Technische Universität München
Lehrstuhl für Datenverarbeitung

Data Protection and Risk Management on Personal Computer Systems Using the Trusted Platform Module

Oliver Welter

Vollständiger Abdruck der von der
Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität München
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs
genehmigten Dissertation.

Die Dissertation wurde am 21.02.2008 bei der Technischen Universität eingereicht und
durch die Fakultät für Elektrotechnik und Informationstechnik am 05.03.2008 angenommen.

Data Protection and Risk Management on Personal Computer Systems Using the Trusted Platform Module

Oliver Welter

Danksagung

An dieser Stelle möchte ich mich bei allen Freunden und Kollegen bedanken, die direkt oder indirekt am Gelingen dieser Dissertation beteiligt waren.

Mein ganz besonderer Dank gilt meiner Lebensgefährtin Tanja, die immer an meiner Seite stand und half, so manchen Tiefpunkt zu überwinden. Bedanken möchte ich mich zudem bei meinen Kollegen Walter Bamberger, Marko Durkovic und Florian Obermeier für die vielen wertvollen Diskussionen, Ideen und investierten Stunden, Anke Bartlau für die Unterstützung bei sprachlichen Fragen und meinem Doktorvater Prof. Klaus Diepold für die hervorragende Betreuung sowie die Gelegenheit, an seinem Lehrstuhl zu promovieren. Nicht unerwähnt lassen möchte ich hier den inzwischen leider verstorbenen, ehemaligen Lehrstuhlinhaber Prof. Joachim Swoboda, dessen Wirken ein geeignetes Umfeld für diese Arbeit geschaffen hat sowie Lieselotte Heckmann, die mich an den Lehrstuhl brachte und bei organisatorischen Fragen immer hilfsbereit zur Seite stand. Meinem Zweitprüfer Prof. Uwe Baumgarten danke ich insbesondere für die tiefgehenden Diskussionen und die immer sehr zeitnahe Reaktion bei fachlichen sowie formalen Dingen.

Nicht direkt an der Arbeit beteiligt, aber als Wegbereiter für Abitur und Diplom unverzichtbar waren für mich meine Schul- und Studienkollegen Florian Socher und Christian Hergl sowie die Lehrer meines Abiturjahrgangs Helga Eham, Helmut Schenk und Manfred Dudda, denen ich an dieser Stelle meinen besonderen Dank für ihre Unterstützung aussprechen möchte.

If I take a letter, lock it in a safe, hide the safe somewhere in New York, and then tell you to read the letter, that's not security. That's obscurity. On the other hand, if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the world's best safecrackers can study the locking mechanism–and you still can't open the safe and read the letter, that's security.
Bruce Schneier, Preface of "Applied Cryptography" [46]

Contents

Preface

Personal computer systems are used in almost every area of our daily life to create, manage, process and consume all kinds of information. To safeguard that this information is used according to their stakeholders interests, data protection methods are applied. The majority of such protection systems finally fail to provide a provable security system as they either rely on obscurity of components or blindly trust the underlying systems to behave fair. The concepts around the Trusted Platform Module (TPM) solve both issues but fail to integrate into a landscape where discovery and exploitation of vulnerabilities makes trust in a software configuration a transient value.

In this thesis, a combination of methods to overcome those deficiency is presented and used to build a protection system which is solely based on provable cryptographic components and adopts to today's software lifecycle. Digital signatures bridge the gap between a formal policy model and the measurement values necessary to technically drive the TPM while a certificate infrastructure provides the ability to react on vulnerabilities. A multilevel security model segments the assets into distinct protection domains which are protected by TPM based keys and can be individually shutdown based on the severity of current threats. The verification system is strictly isolated from the main operating system and under continuous supervision of the TPM, so it can safely survive a compromise of the platform and reestablish trust after the system was repaired. Besides the protection of data on the local platform the new digest creation method eliminates several drawbacks of the remote attestation protocol, especially regarding the privacy of the challenged user.

Compared to current solutions, our system provides several advantages. First, the protection mechanism is based on a tamperproof hardware component and its security is provable by formal means. Second, the impact of attacks based on known vulnerabilities can be estimated and critical data moved out of their grasp while keeping less valuable data available for business. And finally, even if an attacker manages to exploit a vulnerability, there are several barriers hardening his goings and a verification system that reliably detects each modification to the system at latest on reboot.

Chapter 1

Data Protection on PC-Systems

1.1 Why Data Protection on PC-Systems is Crucial

Personal computer systems are widely used by individuals and business people to simplify and organise processes of the daily life. These processes rely on input data and produce new data which is stored on the computer systems. To perform correctly, the processes require that the stored data is not tampered in an unauthorized way. Besides, the data contains information that is intended only for a particular audience and releasing it to a third party can cause a multitude of problems.

In the early days of computer science, everybody was assumed to behave fair and the protection mechanisms focused on technical errors or accidental mistakes of the users. This assumption does not hold today. On the one hand, the relation that users have to the data has changed and we can no longer trust the users to use data only for the intended purpose. On the other hand, computer systems are getting more and more complex and are considered to have unknown vulnerabilities while their connection to the internet moves them into the grasp of criminals worldwide, creating a new kind of offenders. Those people can have different motivations and targets, but usually strive for a direct or indirect personal advantage which almost always implies a damage or disadvantage for another party.

Incorrect process information, lost confidentiality or other negative effects cause damage and must be avoided to ensure business continuity and to protect the privacy of involved individuals and entities. This can be accomplished by the deployment of data protection methods within the systems. Which methods are sensible to apply, depends on the protection targets and the resulting threat scenarios, which we discuss by means of an example in the following section.

1.2 Different Aspects of Data Protection

Data Protection covers a large field of situations, roles and aspects which we will explain with an example. Bob[1] drives an online business in creating expertises on demand of individual customers and uses an online payment system to get paid for his work. Alice is a returning customer of this service and stores the access credentials for her account as well as the bought expertises on her computer system. Due to the nature of the data in the reports, she wants to keep them confidential and prefers to stay anonymous against Bob. To protect her anonymity, they use the payment service of Trent, who issues non-trackable vouchers and bills them via a credit card.

Workflows and Protection Properties of the Given Example

Bob offers his services for cash in advance, therefore the first step for Alice is to pay Bob for the desired service. To obtain payment vouchers from Trent, she submits her credit card data to him and receives payment vouchers. Alice now passes the input data for the requested report to Bob and adds one of the payment vouchers to pay Bob. Bob uses the voucher to collect his earnings from Trent and compiles the requested report, which he then makes available to Alice in a protected area on his website.

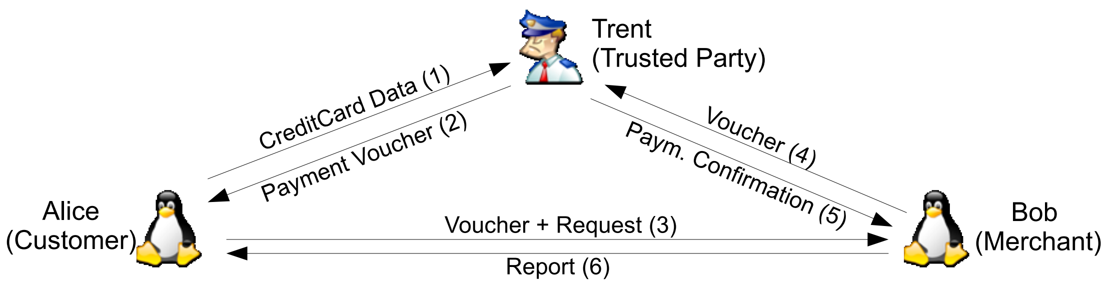

Fig. 1.1: Flow of Information between Alice, Bob and Trent

The three parties involved in the example exchange information as shown in figure 1.1 and it is mandatory, that these connections fulfill certain properties to ensure a satisfying result for all of them.

Confidentiality ensures that information exchanged between communication partners is not disclosed to a third party. The connections from Alice to Bob and Trent must ensure confidentiality, as otherwise payment or personal information gets disclosed. The connection between Trent and Bob can reveal information about Bobs earnings, so it might be of interest to enforce

[1]Alice, Bob and Trent are names which are widely used in cryptographic literature and carry implicit properties. Alice and Bob are two communication partners while Trent represents a neutral and trusted party.

confidentiality here. It is also mandatory to protect all data items against offenders while they are stored on the local systems of the participants.

Authenticity proves the origin of a message and is also used to identify the communication partners in this example. Alice must ensure that she talks to the right communication partners as an eavesdropper might abuse the submitted data directly or launch a man-in-the-middle attack which will affect the confidentiality of the communication.

Integrity of an item indicates that it was not changed between two supervisions. A successful completion of the workflow requires the correctness of all process information and all information flows and therefore depends on an integrity protection mechanism.

An exhaustive summary of common cryptographic methods to ensure the mentioned properties is given, for example, in [9, 46].

1.3 Protection Systems Currently in Use

In this section, we talk solely about systems, which provide protection of items outside a controlled system environment. Protective mechanisms used inside running systems, as process separation and the like, are not included in this discussion.

1.3.1 Technical Foundations

Today's protection systems can be grouped into three major categories. Systems providing protection using obfuscation, such using cryptographic algorithms and such using a combination of both. We give a detailed view on these techniques and their special problems in section 2.1.

Protection by Obscurity

All protection systems that must work without a user provided password use obfuscation to protect themselves. For example the keystores used in Microsoft Windows operating systems to sign and encrypt items are protected by the secrecy of their internal structure. Another product which came to dubious fame in the past, is the internet telephony client “Skype” which uses obfuscation even to encrypt its own program code to protect it from reverse engineering [39].

Protection by Cryptography

Systems relying solely on pure cryptography and the secrecy of the needed key material are, for example, several available disk-encryption tools for the Linux operating system like dm-crypt [44] or eCryptfs [29]. As these are open source projects, it is obvious that there is no place for obscurity here.

The disadvantage compared against the systems using obscurity, is the necessity to find a secure way to load the key into the system. The usage of a hardware token with a cryptographic engine and secure key storage solves this problem but leaves the issue how to protect the complete cryptographic engine from abuse as the external token can not detect by which instance it is called.

Protection by a Combination of Both

The usage and development of systems using only obfuscation is very limited as it is necessary to disclose secret information to potential business partners. Therefore it seems to be a feasible approach to use public, proven algorithms to do the protection and use obfuscation only to provide the necessary key material in an automated way.

1.3.2 System Immanent Deficiencies

As long as we have a friendly user and a system that was setup with security in mind, all off the above mentioned systems will do there job equally well but there are two kinds of problems we have to deal with. Effects on the protection systems by a compromise of the operating system through a vulnerability are discussed later in 1.5.2. The other problem arises from fraudulent user, which try to avoid or circumvent protective mechanisms.

