject, like an identity certificate and a role name. Multiple <subject> entries may be part of a policy, all of which have to be satisfied in an access. A <right> entry can have some parameters, such as, an object can be viewed 5 times (*viewTimes* attribute with value 5). A <condition> with type TP defines a platform environment, including a running applications state, such as version, integrity hash, and a conformance certificate. A conformance certificate is issued by the vendor of the application software and certifies a set of properties of the software.

More complex policies can be specified according to the requirements of the object owner. For example, if a policy is migratable, then the possible platforms that a policy may be migrated to can be defined in the policy.

5.1.4 Policies and User Attributes

Till so far the architecture describes access control that is based on the properties of platforms and applications only. An object owner may also want to control by which user an object can be accessed. Generally, a user is associated with one or more security attributes, like a role name or a clearance level. Figure 5.4 shows that a subject type is an attribute name, such as "role" and the value is a role name. User-based access control policies are enabled by extending the architecture described above.

A user identity can be imported into a platform if there is a User Agent (UA) in each platform. A UA can be both an independent service or a component of a service that manages user authentication and identities. Like a TRM, a UA in a platform has a key pair PK_{UA}, SK_{UA}. Each user has at least one identity key pair PK_u, SK_u, which is a migratable object protected by a TPM and encrypted with the UA's credential in the protected object hierarchy of the TPM. A migratable key can be generated by the local TPM or some other platform and can be securely migrated from one platform to another platform with the authorization of the key owner.

Identity and Role Certificates In principle a user can be certified by a trusted party just like a platform attestation identity key is certified by the pivacy CA. A user receives a role as well, which is also done in the form of a certificate by a role server. A role certificate is based on a user's identity

and contains the role name. The mechanisms to obtain a user's identity and a user's role certificate are not described here but can be read in [SZ05].

```
<?xml version="1.0" encoding="UTF-8"?> <policyns
xmlns="http://www.example.com/tc-policy"
owner="example.com" filename="mypolicy">
<policy migratable="false">
 <object_id type="object_type">object_ID</object_id>
 <subject type="cert"> Bob </subject>
  <subject type="role"> employee </subject>
 <right name="view">
  <param name="viewTimes" value=10/>
 </right>
 <condition type="TP">
  <platform_id> Bob_office </platform_id>
  <environment>
   <title> APP_B</title>
   <version> 1.0</version>
   <integrity alg="sha1"> 0x487A3D... </integrity>
   <certificate> 0xA48ED...<certificate>
  </environment>
 </condition>
</policy>
```

Figure 5.4: An example of a policy.

Role-Based Policy Enforcement in a TRM In order to enforce a role-based policy in a platform, a TRM starts with sending an attestation challenge to the UA in the local platform after which the UA responds with attestation information. If the TRM decides to trust the running UA, it sends a request for role information of the user that invokes an application to access an object, which is issued either by a role server or migrated from another platform. The UA sends back the role certificate of the user.

Migration of User Credentials The idea of an identity of a user is quite close to that of the attestation identity of a TPM. An attestation identity key is flagged as non-migratable when it is created, because it has to be tightly bound to a single platform. In general, a user should be able to access a number of platforms, such as a desktop at home and a laptop in the company. This implies that a user identity key is a migratable object, and can be copied from one platform to another under the authorization of the user. The identity owner can define the restrictions on the set of the destination platforms that an identity key can be migrated to. Trusted Computing enables the copying of a migratable key object protected by a TPM to another TPM. Figure 5.5 shows a way of copying a user identity from the source platform to the destination platform directly. However, this requires that both platforms are available at the same time. In reality intermediary entities are possibly needed and this is also supported by TPM specifications [Pea03]. The migration process is listed below.

1. The UA of platform 1 (source) sends an attestation challenge to the

UA of platform 2 (destination).

2. The UA of platform 2 responds to the attestation challenge with the corresponding PCR value and its public key, signed with the AIK of the TPM in platform 2.

3. The UA of platform 1 checks the attestation. If platform 2 and its UA are trusted, then the UA of platform 1 unseals the identity key and encrypts it with the public key of the UA in platform 2 and sends it back to platform 2.

4. The UA in platform 2 decrypts the received identity key and seals it with a storage key of the local TPM.

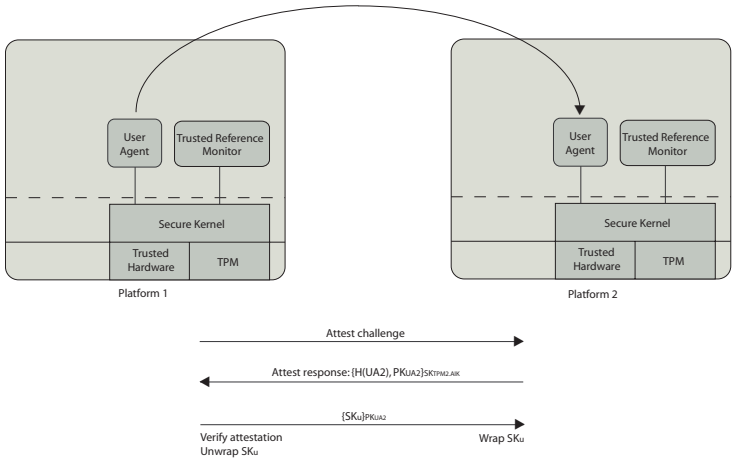


Figure 5.5: User identity migration.