It is easy to judge on pure cryptographic systems. Toady's algorithms are mature and do not offer any feasible approach to achieve a result without the correct initialisation values, namely the key material. The other way round, a user with access to the key material can reproduce valid results with an arbitrary implementation of the algorithms, regardless of any context. The main problem of such a system is the question how to provide the key to the algorithms and how to protect it from abuse by the different kinds of attackers.

Systems using obfuscation are autonomous, which means they carry all necessary information to generate the reply on a certain input inside the system. To emulate the behaviour of such an obfuscated protection system it is necessary to understand the algorithms running inside and know all parameters of them. If access to enough input/output pairs is given and the mechanism is simple or partially known, a mathematical analysis can reveal enough information to mimic the system. In general, the only way to reveal the internal structure is a stepwise reverse engineering [53] of the deployed binary code which might be expensive but can never be prevented.

1.4 TPM - One Major Concept of Trusted Computing

1.4.1 Key Functions of the Trusted Platform Module

The Trusted Platform Module (TPM) is a new hardware component which resides on a computers mainboard and is the central piece in the concepts of the Trusted Computing Group (TCG). It consists of three major parts. A cryptographic engine supporting the RSA operations, a shielded storage for key material and the so called platform configuration registers, PCR, which are basically a storage for hash values with a special input function. A detailed description of the TPM and its constituents is given in 2.3 so we just briefly explain some important issues here.

The basic functionality of the cryptographic engine of the TPM is similar to the one on conventional smartcards. All key material is hold and processed in a shielded environment and therefore secret information is never disclosed to the outside. The power of the TPM compared to a common smartcard is the combination of this engine with the platform configuration registers. When using a TCG conformable platform, these registers contain a fingerprint of the code used to bring up the platform. As the registers can be initialised only on a hardware reset and always reflect the history of all input values starting from the last reset, it is impossible to force them to a special value after the platform was started. This leads to the assumption, that a given system state is represented by a combination of register values and that it is impossible to find another system state resulting in the same values.

The first of two groups of key functions specified by the TCG now uses the values of the PCR registers as an access credential to key operations which binds the key, and with it all data encrypted under it, to a defined system state.

The second key function is the attestation of the platform state against a challenger. The basic idea is simple while its implementation is one of the most complex protocols in the whole TCG concept. Roughly speaking, a challenger asks the platform to identify itself and receives the current state of relevant PCR values signed with a special key that can be tracked to belong to a genuine TPM by a set of certificates. If the challenger successfully verifies the signature and the certificate chain, he can be sure that the received data was created by a TPM. The submitted values represent the platform state and if he knows to interpret these values he can be sure about the hardware and software running on the challenged system.

1.4.2 Unsolved Problems

If we neglect hardware based attacks and take for granted that no unknown attacks on the used algorithms, RSA and SHA1, exist, a cryptographic system build around the TPM, can be seen as a mature solution.

The problems arise from the variety of software on the market, the necessity for interoperability and the lifecycle of toady's software. In the current specification, the values inside the PCR registers depend directly on the binary code used to launch the system and it is not possible to assign a set of multiple combinations as access credential to an object. This makes it impossible to directly use TPM protected keys when they must be available in more than one system state. Furthermore a similar problem arises if we want to include later boot stages into the measurement values. As modern systems often use a multi-threaded initialisation procedure, where multiple processes run in parallel and the execution order is not fully deterministic, we will end up with different measurement results on individual boot cycles. But even if we omit multi-boot scenarios and linearize the boot process, we still stick to the problem of vulnerability management as explained in 1.5.2.

Another issue regarding the attestation protocols is not only technology related but also driven by privacy concerns and political discrimination. The response send upon an attestation request must contain sufficient information to allow the challenger a judgement on the state of the platform. With the current solutions this implies, that we reveal the name and exact version of each component which is part of the measured hardware and software stack. Besides the technical difficulty to maintain a huge database of all possible components and the ability to derive a judgement from this knowledge, the challenger can abuse this information to the disadvantage of the user. User organisations, like the Electronic Frontier Foundation [15], as well as respected researchers [42] worry about the possibility that companies lock out particular platforms from their business due to commercial interests. Criminals can also benefit from the attestation, as they gain detailed knowledge about the platform which they can use to create tailored and most effective attacks for the target system. Finally, due to the huge variety of combinations, it is likely possible to link together independent transactions of a single user by watching the received attestation results. Such a tracking affects a users privacy as it reveals behavioural information where it is not recognized and also not avoidable.

1.5 Today's Software-Lifecycle and Problems Arising from It

As we explained earlier in this section, purely software-based protection systems can never be secure as they must use some kind of obscurity to protect themselves. But besides this immanent deficiencies there are some issues that are founded in the lifecycle of toady's software systems, which we want to discuss here.

1.5.1 A Lifecycle-Model for Computer Systems

"Any given piece of software has some number of publicly disclosed vulnerabilities at any moment, leaving the system exposed to potential attack." [32]

Even if we cannot formally prove this statement, it matches the experience of people working in the field of software development and security. Based on their vulnerability lifecycle model, the authors of [1] propose a lifecycle model for computer systems as shown in figure 1.2. According to it, a computer system repeatedly toggles between the states `hardened` and `vulnerable` during its entire lifetime and might pass into the state `compromised` in the case of a successful attack.

Hardened State

While the system is in hardened state, it is not affected by a known vulnerability and considered to operate as expected. Based on the vulnerability life-cycle model of Arbaugh et al. [1] it is an unsolvable question if the system is really in hardened state. Prior the public announcement of a vulnerability, the individual who discovered it might abuse this knowledge to launch attacks on affected systems without informing a larger audience and misleading an external observer to the assumption that the system is in hardened state while it is not.

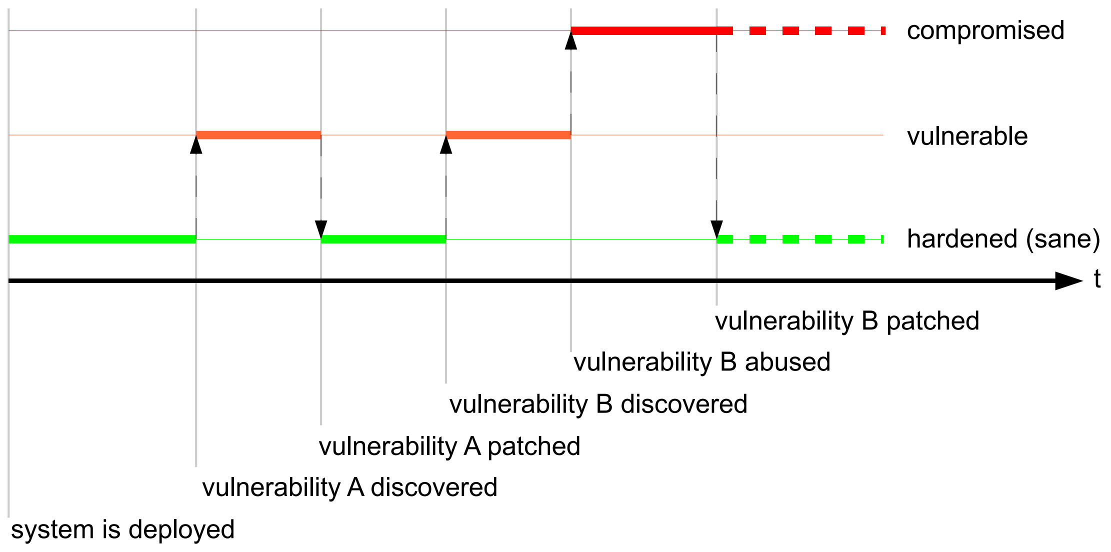

Fig. 1.2: State-Transitions of a System during its lifetime

Vulnerable State

Once the problem was disclosed to the public audience we know that the system is vulnerable, but we do not necessarily known at the time if an attacker has already compromised the system. Besides, there is the question on effective and feasible countermeasures to prevent an attacker from compromising the system while the vendor prepares the final fix for the issue.

Compromised State

By a successful abuse of a vulnerability an attacker gains unauthorized access to system resources or the system itself. Depending on the kind of vulnerability and the abilities of the intruder he can manage to hide the traces of the manipulation while preserving control over important system components [19]. This circumstance brings the system owner into the dilemma that he can not reliably determine a compromise without an external reference as he can not trust the system to tell him the truth.

This problem stays effective even after a final fix for the initial problem is available. Attackers frequently abuse the initial problem to install their own access tools with the purpose to keep control over the system upright after the problem was fixed. In this case, the removal of the original vulnerability prevents a new compromise but does not remove the already existing modifications.

1.5.2 Influences on the Effectiveness of Technical Protection Systems

The technical systems to ensure the protection properties given in 1.2 usually work as a component of the running operating system or at least rely on some assurances given by it. If an attacker has compromised the underlying system he is likely in the position to interrupt data streams from and to the protection systems and read and manipulate them at will. This will include the manipulation of measurement values, as for example integrity hashes of binaries, access to confidential data or even access to key material of the used cryptographic algorithms.

Needless to say, that such interference renders the protection system pretty useless regarding all activities of the local attacker. As the attacker can forge any information about the state of the system itself, it is impossible for a local application to judge on the sanity of the system. This is also true in a communication scenario with a remote partner, but we have a chance to improve the situation here using an external observer who attests the correctness of the measurements.

Shortly summarized, a compromised system is unable to provide a resilient foundation for the protection systems and therefore they fail to fulfill their task. As it is possible for a skilled attacker to avoid the detection of a compromise, we can never be sure to communicate with an intact system without the presence of an external attestor.

1.5.3 Lifecycle Related Problems when Using the Trusted Platform Module

The fashion how the platform state is determined and linked to the access control results in an incompatibility with the described lifecycle model. While the lifecycle model requires that certain items of the system are exchanged with successive versions due to disclosed vulnerabilities the TCG concept links the access decision directly to the hash value of a binary. The concept does not foresee a migration path for these dependencies, and therefore even an intentional manipulation of a system file blocks access to the associated key material.

The only possibility to perform a system update is very expensive and imposes several security risks. For certain types of keys the specification defines a method to export them to another platform but as the exporting TPM can not prove the identity or state of the offered target it is possible to export them to an unprotected system. For non-migratable keys we have no other option than decrypting each item protected under this key and re-encrypt it with a fresh key after the migration.

From a security point of view, one must assume that the known vulnerability was already exploited and the system was modified. An attacker already present on the system can manipulate the export routines to intercept the data flows and extract keys and data in unencrypted format. If it is infeasible to prove the absence of any modifications, there is no way to securely migrate the data from the affected to an updated system state which leaves us with the decision between terminating or updating the system immediately and loosing the data, or migrating the data and accepting the risk that an attacker might gather or even tamper the data while it is migrated. In most cases it should be possible to lock out an attacker by disconnecting the system from a network, which protects at least the confidentiality of the data but in either case, the efforts necessary to transfer and re-encrypt all data items make this an impractical solution.

Even if we solve the problem of a secure migration to a new system configuration, we still stick with another issue. The key material which is managed by the TPM is not kept in the hardware storage but swapped out to the hard disk where it is accessible for a user with sufficient access rights. An attacker can therefore make a copy of it and recreate old keys from a backup even after they were deleted during an update. Together with a copy of protected items and binaries taken before the update, he can recreate a system which is declared as valid by the TPM and has access to all key material although it contains known and exploitable vulnerabilities.

1.6 Organisation of this Work

1.6.1 Objectives of our Work

In the former sections of this chapter we have shown why we think data protection is important and that the solutions currently in use have several deficiencies. While the concepts defined by the TCG have the power to solve the methodical problems of software based cryptographic solutions, they fail to adopt to toady's software lifecycle and require huge efforts if protected objects must remain usable under different software configurations.

The main objective of the work presented in this thesis is a protection mechanism, that is provable secure, has no methodical deficiencies and works with toady's software and hardware landscape. The mechanism should leverage the abilities of the TPM to care about the protection of locally stored data as well as the attestation of the own identity in a communication scenario. The necessary modifications should be reduced to the possible minimum while existing concepts and projects should be reused to minimize the risk of new errors wherever possible.

The discovery and exploitation of vulnerabilities does not only result in the necessity to implement a patch management but also raises the demand to have a defense against an attacker who compromises the system.

1.6.2 Main Contributions

The main contributions of this work can be split into two separate parts. The first contribution is a new method to create digest values which are then used to fill the platform configuration registers. The second invention is an enhanced access control and enforcement system, based on a multilevel security model and encryption.

The new digest creation method is actually made out of two individual subsystems. In the first step, we overcome the strong dependency on the immutability of binaries in the TCG measurement concept and make the TPMs abilities usable with a mutable and transient system. Instead of creating the digest directly from the binary, we use digital certificates in a public key infrastructure and make the recorded digests a representative of important components and parameters of this infrastructure. Besides a file-based method, that stays close to the original TCG measurement approach, we created a state-based digest creation method, which can become useful in more complex boot scenarios. In a second step, we add a certificate revocation management to react on discovered vulnerabilities. While the used cryptographic mechanisms represent the state of the technology as of today, their arrangement and especially the selection of components which make up the new digest values are the result of our work.

Our second contribution enhances a given access control and enforcement system with ideas taken from a multilevel security model. Based on the assumption, that risk is not a “yes-or-no” decision, we map protection demands of individual items and assumptions about the quality of used software components to trust levels. Using an existing solution for file encryption with TPM based keys as foundation, we have developed a concept to transform the trust level into a physically enforced protection property. By interconnecting the vulnerability management system with the trust level assignment, we achieve a protection system which is capable to moderate access to items based on the currently assumed threat level on a system.

1.6.3 Structure of this Document

The organisation of the remaining document is as follows. Chapter 2 summarises important issues of the state of the technology. In the beginning, section 2.1 explains the mechanisms used today and their problems, which was already done briefly in the previous chapter. The successive sections 2.2 and 2.3 give a short introduction on the relevant issues of digital signatures and public key certificate infrastructures as well as recent development and basic components of trusted computing technology.

The first two sections of chapter 3 introduce the theoretical basics and the mechanisms of the new digest creation methods followed by a section on risk management. The last two sections of this chapter explain the assumed system model and the runtime protection model based on the multilevel security concept. How to put these ideas together to get a working system is outlined at the example of our prototype implementation in chapter 4.

Before we finish the thesis with a short conclusion and an outlook in 6, our analysis in chapter 5 points out the necessary efforts, gained advantages and possible threats of the invented protection system.

Chapter 2

Introduction into the Used Basic Technologies

2.1 Currently Used Techniques

This section explains the mechanisms and deficiencies already introduced in 1.3 in more detail and describes common attack methods and their consequences.

2.1.1 Protection Mechanisms

Protection mechanisms are technical solutions to provide the demanded protection properties given in 1.2 inside a specified environment. The expected presence of a fraudulent party amongst the communication partners makes it necessary to use secret knowledge which is only disclosed to legal participants of the group.

In the former chapter we separated mechanisms whether they use obscurity or known cryptographic systems to reach the protection target. In the following, we will describe both kinds of mechanisms.

Mechanisms Based on Obscurity

Mechanisms based on obscurity rely on the secrecy of internal structures of the processing algorithms and used transport formats. As a piece of binary data does not disclose any information about itself, knowledge about the algorithms and parameters used to create the data is necessary to reveal any structured information from it.

There are three ways of breaking such a system. The obvious one is to leak the information about the used algorithm through a social attack. As everyone who is involved in the development of the system must have knowledge of, at least parts of, the secret processing code, it should be possible to find somebody who discloses his knowledge. If this is not an option, two technical solutions remain.

A cryptanalysis requires at minimum access to pieces of protected data and an idea about what the plain text should look like. The chances for a successful analysis raise, if we have access to the mechanism and can watch the systems response on freely chosen input values (known-text-attack [46]).

If we have direct access to the software, we can perform reverse engineering [53] on the code and will finally receive a readable copy of the algorithm inside the binary.

Mechanisms Based on Cryptographic Algorithms

Cryptographic methods comprise of two components, an algorithm and a set of initialisation parameters. According to the principle of Kerckhoffs, the algorithm should be public and the secrecy of the initialisation parameters, usually referred to as key, the sole requirement for the systems security [9].

There are a lot of mature algorithms available today, that follow this principle and provide a sufficient solution for our demanded properties. A good summary on such algorithms is given in [46].

What possibilities are now left for a fraudulent user to circumvent the protection system? Leaking knowledge about the algorithm is ineffective as it is usually already public. Leaking the used key by a social attack is again a possible method, but it would be harder if not impossible as the amount of users knowing such a key is significantly smaller than in the previous example. If keys are provided by fully automated processes, a social attack is even impossible.

Cryptanalysis is usually no option as modern algorithms are designed not to allow conclusions about the used key, even if an offender can chose and watch arbitrary input/output pairs. Reverse engineering of the software itself is also useless, as it does not contain any secret information.

So, if we cannot gather information from captured information by cryptanalysis and we cannot convince somebody to tell us the used key, what else can we do. We must try to get the key material during or after it is inserted into the system, which requires access to one of the systems legally participating in the communication.

2.1.2 Methods to Attack a Protection System

In the upcoming paragraphs we will outline some practical entrypoints to circumvent protection systems.

Eavesdrop a Message Transfer

Eavesdropping a connection is the least invasive method, easy to conduct and impossible to prevent or detect when using public transport lines. The only possible attack is a cryptanalysis on the gathered data which imposes no risk when mature algorithms are used. Only a badly designed system might become broken by such an attack.

Man in the Middle

For a Man in the Middle attack, an offender must not only eavesdrop the data transfer but manage to intercept all data packets between the two communication partners. If the attacker can intercept the initial connection request and trick the legal opponents to accept his identity, he is able to read and manipulate all exchanged information. Using strong authentication requires a preshared base but eliminates such a threat.

Sandbox Examination

The most promising technical attack is the examination of the protection system while it is in operation. For the obscurity part of a system, we can do such an examination in a special laboratory sandbox where we can control the processing environment and easily monitor or even modify each resource access done by the examined subject. As the obscurity system does not use external input, it should behave equally on the real target system and inside the sandbox.

Subverting a Live System

For cryptographic systems, we must perform the examination in the live environment, as we do not have the desired keys in a sandbox setup. To conduct such a live examination, we must subvert the protection functions of the live system in a way, that the legal user does not recognize the modifications and enters his key material into the compromised system.

Usually a running operating system protects itself from manipulations to its own core system. An ever possible and easy approach is the insertion of a control layer underneath or into the operating system kernel while the system is shut down, which is an easy task for an attacker with physical access to the platform. An attacker without physical access must circumvent the protection mechanisms of the operating system to place manipulations directly into the live system, which is possible through an unpatched vulnerability. A look on the security advisories of Secunia [47–50] shows, that such vulnerabilities are discovered from time to time in all major desktop operating systems and as “Many systems remain vulnerable to security flaws months or even years after corrections become available” [1], we can expect that the chance to find an exploitable hole in a system is high for the majority of systems during a huge part of their lifetime.

There are some studies about new threats arising from modern architectures with support for hardware virtualization. Such a virtualization is fully transparent to the guest system and eases the development to insert and hide control layers into the monitored system [41, 43].

2.1.3 Consequences of Disclosure

While the efforts to successfully disclose an algorithm or key material are hard to guess in general, the consequences can be stated without any doubt. If we manage to reveal an obfuscated algorithm, the result is a total loss of security for all installations using this particular system. All material released in the past can be decrypted and the whole algorithm must be redesigned and deployed to compensate the breakage.

If a key gets disclosed, the attacker primarily gets access only to material which is under control of this particular key. If he manages to reproduce the necessary steps to extract the key from a running system, he can extract arbitrary keys whenever they are handled by a system. Contrary to a disclosure of the obfuscated algorithm, we are faced with a problem in the implementation of the algorithm which can be easily modified to prevent any further exploitation of the gained knownledge. Material protected with other than the disclosed keys is not affected as long as no legal user reveals keys to a system running the weak implementation while an attacker is present.

2.2 Digital Signatures and Certificate Infrastructures

Digital signatures are an appropriate tool to provide integrity and authenticity for arbitrary data items, even in a hostile environment. This section gives a short introduction on signature schemes based on symmetric and asymmetric cryptography and outlines the properties and mechanisms of a public key infrastructure based on the ITU-T-Standard X.509 [55].

2.2.1 Symmetric Key Signatures

Signatures based on symmetric algorithms require a shared secret for all involved parties.

An easy way for such a signature is a salted-hash or HMAC function as denoted in RFC 2104 [35]. The input data is extended with the common secret using basic operations as concatenation or the XOR operator before it is passed to a hash function like MD5 or SHA-1. As the hash function is irreversible it is impossible to gain knowledge about the input data and therefore it is not possbile to extract the key from the signature. The verifier just does like the signer and compares the result against the received signature token.

As an alternative, the hash-value can be taken by a standard hash function and encrypted with a symmetric algorithm afterwards. The receiver decrypts the received hash or encrypts the one he calculated on the probe and compares both results. As an attacker has access to clear text and cypher text, algorithms that are known to be vulnerable against a known plaintext attack, like DES, are not suited for this purpose.

HMACs are very fast and symmetric encryption is still a lot faster than public key signature schemes. Besides the symmetric methods provide an equivalent security level at a shorter keysize. Their main problem is the distribution and storage of the shared secret. As every party has access to the same secret, everyone is able to create a valid signature. The TPM might offer a secure storage for such a shared secret and there are secure methods to distribute it, but as the TPM does not support symmetric algorithms, the actual cryptographic operation is always done in software. This requires availability of the key in the system memory and in case of a security leak, the key might be disclosed which breaks the security base of the system.