5.2 FTP and Access Control

FTP, the File Transfer Protocol, is one of oldest application layer protocols which is still in widespread use. It was designed in a time when computers could trust each other and security attacks were quite rare. An important security drawback of FTP is that is does not encrypt the username and password of the client, which may make the account vulnerable to an unauthorized attack from a person or eavesdropping. Therefore FTP is usually not recommended, except for anonymous FTP. The secure alternatives for FTP are Secure Copy (SCP) and Secure FTP (SFTP) which do provide encryption.

Anonymous Login In contrast to FTP, SFTP does not directly provide anonymous login because a username and password are required to access the server. It must be noted that providing anonymous access may not be a security advantage because then it is difficult to perform accounting and audit: it is impossible to tell the difference between several users logged in under the anonymous account. However, Trusted Computing may be used to allow users to be anonymous so that the server does not know their personal identity or name, but still can tell the difference between several anonymous logged on users. The TCG specification provides "pseudonymous authentication" technology that addresses this issue. A user can use a TPM identity to present credentials to the server that show that the platform is unique which gives protection against spoofing. Furthermore, the credentials show that the platform is reliable as they include the platform state. This is all realized without disclosing anything that can be related to the specific user.

Is is also possible to implement anonymous access in SFTP by adding a guest account and then sharing the username and password with several users. This is basically how anonymous FTP works, except the username is fairly standardized: it is usually "anonymous" with an email address as password.

Enhanced Access Control All file transfer protocols mentioned above are based on a traditional architecture where sensitive objects and policy enforcements are located on the server. The following text, which is an excerpt from RFC 959, explains what is understood by access control in FTP:

"Access controls define users' access privileges to the use of a system, and to the files in that system. Access controls are necessary to prevent unauthorized or accidental use of files. It is the prerogative of a server-FTP process to invoke access controls."

This statement clarifies that access control in FTP does not deal with unauthorized or accidental use of files when the files are not under the supervision of the FTP server anymore. So, once a client downloads objects from an FTP server, the information is released to the client and there is no further control.

However, due to recent and emerging technologies like Trusted Computing, an FTP server is able to trust the client with respect to both the platform and the authentication. It would be a considerable improvement if the object owner knows for sure that the security policies are enforced on all platforms the object is distributed to. In this section the access control architecture described above is used to realize an enhanced form of access control.

It is important to notice that in the approach chosen here, the access control policy is not sent separately but is sent together with the specific object it is related to. The access control policy and the corresponding object are contained in a single object. This single object can then be encrypted and decrypted by the TRM. The reason for this solution is that the access control policy and the object of interest would otherwise have to be stored at two different places in the system. This means that when a client requests an object, the FTP server has to know where to find the corresponding access control policy and this would add needless extra complexity.

The control over the dissemination of data is a fundamental problem in access control. By using the trusted access control architecture described above a policy like "Alice downloads *OBJ* from Bob's FTP server after which it can be viewed 5 times" (P1) can be formulated. This policy can be further extended by including user role attributes, like "Alice downloads *OBJ* from Bob's FTP server after which it can be viewed 5 times only by the employees of the Alice's company" (P2). Under certain circumstances objects must be moved to other platforms (platform roaming) and then a policy could look like "Alice downloads *OBJ* from Bob's FTP and it can be viewed 5 times on the office desktop and laptop" (P3). These three policies are further discussed below.

Simple Policy Policy P1 is platform-based only and can be enforced by using the basic architecture shown in figure 5.2 with the policy that is specified in figure 5.4, although there is no <subject> field. The parameter *viewTimes* is created for <right> to indicate the available times that the object can be viewed on the specific platform. The TRM of Alice's platform updates the value of *viewTimes* each time an authorized access is performed. The policy here is non-migratable because Alice is not allowed to distribute the object to other platforms.

Role Included in Policy Policy P2 also contains a role attribute of a user, so that every employee in the company can view the document on Bob's platform. The policy is identical to the policy shown in figure 5.4, although there is only a single <subject> entry with type "role" and value "employee". Again, the policy is non-migratable because Alice is not allowed to distribute the document to other platforms.

Accessability From Different Platforms Sometimes a user must be able to access an object from different platforms (platform roaming). Ideally, both platforms are available at the same time and the object can be transferred by direct migration of the key and policy from one platform to the other when both of them are migratable. This can be realized in the same manner as is done in identity migration, explained before. However, in most

cases two platforms will usually not be available at the same time and therefore one or more intermediaries may be used like a centralized online service [SZ05]. An online policy service component can be introduced in order to support platform roaming. The object owner first has to trust the component to store and update policies.

It is a problem when the consistency of possible policy updates in different platforms for a single object has to be maintained. Policy P3 says that Alice can view the object at two platforms with a total usage of 5 times. So if Alice reads the object once on her desktop in the office by migrating the policy and secret from her laptop, the policy should be updated, which concretely means that *viewTimes* is decreased by one. When Alice wants to view *OBJ* again on the laptop, the policy should be synchronized in order to count the access in the desktop. This can be solved by performing a policy migration each time a platform roaming happens. A second solution is an online trusted policy service component on which each update is performed.