2.2.2 Public Key Signatures

Public Key Signatures use asymmetric cryptographic algorithms where the key consists of two pieces, one of which is secret and one is public. One of the best known algorithms of this class is the RSA algorithm [33], which is used widely on the internet, for example for securing HTTP [40] or signed/encrypted eMail [20], and which is the main algorithm in the TCG concept. One of the major benefits of public key signatures is, that distributing the public key imposes no security problem and that everyone who knows this key can check the validity of a signature. A disadvantage is the high complexity of the underlying mathematical calculations. If a large number of signatures must be processed or the available resources are very low, like on mobile or embedded systems, the RSA algorithm might not perform in a suitable manner.

While RSA is suitable for signature and encryption, there are other algorithms like the Digital Signature Algorithm DSA [46, p553] or its successor ElGamal [46, p543] that provide only signatures.

2.2.3 Public Key Certificate Infrastructure

A Public Key Certificate Infrastructure according to ITU-T-Standard X.509 (ISO/IEC 9594 [55] respectively RFC3280 [30]) consists of services and structures that provide manageability of key material and signatures. The basic element is the digital certificate, a data structure containing a set of attributes. These attributes describe the technical methods, like the cryptographic algorithms, that were used to create the certificate, an unique identifier for the subject certified and a serial number for the certificate. Further there are attributes that describe the issuer and, which will become important in our usecase, attributes to give a time based validity interval and a pointer to a revocation service, where certificates can be declared invalid before their stated validity period is over.

X.509 Public Key Certificates

Public key certificates are used to certify a cryptographic key. The subject is used to bind the certificate to an entity, for example an eMail address or the name of a person or institution while the public part of the users key is referenced by a special field, named public key info.

By daisy-chaining more than one certificate a key hierarchy is created. Usually, a key is used either for signing other certificates or for end-user purposes like signing and encrypting documents. A typical setup consists of three hierarchic levels, whereas the topmost node is called the root key. As no authority exists above this key, the certificate about the key is signed with the key itself and represents a certificate authority (CA). This root key is used to certify a small number, for most setups one is sufficient, of intermediate keys, often referred to as Sub-CA or User-CA. These intermediate keys are now used to sign the end-user keys.

From a functional point of view the root and intermediate level can be merged, if there is only one intermediate CA, but having them separated has a big advantage from a cryptographic point of view. The intermediate key is used frequently to sign the end user certificates. Each usage raises the risk that the key gets revealed and raises the chance for an attacker to create a collision between a self chosen fraudulent certificate and a legal one. As a countermeasure the lifetime of the signing key should not exceed a certain amount of signatures. On the other hand, exchanging the root key has to be done out of band of the existing infrastructure raising a strong effort and new risks that should be avoided. Hence, it is reasonable to stick with separated root and intermediate keys where the root key can have a much longer lifetime as the intermediate key, superseding expensive root key exchanges while keeping the risk of a key compromise low.

X.509 Attribute Certificates

Attribute certificates are another kind of certificates within the X.509 family, defined in section 12.1 of ISO/IEC 9594-8:2005 [55], but not that widespread today than their public key pendants. While public key certificates link a public key and a subject, an attribute certificate assign a set of arbitrary defined attributes to a holder. The description of the holder is not limited to distinguished names but can also reference a filename, integrity hash values or other, nearly arbitrary chosen identifiers. The definition of the attributes is left to the user and is just limited to the technical boundaries of the ASN.1 language and the used encoding rules.

An example on the usage of attribute certificates for authorization is the profile given in RFC 3281 [14], which defines an attribute certificate used to grant rights to an entity. The profile offers some basic attributes to describe common properties in the field of authentication and authorization, like role and group membership, but also allows the creation of own attributes.

For our work, we need such attribute certificates in two places. First, for the rollback protection of the CRL verification system and second, for the property definitions of the system files. As the profile given in RFC 3281 does not suit our needs for these, we created two new profiles based on ISO/IEC 9594. A formal definition of these is given in A.1, A.2 in the Appendix.

Certificate Revocation

The possibility to revoke certificates is one of the most important features we need for the proposed content protection system. Usually a certificate is issued with a fixed validity period, but there are two classes of reasons why it can be necessary to invalidate a certificate before the validity period is over. Loosing control over the certified key is one reason, the expiry of the conformance between the actual situation and the certification policy the other one. To explain the issues, a short real life example is given. A company hands out keys to each employee for signing and encrypting email. If the employee looses the key or has the suspicion that a third party revealed it, the control over the key is lost. If the employee leaves the company, he is no longer allowed to sign mails in the name of the employer and the requirement of the certification policy has expired.

A straight forward approach is re-issuing certificates with a very short validity period, so called Short-Lived Certificates [7,45], as long as the requirements are fulfilled. This method causes a high output of certificates that must be distributed to all involved parties, forces a faster renewal of the intermediate keys and requires the frequent availability of a network connection. The major design parameter is the validity period which affects the necessary workload of the issuer on the one hand and the accepted latency and the dependence on a network connection on the other hand.

To overcome this situation, the X.509 standard defines two explicit ways for revoking certificates. First, each CA should issue a certificate revocation list (CRL) that contains the serial numbers and the date of revocation of all revoked certificates. The list itself has also a validity period after it should not be used and an updated version should be fetched from the crl distribution point (CDP). It is allowed and common practice to poll the CDP more frequently than at the end of the given validity period, thus the latency for detecting a revoked certificate is at maximum the time elapsing between two polls which can also be adjusted depending on the importance of the requested validation.

While CRLs can be distributed via insecure or asynchronous channels and are suitable for systems without a permanent connection to the internet, the second method within the X.509 standard is for systems with a permanent connection to an authoritative server. The online certificate status protocol (OCSP), as specified in RFC2560 [37], describes an online service that delivers information about the status of a certificate. The verifier send a list of identifiers for the certificates in question to the responder and obtains a response indicating the status of these certificates. The standard does not require but offers the usage of a sender generated nonce and signed responses to prevent replay attacks and forgery. As the response is created fresh upon the request, no latency is introduced by the protocol. The latency of the response is solely determined by the freshness of the underlying datasource at the responder system. If the responder is connected directly to the leading database of the CA, there is almost no latency between a revocation by the CA and the availability of this information at the verification instance.

The server-based certificate validation Protocol (SCVP) [18] is another online status protocol that basically provides a similar service like the OCSP but also verifies the validity of the trust chain of the requested certificate. The protocol is in draft status since it was first published in 1999 and no usable public implementation is known to the author at the time of writing, so it was not investigated further during this thesis.

2.3 Recent Developments on Trusted Computing

2.3.1 The Trusted Computing Group

> "The Trusted Computing Group (TCG) is a not-for-profit organization formed to develop, define, and promote open standards for hardware-enabled trusted computing and security technologies, including hardware building blocks and software interfaces, across multiple platforms, peripherals, and devices. TCG specifications will enable more secure computing environments without compromising functional integrity, privacy, or individual rights. The primary goal is to help users protect their information assets (data, passwords, keys, etc.) from compromise due to external software attack and physical theft." [24]

PC-System Related Activities

The TCG workinggroups are active on several areas of computing and infrastructure equipment whereas the majority of issued specifications provide services based on the Trusted Platform Module. When working with PC-Systems, three different specifications are of interest. The "Trusted Platform Module (TPM) Specifications" [25] describe the general properties of the TPM hardware while the "PC Client Specifications" [22] go into detail about the implementation of a TPM module into a PC-style computing system. The current specifications support platforms with a conventional BIOS [21] and the newer EFI [27, 28]. The TPM module can do only a few basic operations on its own while a lot of services are actually provided by a software stack running on top of the hardware. The TCG issues the “TCG Software Stack (TSS)[1] Specifications” [23] which describe the extended services and their interfaces.

2.3.2 Trusted Platform Concept

General Concept

The main idea behind the trusted platform concept, as defined by the trusted computing group, is the creation of a chain of measurement values that prove the trust state of the platform. A measurement in this context means, the SHA-1 hash value of the item to measure is calculated and the result is written into one of the platform configuration registers. The next paragraphs explain the individual functional components inside and around the TPM, which are responsible to perform those measurements or necessary for the functions offered by the TPM.

Trusted Building Block

The “Trusted Building Block” defines the peripheral components aside the Trusted Platform Module which are necessary to initialise the trust chain and communicate with the TPM. This section gives a short overview of the accompanying entities on the PC platform.

LPC-Bus The LPC-Bus (Low Pin Count Bus) [5] is the default bus system on PC-style motherboards to communicate with low speed I/O devices. The TPM is connected to the chipset and the processor of a system through this bus. One remarkable point regarding this bus is the presence of a dedicated reset pin which enables an attacker with physical access to compromise the platform under certain circumstances [34].

Core Root of Trust for Measurement (CRTM) The Core Root of Trust for Measurement complements the BIOS of a PC platform. It provides executable code to perform the initial measurement of the platform and represents the basis of the chain of trust. According to the specification, the code must be immutable and each code execution after a reset of the platform must begin at this code [21, p30-31,33].

Physical Presence Mechanism Some TPM commands require the proof, that the user is physically present at the platform. One possible mechanism may be implemented via the keyboard controller, which is therefore a part of the Trusted Building Block [21, p117].

[1]The meaning of the letter ”T“ in TSS is not clear, the TCG uses both ”TCG Software Stack“ and ”TPM Software Stack“ in public documents, while in researcher papers and projects often the word ”Trusted Software Stack“ is read. For this work we stick to ”TCG Software Stack“ as it is the title of the specification document.

Platform Configuration Registers (PCR)

The Platform Configuration Registers (PCR) are the key element of the TPM where measurement values of the different boot phases are stored. As the number of measurements can get large, the storage mechanism should not limit their number while keeping a complete history off all measurements done.

The measurement routine bases on the SHA1 hash function and is represented by the pseudocode

$$PCR_{new} = f_{SHA1}(PCR_{old}|Value),$$

where | is the concatenation operator. Due to the irreversibility of the hash function it is infeasible to predict an input value that leads to a certain new register value which is also true for the concatenation. Thus an attacker might not find a sequence of measurement values that ends up in the same register value than the present one.

Each register needs to be 160bit wide to store the result of the hash function and must be kept in a shielded location to prevent external modification. To be compliant with the current PC Client Specification a TPM must provide at least 24 individual registers [21, p19] which are assigned to different phases of the boot process. The first sixteen registers, PCR0 to PCR15, form the static core root of trust (see 2.3.3) where the lower eight are used during the pre-operating system phase by the CRTM, the BIOS code and the platform loader while PCR8 to PCR15 can be used by the operating system. These PCRs are initialised on each platform restart with zeros and are not resettable while the platform is running.

PCR17 to PCR22 are usable only in the context of a dynamic root of trust (see 2.3.3) which requires a recent mainboard/CPU with a special instruction set[2]. While the lower PCRs of the SRTM can be read or written at any time, these PCRs have a new property named “locality” assigned. The current locality level is determined by the CPU and included in every command send via the LPC-bus. Each register has its own bitmap under which locality it is allowed to read from or write to it. As a new feature, the PCRs are resettable without taking down the whole platform. A brief description how these new features support dynamic loading of different operating systems and help to eliminate some issues with the SRTM is given in 2.3.3.