The given examples are all one-step user-to-user transfers of objects. However, the chosen architecture also supports multi-step transfer control by defining all permitted steps in the policy. For example, assume that Alice downloads *OBJ* from Bob and Bob allows Alice to distribute *OBJ* to Charlie (Alice can either upload the object to Charlie or Charlie can download the object from Alice). Then Bob must place both Alice's and Charlie's information (platform, running environment and user attributes) in the policy, after which Alice can download the object and policy. The TRM in Alice's platform can enforce the policy by verifying Charlie's platform and user attributes if there are any requests. Also in this case a trusted online policy component can be an option to enforce multi-step transfer control.

5.3 Whiteboarding and Trusted Computing

A whiteboarding connection allows clients to make changes to each other's screens in a peer-to-peer model. For example a user can draw a diagram on his screen and it would automatically appear on everyone else's screen. In contrast to a video connection which can be implemented over a one-way device (as there is no need for control signals to flow in the opposite direction other than possible acknowledgement packets for quality of service, but these could be omitted) a whiteboarding session has to allow two-way data exchange. Unfortunately, not much information can be found about protocols used in whiteboarding and therefore a more general approach is taken which is based on the model of end-to-end communication in a peer-to-peer connection, described in [SZ05].

There has been done a considerable amount of work in secure routing and network connections in peer-to-peer. If the encryption of the stream generated by the initial platform is assumed, the security problem in the intermediate nodes can be ignored. Therefore, the main security concerns lie on the end platforms. An important requirement is the real-time protection of a whiteboarding connection in a platform.

A real-time protection demands that the streams in a platform are not eavesdropped or illegally recorded by other processing during the whiteboarding session. Besides process isolation in runtime space, another requirement for whiteboarding is the secure channel between the client application and the video card driver. The TRM has to secure the communication between the client application and video card so that a possible compromise is prevented. Trusted Computing technologies such as LT (LaGrande Technology) make it possible to establish a trusted channel between an application and input or output devices. Figure 5.6 shows the architecture of an end-to-end whiteboarding session.

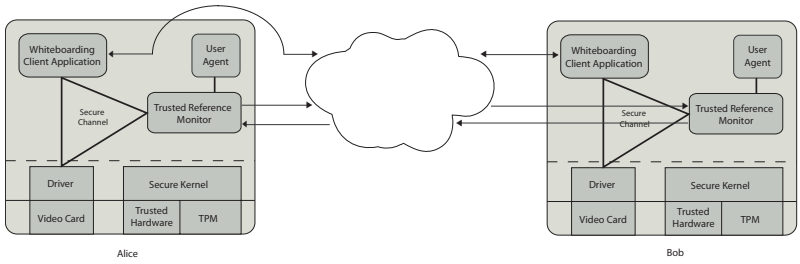


Figure 5.6: Secure whiteboarding architecture.

The distribution of the whiteboarding stream encryption key and policy between the TRMs is identical to the policy and secret distribution described in the access control architecture discussed above. Furthermore, the integrity of the loaded video card driver is also attested and checked by the TRM in the sender's (Alice) platform. Mutual attestation may be required if the receiver (Bob) needs to trust the sender's platform and application because a whiteboarding connection is bidirectional. In this section a one-way flow will be discussed only. In the sender's platform, the whiteboarding client application accepts video streams from the video card by using a secure channel with the video card driver. After the video stream is encrypted by the TRM with the object encryption key k_{OBJ}, it is sent to the receiver's side. Bob's client application receives the encrypted stream and sends it to the TRM for decryption and finally sends it to the video card through the secure channel. In [SZ05] the policy enforcement in the receiver's platform is

discussed in quite some detail. A similar kind of policy enforcement is now described in the context of a whiteboarding client application. The various steps of the policy enforcement in the receiver's platform are shown in figure 5.7 and explained below.

1. The TRM challenges the whiteboarding client application and receives the attestation response.

2. If the integrity verification turns out to be valid according to the specified policy, the TRM creates a secret key k_s and sends it to the client application.

3. The TRM challenges the video card driver and receives an attestation response.

4. If the integrity verification turns out to be valid according to the specified policy, the TRM sends the secret k_s to the video card driver. Then the client application and video card driver both poses the secret key k_s which will be used to encrypt the stream between them.

5. If any subject information is specified in the policy, the TRM sends an attestation challenge to the User Agent together with the expected information of the user that calls the whiteboarding client application.

6. The User Agent sends an attestation response and the related authentication information (which are attributes) of the user back. If the TRM decides to trust the User Agent and the user attributes are in accordance with the specified policy then the whiteboarding session is authorized and the client application can start receiving and sending streams.

7. If the client application receives the encrypted stream, it first sends it to the TRM, after which the TRM decrypts the stream with the secret key k_{OBJ} that was obtained from the sender's TRM, and then the TRM encrypts it with k_s and sends the newly encrypted stream back to the client application. The client application must rearrange the receiving stream (because of the different delays of the streams from the network) and sends the stream to the video card driver which will finally send it to the hardware. Sending streams to the other platform goes in the same way. More precisely, the client application creates a stream, encrypts it with the session key k_s and sends it to the TRM. The TRM decrypts it, re-encrypts it with the object secret k_{OBJ} shared with the TRM of the other platform and sends it back to the client application which is responsible of delivering it to the network.