The PCR23 is intended to be used directly by applications and is accessible and resettable at any time

PCR16 is for testing and debugging and not used during normal operation.

Endorsement Key (EK)

The endorsement key is a RSA key-pair with a keysize of 2048bit [25, p29] and is unique for each TPM. The TPM can create the key internally, which is recommended from a security point

[2]Currently offered as Secure Virtual Machine Architecture (AMD) and Trusted Execution Technology (Intel)

of view, but can also use a key injected from outside. The main purpose of the EK is the authentication of the platform which requires a certificate over the EK, stating that this key belongs to a genuine and compliant TPM.

The private part of the EK is obviously critical for the security of the authentication process. The public part is not security critical in the usual sense but can be used to trace the platform over several unique transactions, which might affect privacy of users. Therefore, the public key is guarded by the TPM and handed out only after proper authorization by the owner.

To protect the privacy and also the security of the platform, external signing requests are performed with an “Attestation Identity Key” and not with the EK itself wherever possible.

Attestation Identity Key (AIK)

The attestation identity keys were introduced as an alias for the EK to protect the secrecy of the EK and the privacy of the user. A platform can, in theory, create an arbitrary number of such AIKs using a different key each time it wants to prevent the linkability of two transactions. To convince the opposing party that the used AIK belongs to a real TPM, a certificate of a trusted third party, often called Privacy CA, is used. Each time a platfrom creates a new AIK, it connects to the Privacy CA, authenticates itself using the EK and the EK certificate and sends the public part of the AIK. The Privacy CA will verify the EK certificate and issue a certificate over the AIK without including any information that is suitable to link an AIK to the EK or other AIKs. A detailed description of AIK creation is given at [25, p58-59].

Using AIKs instead of the EK provides another advantage if the platform owner is not equal to the user or has more than one user. Using the EK requires the TPM owner password, which is also used for other critical operations that in general should not be performed by a user. The AIKs are protected by user-provided credentials and thus can be used independently from each other. This is important especially in the case where the AIK is used to deliberately link transactions.

Storage Root Key (SRK)

The storage root key is a non-migratable 2048bit RSA key and represents the topmost node of the local key hierarchy. It is created during the initialisation process of the TPM and its private part will never leave the originating TPM. Its public key is exportable and can be used to encrypt data for this particular TPM, but it is up to the user to prove the relation between the public key and the platform, as the key contains no direct proof of its origin.

The SRK is the only key, except the EK, which is stored permanently in the shielded environment of the TPM. Any other key used with the TPM is encrypted, at least indirect, under the SRK and stored outside the TPM on an external storage device.

The `TPM_Clear` command destroys the SRK and with it all keys underneath and renders all data associated with the platform inaccessible. This step is essential if the platform is handed over to another party to prevent the new owner from accessing keys or data.

Monotonic Counters

The monotonic counters are designed to provide an ever-increasing value during the whole lifetime of a TPM. The creation of a counter requires the TPM owner password and the assignment of a new authentication credential which is used subsequently to increase the counter value while read access to the counter is possible without any authorization. The specification demands, that the counters can handle an increment rate of once every five seconds for at least seven years of continous operation. The TPM must provide at least four individual concurrent counters, but demands that only one of them can be incremented between two platform resets.

When creating a new counter, the initial start value is set to the increment-by-one of the largest existing counter value, as otherwise an attacker might decrease a counter by deleting and re-creating it with the same name. In [25, p88] the internal structure is explained by an example.

TCG Software Stack (TSS)

The TCG software stack specification is intended to provide a universal programming interface for application developers across operating systems and platforms. Its specification is more than a pure hardware driver and defines high level functions and services instead of direct interfaces to the TPM hardware. For economic and security reasons, the TPM was designed to

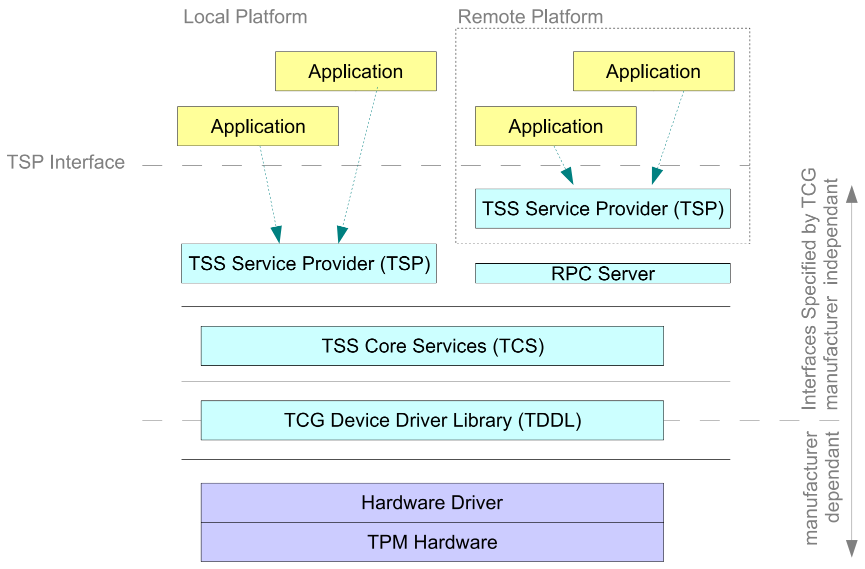

Fig. 2.1: Layered Architecture of the TCG Software Stack

handle only the absolute minimum of processes and data while the majority of calculations and operations is done inside the unprotected application space of the operating platform.

The TSS runs entirely in user mode and is split up into three layers as shown in figure 2.1. The lowest layer is the TCG Device Driver Library (TDDL) which communicates directly with the kernel space hardware driver. As the hardware interface is not fully specified by the TCG, this part is hardware dependant and therefore the TDDL needs to be provided by the manufacturer of the hardware platform. The TDDL-interface towards the next layer is standardized in Chapter 6 of the TSS specification [23, p711ff] and provides a basic instruction set to communicate with the TPM.

Upon the TDDL, the TSS Core Services (TCS) are running. Usually there will be one instance per platform running as a system service. The interface exports the primitives and basic functions to handle the resources of the underlying TPM. The TCS interface is also specified in the TSS specification [23, p496ff] and is available locally as a linked library but can also be exported to a remote platform using an RPC server.

The top-most layer is the TSS service provider (TSP) which provides an object oriented, application centric interface to different services. Multiple TSPs can run in parallel and share a common TPM as the underlying TCS interface will keep track of the individual sessions and assign the TPMs resources adequately. Instances of the TSP can run either locally or remote via the RPC connection simultaneously. The TSP interface is the same, regardless if the TSP runs on a local or a remote TSS, so the actual location of the TPM is irrelevant for the application.

2.3.3 Application of the Trusted Platform Concept

Initial Setup of a TPM-enabled Platform

Before the TPM is usable, we must setup the endorsement key and the storage root key.

The endorsement key is usable only with a corresponding certificate of a trusted authority to convince a remote party about the origin of the key. As the authority can verify the origin and uniqueness of the key only by physical presence, the key is usually created during the manufacturing of the TPM chip itself or at the time when the platform is assembled.

With version 1.2 of the TPM specification a set of commands to erase and recreate an EK was introduced. This feature is unnecessary from a technical point of view but was demanded at most by privacy organisations and is useful for usage scenarios in closed groups, like in a company environment, where the EK certificate is issued by a special authority. For individual users, the creation of a new EK raises the problem to receive a trusted EK certificate for the self-generated key, as no other method than physical supervision guarantees that the certificate request belongs to a TPMs EK.

The second initialisation step is executed by the platform owner when control over the machine is passed over. During the “Take Ownership” process, a new SRK is generated and all registers of the TPM are initialised with default values. The owner is prompted to provide a new password for owner authorization and the storage root key. While the owner credentials are rarely used and should be set to a secret value it is usual to set the SRK password to a well known or even no value and put it into the startup code on disk. This is necessary as each access to keys of the TPM requires access to the SRK. The well known password is uncritical for security as each operation with a key underneath the SRK can be protected with another password and the SRK itself is never used directly.

Static Root of Trust measurement (SRTM)

Most of the services the TPM offers rely on the integrity measurement values stored in the PCRs. How these PCRs are filled is partly defined in the platform-dependant parts of the specifications and we will give a short summary of the information for PC-platforms given in [22, p33ff] here.

Upon a restart of the platform, the first code that gets executed is loaded from the Core Root of Trust for Measurement. This code is responsible to perform the measurement of the hardware platform, which is grouped into three parts. The first measurement round includes the CRTM itself and all components that are physically bound to the mainboard. Afterwards, firmware and configuration registers of external components, like PCI-Slot cards, are measured. As last entity the initial program loader is measured and executed. For each measurement group, executable code and configuration options have their own PCR assigned. The specification is not fully accurate what entities must or must not be measured into these PCRs but at least advises to exclude user-dependant data from all measurements.

After these measurement steps, the PCRs 0, 2 and 4 contain values that should be predictable by the manufacturer of the platform and immutable between boot-cycles, as all data that was measured includes only code maintained by the platform manufacturer or its delegates. The PCRs 1, 3 and 5 store only configuration data which should not influence the trust status of the platform, if it is guaranteed that all subsequent steps continue with performing measurement steps. The only exception is PCR1 which can include optional microcode updates for the CPU. Such updates might affect the behaviour of the main processor on the lowest level and cannot be detected afterwards, which requires that this PCR is included when making a trust statement, if microcode updates are possible on the platform.

The PCR6 is responsible to guard transitions between the different power states of a system. The specification suggests not to guard transitions between the ACPI Sleeping states S1 to S3 [12, p24], as the operating system stays active during these states and the chain of trust remains intact. A problem might arise from the transition out of S4 (“Suspend-to-Disk”), as the whole operating system, including its internal states, is written to the disk. While the system is suspended, an attacker might tamper with the saved system state and bring up a modified system on resume. To signal the operating system, that it was resumed from the S4 state, the PCR6 records transitions from S4 and S5 (“Off”) state differently.

The usage of PCR8 to PCR15 is not specified by the TCG and left to the operating system which is started from the boot loader. At the time of writing, no of-the-shelf software is known to the author, that use the SRTM to protect from tampered boot binaries. Microsoft Windows Vista seems to use only the lower PCRs to seal the BitLocker encryption key to the BIOS and MBR integrity [6] but does not use the PCRs for any additional measurements after the boot loader was executed. One of the most advanced projects, that uses the higher PCRs is the TrustedGRUB [52] project. A modified version of the standard bootloader GRUB [17] that writes fingerprints of itself and the kernel image to a PCR before executing the code. Table 2.2 shows the measured entities and their associated PCRs.

PCR Index	Usage
PCR 0-7	Hardware
PCR 8	Binary of stage1
PCR 9	Binary of stage2
PCR 12	Arguments given to Grub
PCR 14	All Binaries loaded by stage2 (kernel & modules)

Fig. 2.2: PCRs used by TrustedGRUB

Dynamic Root of Trust measurement (DRTM)

The SRTM got its name because it is not possible to reset the PCR values or perform additional measurements during the runtime of a system. The concept of a dynamic root of trust measurement allows multiple independent measurements without a full reset of the platform and, what is even more important, eliminates the dependency on the measurement values of the SRTM which, as shown in this work, imposes some major problems.

The DRTM measurement is based on a new instruction supported by recent processors. By calling the instruction `skinit`[3] /`senter`[4] from an already running program, the processor recreates a new environment and bootstraps from a verified source. The initialisation of the secure environment is done in several steps. First, the processor is initialised as it would be done on a full restart, the locality index is set to 4 and the registers of the DRTM are reset. Now the bootstrap code is read from a trusted source, usually from the BIOS and locked into memory in a way that no external process can access it. Afterwards the code is validated against a certificate which is shipped with the code and send to the TPM via a secured channel, where the fingerprint is extended into PCR17. Before the bootstrap code is executed, the locality register of the processor is decreased to 3, which disables the reset capability for the registers PCR17 and PCR18. The further steps are not defined in the TCG specifications, the PCRs 19 to 22 are usable by the trusted operating system whereof PCR20 to PCR22 are resetable from locality 2.

[3]on AMD Platforms supporting Secure Virtual Machine Architecture

[4]on Intel platforms supporting Trusted Execution Technology

Key Hierarchy and Cryptographic Operations

Key Usage Types and Key Hierarchy The TPM provides a secure storage system for RSA keys and supports encryption and signature generation with these keys inside the shielded environment. The interface was not designed to provide a fully fledged cryptographic coprocessor for arbitrary keys but is limited to be used with TPM-owned keys on data blocks up to the RSA key length. Each key must have one out of the four different key usage types Storage, Signature, Migration or Attestation assigned. While keys of the latter type are created and handled with an own set of commands the other three are handled equally through the same management functions.

Due to limited storage inside the TPM, all keys are stored outside the TPM when not in use, protected by encryption with a wrapping key. For the AIKs this wrapping key is always the storage root key, for all other keys it can also be any other TPM-owned key of type storage as shown in figure 2.3. Based on this parent-child relation the keys span a hierarchy tree with the SRK as the topmost node. It is obvious, that using a key requires access to its parent key to get it decrypted from the external storage. As this requirement applies recursive to the parent keys, all keys in the hierarchy up to the SRK must be loaded into the TPM when access to a key is required. The number of parent keys between a key and the storage root is not limited by the specification or technical means, but as all keys of the hierarchy must be loaded into and decrypted by the TPM, the latency grows with each level. As each key can be protected by a user provided authentication secret, this hierarchical storage mechanism provides an access control system at the same time.

Binding Keys to a Platform State The protection of a key through encryption and authentication credentials prevents an attacker from directly accessing or using the key without the interaction of a legal owner but does not provide protection from abuse by system compromise. The specification offers the possibility to lock down each key to a specified set of PCR values,

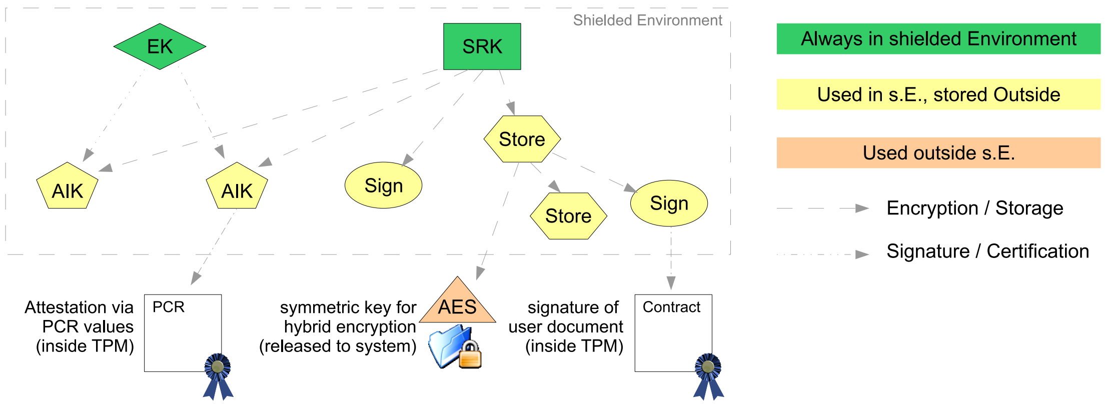
Shielded Environment
EK
SRK
AIK
AIK
Sign
Store
Store
Sign
Attestation via PCR values (inside TPM)
PCR
symmetric key for hybrid encryption (released to system)
AES
signature of user document (inside TPM)
Contract
Always in shielded Environment
Used in s.E., stored Outside
Used outside s.E.
Encryption / Storage
Signature / Certification

Fig. 2.3: Hierarchy of Keytypes using the TPM

that must be present when the key is loaded into the TPM. In combination with a suitable measurement routine a compromise of the system reflects in modified PCR values which causes the TPM to deny access to the key material. In conjunction with the hierarchical storage model it is even possible to bind a key to the transition of a sequence of different PCR states.

When binding a key to a certain platform state, the migration property of a key is of essential matter. Basically, keys protected by the TPM never leave the shielded environment in an unprotected form and are bound to the particular platform. Thus, a party who secures material with a TPM-owned and PCR-bound key can be sure that it becomes accessible only on a genuine TPM platform during a defined state of operation. For certain scenarios it is necessary to share keys between platforms which is supported by a set of commands to migrate key material. Unfortunately there is no technical way to ensure that migration is done only between TPM-protected platforms and thus the migration system can be abused to reveal plain keys outside a TPM-protected environment. As a proper authorization of several entities, including the platform and the key owner, is necessary to migrate a key, this does not raise any security considerations as long as the credential holders cooperate with the security target. This attitude changes, if a third party hands out material under the assumption that the TPM platform protects their assets. To bridge the gap between both necessities, each key carries a flag that indicates if the key is migratable or not. The specifications mandates, that certain keys, for example AIKs or wrapping keys for the seal operation, are non-migratable. For obvious reasons, a set migratable flag is virulent for all children.

Data-Encryption and Sealed Storage The TPM offers two different methods for encrypting data. The first method is a simple rsa-based encryption with TPM protected keys. As described above, a user can create a new keypair inside the TPM and retrieve the public part of the key. The TPM itself does not offer encryption under a public key but this is easily done by hand or using an appropriate helper function from the TSS. To decrypt the data of such an operation, the user must load the correct key into the TPM and run the TPM_Unbind command. This sequence will work also with migratable keys and therefore allows share or backup of encrypted data between different platforms.

The other method is known under the term sealing and frequently used in our system approach. The `TPM_Seal/TPM_Unseal` commands are both executed inside the TPM and differ in two properties from the encryption method above. First, sealing is not allowed using migratable keys and thus the data is bound to the specific platform. Additionally, the sealing operation includes two sets of PCR patterns, whereof one shows the PCR state at the time the seal was created and the other one acts as binding condition for the unseal process. The TPM will refuse to decrypt the data, if the values in the PCRs does not match the ones mentioned in the object.

From the view of the bare encryption process, both methods are RSA calculations without additional preprocessing like padding or segmentation and therefore the datasize is limited to the key size. To encrypt data blocks larger than the used RSA key while retaining the strong protection of the TPM, a hybrid encryption system should be used. A possible approach uses the symmetric AES algorithm to encrypt the application data and uses the TPM to wrap the used AES key into a cryptographic envelope. A major problem, when using encryption techniques, is the generation of a real random key, so it is not possible for an attacker to shorten the key range for a brute force attack. The TPM can support this critical task with its built-in random number generator and provide unpredictable real random data for the AES key.

Digital Signatures Besides data encryption, the TPM provides a facility for digital signatures. Signature keys are always leafs of the key tree, depending on the SRK and an arbitrary number, including none, of intermediate keys. The TPM supports two different signature commands, which are both usable with every key from the key hierarchy that has the correct usage flag set.

The `TPM_Sign` command [26, p126] accepts input data from an external source and signs it with the loaded key, which must have its key usage flag set to Signature. The only benefit of the TPM in this case is the protection of the signature key as no statement about the platform state nor a relation to a TPM platform is expressed directly by the signature. If the used key is bound to a PCR configuration, the signature creates an indirect relation between the system state and the signature but requires an intermediate authority to express this circumstances.

The `TPM_Quote` command [26, p160] is used to express the relation to a certain platform state directly, as it includes a structure of selected PCR values into the signature. As mentioned above, the signature key has no provable relation to the platform and thus this reporting of PCR values is only useful in a controlled environment. Otherwise, the key owner might prepare a faked structure looking like legal PCR information and sign it using the `TPM_Sign` command or create such a signature without a TPM at all. The `TPM_Quote` operation also accepts AIKs as signing keys, which is then called Remote Attestation and explained in the next section.

Remote Attestation

A remote attestation should prove the integrity of a platform against an external challenger. If both parties agree on the integrity of a hardware configuration and a set of binaries, it is sufficient for the platform to prove, that only such binaries were used to reach the current state. If platform and binaries use the PCRs to measure themselves, the current PCR values are suitable to prove the correctness of an untrusted transaction log.

A successful attestation splits into three steps.

1. The challenger sends an attestation request to the platform. A nonce is included in the request to prevent replay attacks. The platform creates a signature over the nonce and the current value of its PCRs and sends it back to the challenger. The transaction log

over the boot process might be included to support the challenger to identify the running configuration.

2. The challenger uses the transaction log, or other knowledge, to calculate the resulting PCR values for a platform running in the assumed state and compares them against the ones submitted by the attestor.

3. The challenger uses the nonce to check that the signature is fresh and verifies the certificate chain of the signature to ensure it was created by a genuine TPM.

The current TPM specification defines two different protocols to verify the authenticity of a signature. The older method uses the `TPM_Quote` command to create an RSA signature with an AIK to link the message to a genuine TPM. This approach has two pitfalls for the privacy of the user and raised a lot of criticism regarding the role of the Privacy CA. If a platform uses an AIK in two individual transactions, the challenger can link both together as the AIK identifies the platform. Creating a new AIK for every transaction might be a theoretical solution, but makes a high throughput and a fast response of the Privacy CA mandatory, which conflicts with the necessary security requirements. A second aspect is the trustworthiness of the Privacy CA, as they are in the position to link each AIK to the corresponding EK.