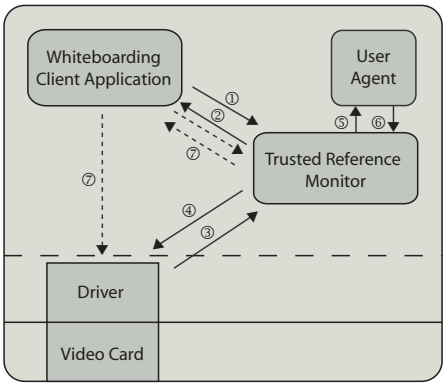


Figure 5.7: Whiteboarding policy enforcement in a platform.

The policy enforcement in the receiver's platform, described step-by-step above, includes the support for policies with user attributes. This means that users can have different access rights which have to be enforced on all platforms. For example, Trudy may have full access to all possible operations (adding, changing and deleting drawings) during the session. Alice however, may only add drawings but not change or delete them. These kind of access control policies can be specified in a straightforward manner. By migrating user identities, users can perform the same kind of actions on all Trusted Platforms.

It should be mentioned that the secret key k_s that is shared by the TRM, client application and device driver is only a one-time key and is generated each time a new whiteboarding session is started. The object secret k_{OBJ} that is shared between the two TRMs can be one-time if there is no storage or replay after the whiteboarding session, otherwise it should be sealed by the TRM along with the policy that deals with replay. If it is one-time, then k_{OBJ} can be sent to the client application by the TRM for better performance. This means that the client application does not have to send the received stream to the TRM for decryption and encryption. However, the attestation challenge of a client application by the TRM is always necessary before a whiteboarding session is allowed.

5.4 SMTP and Trusted Computing

This section describes the application of Trusted Computing to SMTP. There are two interesting issues related to SMTP that can be addressed by Trusted Computing. First, the problem of spam will be discussed and a design for a rate limiter is proposed to reduce the rate at which spam is

generated. Then the secure storage of email messages is investigated.

5.4.1 Spam

Basically, spam is flooding the internet with a large number of copies of the same message, in an attempt to force the message on people who would not otherwise choose to receive it[1]. Normally, spam costs the sender a little to send and therefore most of the costs are paid for by the recipient or the relay hosts rather than the sender. Email spam targets individual users with direct email messages. Receiving spam costs people money: people with a measured phone service read or receive the email while the meter is running and spam costs them additional money. Furthermore, it also costs money for the ISP and online services to transmit spam, and these costs are transmitted directly to the subscribers.

Transmitting spam is often achieved by by misusing relaying hosts. This problem can be partially solved by rejecting specific incoming email at relay hosts. However, the root of the problem lies in people sending spam. One possible solution for this is to use client puzzles that slow down the rate in which emails can be sent on a client platform. Trusted Computing though, enables a new and more efficient method of preventing spam. By limiting the rate at which machines can send email, the rate at which spam email is generated can be reduced. The implementation of a rate limiter was already described in the previous chapter and will now be applied to sending email with SMTP. The approach chosen here is based on [GRB03] and requires isolation of processes by using a TVMM.

On every Trusted Platform a ticket-granting service is running in a closed-box VM. This ticket-granting service releases at most one content dependent ticket every time period. In order to limit the rate at which an email client can send emails there should be an open-box email client VM which tries to obtain a ticket from a ticket-granting VM for every email being sent. More precisely, this means that before sending an email, the email client VM sends a hash of the email to the ticket-granting VM. The resulting ticket will then be attached to the outgoing email. The network of SMTP servers will reject any incoming emails that contain no ticket or an invalid ticket. Figure 5.8 uses Terra's architecture [GPC+03] to visualize the process.

An obvious drawback is that this approach only works if all SMTP servers in the message transit are Trusted Platforms that implement a rate limiter with a ticket-granting service. People who are using a traditional platform can still send spam, although the email messages will be directly rejected

[1] See also http://spam.abuse.net

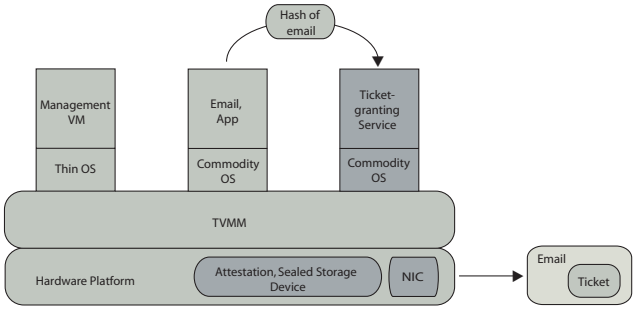


Figure 5.8: Preventing spam by rate limiting.

by the Trusted Platforms. An approach like this one will only have a significant impact if IT companies feel responsible and introduce software that exploits the potential benefits of Trusted Computing. As part of its Trusted Computing initiative, Microsoft has already been busy adjusting Outlook to provide some functionality that is based on Trusted Computing technology[2].

5.4.2 Securing Message Store

As already explained in the previous chapter, current message infrastructure provides no means to verify the integrity and security of server machines, so an adversary may read or modify the content of an email message stored on disk of an SMTP server. Trusted Computing, however, can protect messages by using a mechanism like protected storage. A big advantage of protected storage is that data can be stored in such a way that prevents disclosure of the data if the platform is not in the expected state. In this section the protection of an email message in a platform is further investigated.

SMIME In order to secure email messages SMIME (Secure Multipurpose Internet Mail Extension) is usually used. SMIME is a specification for secure mail and was designed to add security to email messages in MIME format. It provides encryption and signing[3] of email messages to prevent adversaries from gaining access to the content of an email message. However, SMIME only encrypts the body and attachments of an email message and leaves the header unchanged. Therefore SMIME does not prevent adversaries from observing and changing the header information. See also figure 5.9.