The solution proposed in [3] solves both issues and was adopted as second method for remote attestation for the current version of the TPM specification. When using this “Direct Anonymous Attestation”, we still need a trusted third party to certify that the used DAA credential originates from a real TPM. The DAA protocol uses a proof of knowledge instead of a RSA signature and thus does not expose any recurring information to the challenger over multiple sessions. Hence, neither the certification instance or the challenger, nor both together are in the position to detect subsequent operations from the same origin. take. under

2.3.4 The Open Trusted Computing (OpenTC) Project

About the Project

The Open Trusted Computing (OpenTC) Project is a research and development project co-financed under the 6th framework program of the european commission. The project target is the development of an open source framework around the TPM and the TCG concepts. The implementations are mainly done with respect to the upcoming x86 platforms but the concepts are usable on mobile and embedded devices, too. Further information is available at the project website at http://www.opentc.net [54].

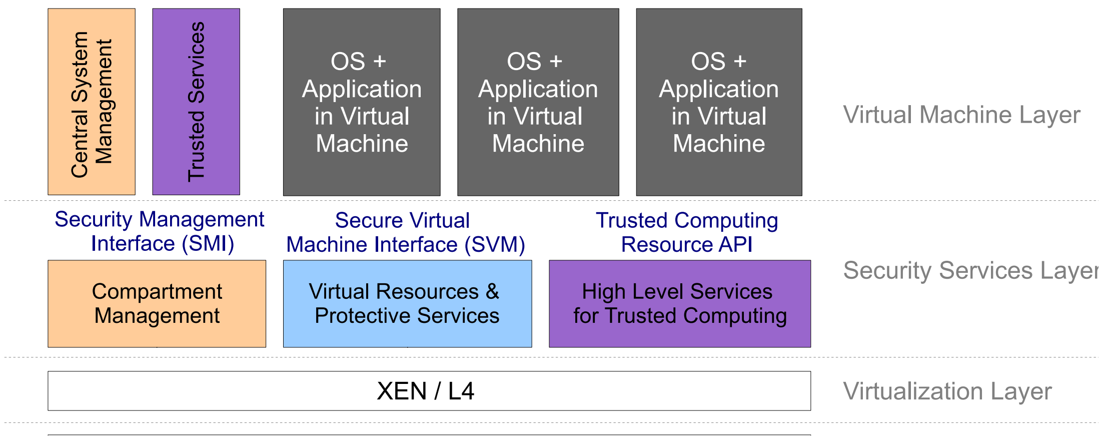
Central System Management
Trusted Services
OS + Application in Virtual Machine
OS + Application in Virtual Machine
OS + Application in Virtual Machine
Virtual Machine Layer
Security Management Interface (SMI)
Secure Virtual Machine Interface (SVM)
Trusted Computing Resource API
Compartment Management
Virtual Resources & Protective Services
High Level Services for Trusted Computing
Security Services Layer
XEN / L4
Virtualization Layer

Fig. 2.4: OpenTC Layer Architecture

The OpenTC System Concept

The system concept is based on virtualization. Instead of having one monolithic operating system running on the physical host, the applications are grouped according to security and usability aspects and are put into virtualised compartments. These compartments run on top of a secure hypervisor which provides strong isolation for the shared physical resources. The technical basis for the PC-platform prototypes are the L4 microkernel or the XEN virtual machine monitor which are extended by a new, intermediate layer, the `Security Services Layer`, as shown in figure 2.4. Besides the virtualised resources, this layer provides three groups of new services which are partially based on the capabilities of the TPM. The individual application systems, which run inside their own virtual machine on the virtual machine layer, can access the provided services through special interfaces, as also shown in the figure. [5]

Compartment Management The first group is a set of management services to handle anything related to the execution of the compartments. Access to this management functions is only possible from a special instance on the virtual machine layer, which takes the role of a management system. One of the most important services provided by the compartment management is the `compartment security manager` which provides a user interface to perform a measurement and attest the current state of the host system platform and the compartments. Regarding the attestation of compartments a set of predefined security properties [11,31,42] is used instead of values from the binary measurement which allows a fast decision weather the started compartment meets a given policy or not. Based on this decision, access to resources as hardware devices, intercommunication between compartments or services running in the management domains is moderated.

[5]This section describes parts of the concept as published or internally discussed at the time of writing. As the whole project is work in progress and not all information is documented or disclosed to the public, the given information might be incorrect or overhauled in the future.

Virtual Resources & Protective Services The provisioning of virtual resources is an inherited function from the virtualization layer while the protective services add secure communication and storage on top of the virtualised resources in a way that is transparent to the guest system. The whole work to technically apply the protection is done by services running on the host system and the protected resources are offered to the guests through standard interfaces. Two major advantages arise from this approach. First, from inside the guest system the protected resources are indistinguishable from unprotected components and therefore usable without any modifications inside the guests. Second, an attacker from inside the guest can neither manipulate the protection system nor gain access to credentials as all related processes happen outside the environment of the guest.

An example using those protective services is the currently developed OpenTC demonstrator for “Personal Electronic Transaction” [11]. A guest suitable for secure webbased transactions, e.g. for online banking, stores the confidential data on a storage device which is under control of the host system. The only outbound connection of the guest is a SSL protected network link, also controlled by the host system. To unseal the user credentials, the guest must pass the local integrity verification and the authenticity of the SSL link must be verified. In addition, a remote attestation against the banking server is done to ensure the integrity of the communication partner. As the verification is done outside the guest and without user interaction, the usual weakness that the SSL verification fails either due to the unawareness of the user or due to malware spoofing the verification chain is eliminated.

High Level Services for Trusted Computing While the already described services are responsible to create and maintain the secure compartments and their environments and are mainly invisible for the guest, the third group of services is targeted to support application developers inside the guests. Examples are convenient interfaces to store and access keys or data under protection of the TPM from inside the guests.

Chapter 3

Concept for a New Protection System

The first part of this chapter describes the main idea behind the new digest creation method and discusses different technical concepts. In the middle part, we give an introduction, why risk and security are no binary arguments and why it is impossible to make a quantitative statement on their value. The chapter closes with a concept, how qualitative risk classification can be used to build a protection system based on a modified multilevel security model.

3.1 Finding a Suitable Evaluation Criteria

3.1.1 Requirements on the Digest Creation Method

The requirements on the new measurement system are twofold. First, we want to distinguish authorized and fraudulent changes to system files and ensure the accessability of sealed data in the former case. Second, a system meanwhile identified as vulnerable should not be able to access formerly stored data to prevent intruders from extracting any data. To use the TPM functions for this purpose, we must work around the relation between fingerprint calculation, values in the PCRs and access to the sealed data.

The access control is based on the selected set of PCRs and is done by the hardware. Changing the behaviour of the hardware is not within the scope of this work, so the only point we can work on, is the association between files, fingerprints and PCR values. The first component under the control of the platform owner is the code of the initial bootloader, which we can modify to fit our needs. The internal hardware-based components like the BIOS code and even the first track of the bootloader software are measured by code implemented in the platform's hardware and thus not accessible for this work either, which excludes these components from our improvements.

The basic idea is, to keep the values assigned to the PCRs constant for all legal situations. This will grant permanent access to sealed data (see 2.3.3) and provides a unique value to an attestor, disregarding actual details of the configuration but proving a predefined statement. Currently, the values feed into the PCRs are created by applying the SHA-1 cryptographic hash function directly on the binary of the measured file. If we want to keep this mechanism, we must ensure that all legal binaries produce the same output, which means

$$sha1(Binary_a) = sha1(Binary_b),\ a \neq b$$

which is obviously contrary to the the design concept of a cryptographic hash function. Instead of running the hash function on the binaries of the software, we need a representative value that can be used to create the relevant PCR values and is received from an artificial mapping. This mapping should deliver an unchanged value for a file that is a legal successor, but should not resolve to the “correct”[1] (old) value for a binary that was trusted before, but has turned out to be vulnerable.

3.1.2 Integrity as Decision Criterion

Before we can start to create a technical representation of this mapping, we must define under which conditions two files are allowed to be exchangeable and therefore be represented by the same value. The, for cryptographers, obvious answer to this question: “if the system-integrity is not affected”, leads to the question, what is a system’s integrity and how can it be measured.

Integrity is an abstract measure - there is no intrinsic method to calculate the integrity value of something. Instead, many formal definitions of the term “integrity” are used in different fields of application.

In information security, the term integrity expresses that a defined set of properties of an observed subject remains unchanged between two observations. This seems to be close to the understanding of “integrity” by the TCG, where the observed property that must stay unchanged is the hash value of the binaries.

The Stanford Encyclopedia of Philosophy [58] gives a broader definition:

> ...integrity is connected in an important way to acting morally, in other words, there are some substantive or normative constraints on what it is to act with integrity.

Having a look on the motivation for the technical concept reveals, that the latter definition is close to the problem that should be addressed. Nobody has the intention to enforce a certain value of the PCR registers or a certain binary to be used. This technical measure is taken upon the assumption, that the selected values represent a system, which behaves according to a given expectation. As we will show in some of the next paragraphs, there is no general way to

[1] correct is interpreted as “expresses the sane state” and allows access to sealed data

formally prove such a conformance and, according to the model of Arbaugh, vulnerabilities will be discovered so the observed behaviour deviates from the expected behaviour after the initial evaluation.

In this case, there are two possibilities to keep the integrity statement on the system upright. In certain cases it might be possible to simply bend the specification to fit the observered behaviour of the software. This might be a good choice, if the abnormality affects only parts that are irrelevant for the actual use case or can be fixed easier outside the system. It is obvious that a change in the specification might imply legal or technical problems, as normally multiple parties agree on such a specification and base their business models or cooperating technical systems on it. Therefore, the normal procedure to react on a software bug is to fix it and release a new version so the observed behaviour matches the expectation again.

Regarding the introductory question, what requirements should be met by two files to be exchangeable, the above paragraph gives an answer. The integrity state of a system can be evaluated by comparing its observed and expected behaviour against a given specification. The result is a suitable criterion, to judge if the update of a particular file should preserve the trust statement, given to the platform.

3.1.3 Measuring Integrity

The given answer again introduces a new question. What does such a specification look like and how is a file compared against it? There are several standards that deal with documentation of requirements and testing during different phases of software development, for example ISO/IEC 25051[2], ISO/IEC 9126[3] and ANSI/IEEE Std 1008-1987[4]. These specifications are suitable to verify that a piece of software operates as expected within its regular field of application, but do not provide any methods to judge on security aspects. Reliable detection of, accidentally left or deliberately placed, code that can be used to produce an unexpected behaviour is an unsolved problem nowadays. There are tools and common guidelines available to assist developers on reviewing the code or performing tests, but besides the inaccuracy of such tools, the ressources necessary to perform such tests prohibit their usage during regular operation. The concept of proof-carrying code [38] sounds promising but is currently a subject of ongoing research and not suitable for a real world implementation.