SMTP on a Trusted Platform Figure 5.10 shows the design of an SMTP server on a Trusted Platform. At the left an encrypted email message is

[2]See also http://www.pcmag.com

[3]The concept of a digital signature was already explained before. It should be noted however, that they do not guarantee that a message has not been read by another party.

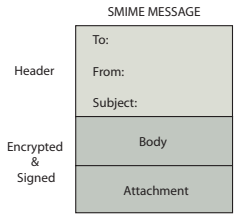


Figure 5.9: An SMIME message.

shown which is about to be transmitted by a trusted client platform. The difference with SMIME is that the email message on the Trusted Platform is completely (including the header) encrypted before it is transmitted into the network. This security measurement directly prevents an adversary from observing and changing the header information while the email message is on the wire.

Before the client decides to trust an SMTP server it verifies the server's trustworthiness by remote attestation. In the example presented here, the client only trusts an SMTP server if its software configuration satisfies the following property: *Sendmail 1.x & 2.x but not 1.1.1.* The SMTP implementation runs in a protected environment and is attested by the Trusted Platform. The protected environment prevents adversaries from reading and changing email messages. At the same time, other components on the Trusted Platform cannot gain access to the email messages either.

If a message does not have to be relayed it is put in a spool area, where all email messages are stored in a separate mailbox for each user. An SMTP server always logs the incoming and outgoing traffic and stores the data in a log file. This log file can be encrypted by protected storage. Protected data can be stored in a way that prevents use of the data unless the platform is in the expected state. Furthermore, protected data can be stored in a way that either *permits* duplication of the data or *prevents* duplication of the data. This implies that a log file cannot be read by an adversary and can only be copied if this is allowed by the owner of the object (the administrator).

Although an adversary cannot read or change the message on the wire and cannot violate the protected environment of the specific SMTP implementation, there is yet another way of gaining access. For example, in Sendmail, the administrator has full access to the configuration management system[4]. The functionality provided by this system is mostly related to changing headers of email messages. Furthermore, the administrator also has access

[4]See also http://www.sendmail.org

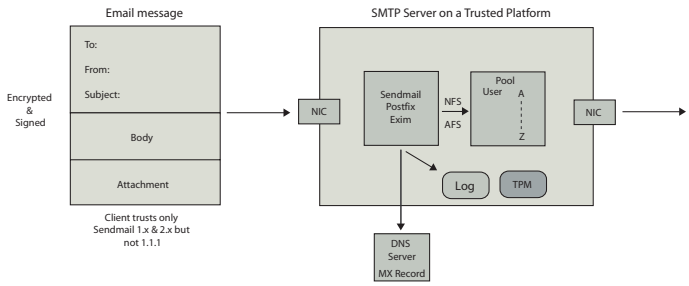


Figure 5.10: SMTP on a Trusted Platform.

to the log file which lists all processed emails. This implies that the solution provided by Trusted Computing only protects adversaries from reading and changing emails if they do not have admin rights to the system. However, a persistent hacker may setup his own Trusted Platform and install an SMTP implementation. If he assigns admin rights to himself, he will be able to do whatever he wants to. Nonetheless, remote attestation can always assure a remote party of the software configuration running on the attested platform.

Relaying Email In email, the store-and-forward principle is used to transmit an email message from the sender to the recipient through some intermediate hosts. The routing of an email message is achieved by looking at the MX records of a DNS server. As already explained before, not all SMTP servers are trusted and therefore cannot be used in the path from the sender to the recipient. Normally a DNS server tells an SMTP server that it should relay a message to a particular SMTP server in order to obtain the shortest route. However, if due to remote attestation an SMTP server does not turn out to be trustworthy, then another route should be taken.

In this particular example the client only trusts an SMTP server if the software configuration satisfies the following property: *Sendmail 1.x & 2.x but not 1.1.1.* The DNS server that resides in the USA could not forward the email message to an appropriate SMTP server and therefore another DNS server should be queried. This implies there are two routing criteria: (1) What is the fastest route? and (2) Is the route secure? An obvious disadvantage of this approach is that expensive lines like the Atlantic line between Europe and the USA may be used without any successful result. See also figure 5.11.

Sendmail and BIND In chapter 2 a fine-grained attestation service for distributed systems was discussed. In order to be effective it assumes that a programmer identifies the relevant process for attestation and annotates the beginning and end of the critical process with an ATTESTATION_INIT and

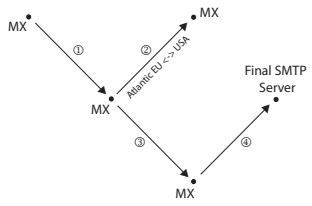


Figure 5.11: Efficiency problem in routing.

ATTESTATION_COMPLETE call [SPvD05]. So every time the process executes, BIND is invoked and will attest the code to its integrity. However, for example in the case of Sendmail[5], most of the program code is written to perform changes to a header of an email message. This implies that if one would like to apply BIND, a lot of code has to be inserted which makes this approach quite unpractical. Therefore BIND does not offer a satisfactory solution and is only effective if relatively few pieces of code have to be attested.

Further Refinement In contrast to SMIME, the approach chosen here does protect from adversaries on the wire as all email messages are completely (including the header) encrypted. However, from a theoretical point of view, a sophisticated hacker can monitor the network traffic to investigate where all the packages are going, without having any knowledge about the content of the packages. By looking at delays and the size of a package before and after it arrives and leaves an SMTP server, a hacker may guess the information that is sent. A way to make this a lot more difficult, a message could be decrypted en encrypted again, so that the output message looks different. This decryption and encryption process must be performed in curtained memory so that no one can observe the unencrypted message. Of course the protected storage of many keys have to be taken into account, but that is something for future research. A rate limiter, which was already described before, can be used to change the delays between the messages so that a hacker cannot analyze delays and draw valid conclusions.

5.5 Cookies and Trusted Computing

Cookies are a mechanism which can be used by server side connections (like CGI scripts) to both store and retrieve information on the client side of the connection. The capabilities of web-based client-server applications can be significantly extended by using a simple and persistent client-side state. Cookies are often used to store information which the user has supplied in

[5]See also http://www.sendmail.org

an earlier stage. A typical example is the shopping list which might be updated from time to time.

It should be noted that cookies cannot be used to steal information from a computer system. Cookies can only be used to store information that a user has provided in the past. Unfortunately, cookies can also be used for more controversial practises. Each time a user accesses a web site with its browser, information is left behind on the computer. Among the data that is left behind are the name and IP address of the computer, the brand of the browser being used, the OS that was running and the URL of the web page that was accessed. Without cookies it is practically impossible to learn about a person's web browsing habits: one would have to reconstruct the person's path by relating hundreds or even thousands of individual server logs. The DoubleClick Network exploits this feature of cookies to build up a database of user profiles and to present advertising banners that are customized to the user's interests.

Besides privacy issues, cookies also have certain security implications [SS]. There are many web sites that make use of cookies to implement a form of access control. A web site may send a cookie back to the client's browser the first time the client user logs in. The site will only provide access to restricted pages if the browser can show a valid cookie. Although this approach has some advantages, the system may be vulnerable to attacks by a malicious party. For example, an eavesdropper may intercept the cookie when it is sent from the browser to the server, so that it can have free access to the web site. Because browsers make use of DNS to decide which cookies belong to a server, it is possible that a browser is tricked into sending a cookie to a false server if an adversary temporarily subverts the DNS. If a cookie is persistent then there is also a chance that it will be stolen from the user's cookie database file.

When designing a system that uses cookies for authentication and state-preservation the possibility of cookie interception should always be taken into account. It is therefore unappropriate for cookies to contain usernames and password in plain text. In order to gain a higher level of security cookies include the following information:

- The session ID or authorization information.
- The time and data the cookie was issued.
- An expiration time. By adding an expiration data and time, system designers are enabled to limit the potential damage a hijacked cookie can do. If a cookie is intercepted it can only be harmful for a finite amount of time.

- The IP address of the browser the cookie was issued to. The cookie will only be accepted by the server if the IP address in the cookie is similar to the IP address of the browser that submits the cookie.
- A message authentication check (MAC) code. The MAC code ensures that none of the fields of the cookie have been changed. The hash value of the whole cookie is incorporated into the cookie data. So when the cookie is sent to the server, the server's software can check if the cookie has not expired and is being returned by the proper IP address.

The most secure method is to encrypt the entire cookie with an encryption algorithm or to encrypt the channel between the browser and server, using SSL. In order to avoid that a cookie is disclosed in a non-secure channel, the security attribute should be set so that the browser only sends the cookie if SSL is turned on.

Trusted Computing can provide the necessary mechanisms to enhance the security of cookies. In this chapter the concept of protected storage has already been discussed in quite some detail. Protected data can be stored in a way that prevents use of the data unless the platform is in the expected state. Furthermore, protected data can be stored in a way that either *permits* duplication of the data or *prevents* duplication of the data. This implies that the IP address field of the cookie becomes obsolete, because protected storage prevents the disclosure of the cookie if the protected object is accessed by another platform then specified in the protected object.

A MAC code is used to perform some integrity checks by verifying the self-consistency of data. It happens, however, that protected storage already has an implicit consistency check, which was already described in chapter 2. So no explicit integrity check like a MAC code is required to verify if unappropriate alterations have been made to the protected data. There is no need to include an expiration data either because a feature like this is also supported by protected storage.

Cross Site Scripting (CSS or XSS) is sometimes used to steal cookies from their corresponding users[6]. Cross site scripting happens when a user is tricked into sending information to a malicious web site through the use of scripts. The users clicks on links that appear to be legitimate and the results that are returned to the user also appear to be legitimate. The attacker uses HEX and other methods of encoding in order to make the links and http addresses look less suspicious. There are many guestbook and forum programs that allow a user to post data that contains javascript which can

[6]See also http://www.cert.org

be used to embed malicious code in the web site. The malicious code will be executed at the moment people read the forum post. Most people will never notice when their computer is running malicious code. As long as the web site looks real enough users will most likely not notice anything.

Unfortunately, this security problem cannot be solved by just encrypting the cookie. In most cases, cross site scripting requires that some form of embedded scripting language is enabled in the victim's browser. By disabling all scripting languages the problem is mostly solved, but has the side effect for many users of disabling functionality that is important to them. However, attackers may still able to manipulate the appearance of the content provided by the legitimate site by just using other HTML tags in the URL. Especially malicious use of the <FORM> tag is not prevented by disabling scripting languages.

Protected storage offers the extra advantage of sealing data to a platform state. Sealing a cookie to a particular web site may prevent an XSS attack. If an adversary duplicates the content of a web page and inserts some cookie-stealing code, then the disclosure of the cookie will be prevented. However, the sealing of a cookie to the details of the web page would be problematic for updates to the web site. The specific platform state that reduces the risk of a cross site scripting attack is the state where all scripting languages are disabled. This state should be included in the protected object, so that the cookie will only be disclosed and transmitted to another party (preferably the legitimate server) if no scripting language is enabled. This prevents most of the cross site scripting attacks. As already mentioned before, attacks can also take place if the <FORM> tag is used in a malicious way. Nonetheless, protected storage can significantly reduce the risk of being attacked by cross site scripting. The inevitable drawback is that important functionality may be disabled.

Chapter 6

Conclusion and Further Research

The aim of this research project is to investigate the benefits and drawbacks of using Trusted Computing in the implementation of end-to-end application security. This final chapter summarizes the performed research and tries to formulate an answer to the problem definition stated before. At the same time some topics for further research may be proposed. To simplify matters an overview will be presented per chapter to emphasize what knowledge has been gained from literature and what contribution has been made to the field.

- **Chapter 2** provided the reader with information about Trusted Platforms and Trusted Computing technology in order to create a basis of understanding for the rest of the thesis. Some applications of Trusted Platforms were brought together and a few relevant properties of Trusted Computing technology were described in detail. During this preliminary investigation, information had to be reconsidered quite regularly as there are a number of papers that contradict one another in matters that are quite fundamental: not all papers explain the process of attestation in the right way or do not fully understand its underlying mechanisms, for example. Occasionally, comments on the TCG specifications were given as there are still various shortcomings that should be addressed.

  There exist many papers in which possible applications of Trusted Computing are proposed. It is suggested that Trusted Computing will enable a large variety of new applications and has the potential to revolutionize the world. However, most literature focuses on low-level facilities and what kind of applications they will support. The details of the design of applications and end-to-end security are usually

quite sketchy. The research fills in some of the gaps and leads to a better understanding of how Trusted Computing can be used in the implementation of end-to-end application security.

- **Chapter 3** presented an introduction to application layer protocols and described their fundamental properties from the perspective of end-to-end communication between applications. Some of the involved security issues were discussed and briefly addressed by Trusted Computing technology. Then an abstraction to application layer protocols was proposed by providing a classification that tries to capture a large part of the full spectrum of protocols. The classification was obtained by a careful and thorough investigation of how application layer protocols play a role in the sharing and dissemination of data. The underlying mechanisms used by the protocols to achieve end-to-end communication are not of importance to the scope of the research. This particular approach is completely in agreement with the OSI model. Furthermore, it enables an interesting perspective on security issues, without focusing on the individual application layer protocols themselves. The nature of the classification ensures that it will be valid for a long time, even when protocols have undergone changes or when new ones are invented.

- **Chapter 4** described the security requirements for the classes contained in the classification. It also provided some brief discussion on how Trusted Computing can be used to address these security requirements.

  In the message connection class, a general solution to denial of service attacks was offered by rate limiting the amount of messages that can be sent from one host to another. An obvious drawback is the explicit need for Trusted Platforms that implement such a rate limiter. Another security issue in this class is that current message infrastructure does not provide any means to verify the integrity and security of server machines, so an adversary may read or modify the content of a message stored on disk. Protected storage can prevent an adversary from reading the protected content. Trusted Computing can also enhance and support digital signatures. A couple of security issues involved in using digital signatures were listed and addressed. However, the risk of a man-in-the-middle attack still has to be addressed and is left for future research.

  In the shared data repository connection class, client authentication can be improved by changing standard client platforms into Trusted Platforms. A server can then be sure that a client is who it says it is,

regardless of the origin of the connection. Furthermore, the server can be sure that it is talking to a Trusted Platform. Trusted Computing can also enhance access control in ways that have not been possible before, by providing a safe way of storing the encryption keys. An extra advantage is that the keys can be sealed to a particular operating system or platform configuration. The issue of web security should be further examined as the enhancements of web applications on a Trusted Platform will have a direct impact on e-business. It is not for nothing that the demand for more security in e-business is one of the major reasons that led to the Trusted Computing Group. Sometimes, it is necessary to restrict queries to prevent a malicious party from downloading an entire database. Again a rate limiter can be used to limit the amount of queries that are sent to the database.

In the conferencing connection class all kind of security policies can be enforced on a Trusted Platform. There are already existing Trusted Computing technologies like LT (LaGrande Technology) that enable curtained memory space, process isolation and secure channels between processes. LT can also provide a trusted path between an application and audio, video and I/O channels. For example, the channel from an application to an audio card and driver can be protected so that no other process can read or change the content.

- **Chapter 5** investigated the possibility of applying Trusted Computing to the architecture of some distributed application components. Four candidates were scrutinized: FTP, whiteboarding, SMTP, and web cookies.

  An enhanced access control mechanism for FTP was proposed by using an already published access control architecture. The control over the dissemination of data is a fundamental problem in access control. However, due to the mechanisms provided by Trusted Computing, an FTP server is able to trust the client with respect to both the platform and the authentication. A considerable improvement is gained here, because the object owner knows for sure that the security policies are enforced on all platforms the object is distributed to. The presence of a TRM on a Trusted Platform is a prerequisite however.

  The same access control architecture was used to provide end-to-end security of a whiteboarding application. A drawback is that the client application has to send the received stream to the TRM for decryption and encryption. Of course, this may be quite CPU intensive. If possible, some improvements should be made to reduce the CPU load as much as possible. This is left for future research.

In order to prevent spam, a rate limiter was described by using a virtual machine-based platform for Trusted Computing. The concept of a rate limiter raises all kind of issues. For example, all SMTP implementations should run on a Trusted Platform that implements a rate limiter. Furthermore, it should be examined what kind of rate would be appropriate.

The secure storage of email messages is an important condition for the transfer of responsibility. SMIME does not encrypt the headers of an email message as the header provides network devices with important information on how an email message should be routed through a network. Furthermore, the message header can always be changed at an SMTP server and it is impossible to find out if the administrator of the SMTP server can be trusted. Attestation however, does provide some security: it can check if certain options are disabled and therefore cannot be accessed by the administrator. On a Trusted Platform email messages can be completely encrypted (and decrypted) so that an adversary cannot observe the header information while the email message is on the wire. All SMTP implementations should run on a Trusted Platform: only trusted SMTP servers will be used in the message transit. This may result in inefficient bandwidth usage. The discussion provided in chapter 5 clearly shows that the design and implementation of an application that uses Trusted Computing technology should not be underestimated. There are still many issues which have to be dealt with in order to create a satisfactory design.

At the end, the privacy and security of web cookies were investigated and some brief recommendations of how protected storage may enhance security were given. The sealing of a cookie to the details of a web page would be problematic for updates to the web site. Another solution is the sealing of a cookie to the state in which all scripting languages are disabled. However, this would imply that important functionality may not be present. The discussion provided in chapter 5 was quite general in character and better solutions should be searched for in the future.

People have just begun to explore the broad range of possible applications that can be realized by Trusted Computing. Nonetheless, the design and implementation should not be underestimated. As already stated before, most papers only focus on low-level facilities and the designs of the applications are often very sketchy. This thesis, however, has filled in some of the gaps and investigated the possibilities of implementing Trusted Computing in

end-to-end application security. Instead of looking at low-level facilities and vague designs, the research has given a fresh and new perspective on Trusted Computing at the application layer. Hopefully, this thesis will inspire more people to further investigate the potential of Trusted Computing.

Bibliography

[And03] Ross Anderson. Cryptography and competition policy: issues with 'trusted computing'. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 3–10, New York, NY, USA, 2003. ACM Press.

[Arb02] William A. Arbaugh. Improving the tcpa specification. *IEEE Computer*, 35(8):77–79, 2002.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 139–147, London, UK, 1993. Springer-Verlag.

[ELM+03a] Paul England, Butler Lampson, John Manferdelli, Marcus Peinado, and Bryan Willman. A trusted open platform. *Computer*, 36(7):55–62, 2003.

[ELM+03b] Paul England, Butler Lampson, John Manferdelli, Marcus Peinado, and Bryan Willman. A trusted open platform. *Computer*, 36(7):55–62, 2003.

[Goh03] Chiew Pheng Goh. A security model for a defence-related organization. Master's thesis, University of Wales, 2003.

[GPC+03] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th Symposium on Operating System Principles (SOSP 2003)*, October 2003.

[GRB03] Tal Garfinkel, Mendel Rosenblum, and Dan Boneh. Flexible os support and applications for trusted computing. In *Proceeding of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, pages 145–150, May 2003.

[HCF04] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation — a virtual machine directed approach to trusted computing. In *VM'04*. USENIX, 2004.

[HvD04] James Hendricks and Leendert van Doorn. Secure bootstrap is not enough: Shoring up the trusted computing base. In *Proceedings of the Eleventh SIGOPS European Workshop*, Leuven, Belgium, Sept 2004.

[IKBS00] Sotiris Ioannidis, Angelos D. Keromytis, Steven M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *ACM Conference on Computer and Communications Security*, pages 190–199, 2000.

[Pea03] Siani Pearson, editor. *Trusted Computing Platforms: TCPA Technology in Context*. Hewlett-Packard Professional Books. Prentice Hall, 2003.

[RCR04] Paul D. Robertson, Matt Curtin, and Marcus J. Ranum. Internet firewalls: Frequently asked questions, 2004.

[SCG+03] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17 Int'l Conference on Supercomputing*, pages 160–171, June 2003.

[SJZvD04] Reiner Sailer, Trent Jaeger, Xiaolan Zhang, and Leendert van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 308–317. ACM Press, 2004.

[SKv03] David Safford, Jeff Kravitz, and Leendert van Doorn. Take control of TCPA. *Linux Journal*, August 2003.

[Smi05] Sean W. Smith, editor. *Trusted Computing Platforms, Design and Applications*. Springer, 2005.

[SPvD05] E. Shi, A. Perrig, and L. van Doorn. BIND: A fine-grained attestation service for secure distributed systems. In *Proceedings of the IEEE Security & Privacy Conference*, Oakland, CA, May 2005. IEEE Press.

[SRC84] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[SS] Lincoln Stein and John Stewart. The world wide web security faq. http://www.w3.org.

[SZ05] Ravi Sandhu and Xinwen Zhang. Peer-to-peer access control architecture using trusted computing technology. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control*

models and technologies, pages 147–158, New York, NY, USA, 2005. ACM Press.