So, for practical reasons it is necessary to delegate the integrity check to a party, that has access to sufficient resources and can verify the conformance of a file with a given specification. As files and their behaviour do not change during the normal operation of the system, it is sufficient to perform the examination once and attest the gained result for later use. It is also feasible and

[2]ISO/IEC 25051 - Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Requirements for quality of Commercial Off-The-Self (COTS) software product and instructions for testing

[3]ISO/IEC 9126 - Software Engineering Product Quality

[4]ANSI/IEEE Std 1008-1987 - IEEE Standard for Software Unit Testing

already common in software development to issue a certificate of conformity based on a quality assured development process without inspecting the actual result.

3.2 Approaches for a New Digest Creation Method

Based on the results of the former section, the technical part of the new digest creation system can be reduced to an attestation system which does not perform conformance testing itself. A given attestation must fulfil three prerequisites. It must be readable within an automated process and contain information about the conditions and the result of the performed test. We further need a process to derive an identifier from this data which remains unchanged for all legal versions of the attested entity[5] but is also unique for every entity. As this identifier is used in the measurement process, it must be impossible to forge an identifier or swap the identifiers of different entities without the authorization of the attesting authority.

Attribute certificates using digital signatures as presented in 2.2 are suited to provide arbitrary but unforgeable bindings between a subject and given attributes under the supervision of a keyholder. In the remaining parts of this section, we explain possible methods for the identifier derivation process and give a solution for an embedded version control system.

3.2.1 File-Based Mapping-Functions

The first class of derivation functions stays close to the original TCG approach and sets an entity to an individual file which requires an unique specification and identifier for each file in the system.

The workflow of a verification is common for all proposed variants. On access of each file, the chosen attestation method is used to extract the identifier for this particular entity and is written into the PCR. In case no matching identifier is found, an error value is written into the PCR. As writing to the PCR is not protected by any means and the used identifier is not secret, an attacker can use a tampered binary without a matching attestation and write the correct value into the PCR himself otherwise. In the end, the system would show the measurement values of a correct system while running a tampered version.

The difference in the proposed variants is the way how identifier and signature mechanism work together.

[5]An entity is the smallest unit which is compared against the specification.

Using One Key per Entity

Each key is used only for signing the different files of one entity which makes the key a unique representative for this entity. Therefore we can directly use the public key as identifier for this entity and write it into the PCR after a successful signature verification. The signature is created over the hash value of the file and can be either directly attached to the file or stored in a separate location. In the latter case its convenient to use the hash-value as identifier to link the signature to a file.

This approach has a very close relation to the original solution proposed by the TCG and while the necessary changes to the measurement functions are small we introduce a huge overhead to manage the keys and signatures. We must transfer one public key along with each file and must securely store and manage their private counterparts at the maintainers side. Both requirements result in a huge technical effort.

Using One Global Key

Instead of using a unique key per file, we sign all files with a global signing key. As the key is no longer a representative for an entity it is useless to write it to the PCR and we need another way to record and represent information about the verified files.

The filename is not a suitable candidate for an identifier as there might be two files with the same name but representing different configurations and therefore do not match the same specification. This will work only if we use different signing key for each possible configuration which is technically possible but causes overhead. It is more comfortable to create an artificial identifier for the entity and include it in the signature. Configurations which differ only in a few files can use one common signature key, as the identifiers of the individual boot configurations differ and lead to a changed PCR value.

If we now use solely this identifier to record the sequence of verified files, we end up with one problem. The PCR values do not contain any hint on the used signing key and therefore an attacker might forge signatures with the correct identifiers using his own key. To prevent this sort of attack, we need to write the used key and the identifier to the PCR. As the key remains unchanged for all signatures, its sufficient to write it to the PCR once. It is important to write to the PCR on start up before the first file is verified as writing it at the end of the start up sequence still leaves the possibility for an attacker to use his own signatures and finally write a forged key value into the PCR.

Writing the identifiers to a logfile in memory and signing it at the end gives a performance advantage but also raises a new attack channel. If a vulnerability in a trusted file is disclosed, an attacker might abuse it to interrupt the boot sequence and afterwards manipulates the logfiles to hide execution of the vulnerable file. The finally signed logfile will allege another state than the present one to a remote challenger.

Hash-Based Reference Database

This method is a mixture of some current solutions like used in the TrustedGrub Project [52] and the signature based approach evaluated in this work.

The two prior methods create a cryptographic signature for each individual file and derive the identifiers from this signature. In this approach, we put all identifiers into one common database and just protect the database as a whole by one signature.

As in the other examples, the identifier is used to represent the membership of a particular file to an entity and we therefore must set up the database to map the stored identifiers to files on the disk. We achieve a fast and convenient mapping, if we use the hash value of the file as database lookup key and put the identifier into the associated dataset.

The signature created on the whole database replaces the role of the global key and represents a common policy which is valid for all bindings contained in the database. The used key is written into the PCR first and for each subsequent file verification the identifier gained from the database is appended to the PCR. As far as the signature key of the database is identical with the global key, the resulting PCR value is identical to the one using the global key approach from the previous paragraph. This model obsoletes per-file signatures and lowers the computational efforts but raises additional ones to extract the necessary information from the database.

Comparison of the Variants

All setups fulfill the requirement to preserve an integrity state for different files but fail to revoke trust from a broken configuration.

Regarding their implementation, the signature-attached solutions are fairly equal. The one key per file solution needs fewer lines of additional code so the risk of introducing an exploitable bug is lower, but as the total amount of code differs not that much and is at all not very complex a strong audit should abolish this advantage.

From a theoretical point of view, the one key per file paradigm looks superior on the first glance as one might argue that one revealed key affects only one file. As breaking one file is sufficient to break the whole system, we can neglect this advantage. On the contrary, as it is more difficult to handle a large amount of keys than a single keypair and the propability that a key is revealed increases with the number of keys used, the expected benefit of multiple keys turns into a disadvantage. Especially in case a symmetric key system is used, the risk of revealing a key might become a substantial security threat. The penalty for this ease of key handling is a larger number of available valid signatures, which lowers the complexity to find two files that match a common signature. An estimation shows that this can be neglected in practical dimensions:

Using a SHA-1 Hash, there are 2^{160} different hash values. Even under the assumption that all files of the operating system are protected individually by this method and each file exists in 10 versions and is signed separately, the number of available signed hashes is not larger than 2^{20}. To compromise the system it is necessary to create a malicious file that matches one of the available 2^{20} signed hashes. This results in a preimage attack using brute-force and requires an effort of $\frac{1}{2^{20}} \cdot 2^{160} = 2^{140}$. Today, this is far beyond the computational feasibility.

The database approach looks superior from a management point but has a larger impact on the code basis, offering a broader spectrum of attacks against the verification mechanisms. Besides, everytime one file is exchanged the whole database must be refreshed resulting in additional management overhead, which might become a dominating factor if the number of protected files is large.

3.2.2 State-Based Mapping-Function

The mapping on a per-file basis sticks close to the mechanisms used by the standard TCG approach and helps us to overcome small variations in the used files, which is mainly useful to address discovered vulnerabilities due to the lifecycle model. We might use it also to run different configurations of an operating system where the general structure of the boot process is identical but the loaded files differ.

The approach fails to preserve the inner status of the TPM, if the number and order of files changes, which is the case not only for different system configurations but also affects the application of the concepts after the linear boot phase in modern multi-threaded operating systems. To deal with those situations, we must extend the mapping function to represent only the result of a whole initialisation step instead of showing up the individual files in the measurement.

The application of the integrity definition onto an individual file was motivated by the current TCG approach. From a general point of view it does not make a difference, if we verify either the state of a complete system or the expected behaviour of an individual file against a given specification. This assumption matches the one used by Sadeghi et al. in their work about “property based attestation” [42]. Instead of attesting individual properties we incorporate them into a specification and guarantee the compliance with it. The technical solutions proposed by Sadeghi in [42] require either a modification of the TPM hardware or the permanent supervision of an external trusted party. We extend the file-based approach given in the former section 3.2.1 of this chapter and provide a solution using currently available hardware.

The main problem we have to work around in this case, is the order of measurement, execution and PCR setting. We cannot judge on the final state until the very end of the startup but we have to take care that an attacker can not interrupt the startup and write the measurement himself. In the next paragraphs, we present three different control models, to bring up a complete system and represent it with a single measurement value.

Snapshot Verification Prior Execution

In a “Snapshot Verification”, the complete measurement is done with the files found on the storage media before any code is executed. As this breaks atomicity of measurement and execution, it must be ensured by the environment, that no changes to the files can be made after the measurement was taken. For local storage media, we might have sufficient control to prevent write access [6], but in general this might require to preload all code into the systems memory and measure and execute it from there.

The measurement procedure itself is a bit different from the one used in the file-based scenario. As we do the measurement prior executing the code, we need to know which files will be involved in the startup process. If relevant behaviour of the bootup procedure can be altered by configuration files or user input, it is also necessary to consider those items during the measurement. The easiest approach to perform the measurement is, to calculate one single hash value over all involved files and create a certificate that binds the final identifier to this measurement value. From a management point of view, it would be more comfortable to use a staged system, where each individual file has an identifier assigned and their sequence is used to calculate the final lookup key.

If the final identifier is found, it is written into the chosen PCR and the code gets executed. It is important, that no one can interrupt or change the boot sequence once the procedure was started as otherwise the code executed does not match the one reflected by the already written identifier. If we can not find a matching certificate for the calculated measurement value, we have to write an error value instead of the identifier. In this case, we do not have to monitor the further bootup, as the integrity statement is invalid anyway.

Inline Verification

One possibility to perform a verification at the time when the code is actually executed is a direct verification by the calling party. This requires that each binary which directly calls another piece of code must be extended to measure and verify the code before it executes it. Such an implementation is used for example in current implementations of the TrustedGRUB [52] bootloader.

Compared to the snapshot approach we neither have to care about immutability or intermediate storage of code nor we have to know the executed files in advance. To determine the final identifier we can follow two different approaches. First, we can agree on the expected target in the beginning of the execution sequence and enforce that only code which leads to this chosen target gets executed. Alternatively, we can pass over the current result of the measurement to each following component and delegate the creation of the correct identifier to the last component.

[6] Tampering with the hardware always offers ways to do so while countermeasures raise the efforts for successful attacks. The protective actions taken should match the overall protection level and value of the system

This requires at least, that each executed component is trusted to preserve the passed information and call the correct successor. If we support only a single state of trust, the consequences of an undetected compromise are the same, namely access to the protected resources. In a system where different trust statements are possible within the same verification system, the second approach raises a security relevant threat. A compromised component originating from one configuration can manipulate the measurement to represent another configuration and finally gains access to information assigned to this configuration. The first approach does not impose such a threat, but requires to select the configuration and write the identifier to the PCR at the very beginning. If code does not match the chosen configuration, its execution must be denied or the PCR must be set to another value before the code gets executed.

Verification by External Supervisor

This third method is a combination of the ones given before and exploits the nature of the operating system, that all requests to load or execute data go through the memory manager routines of the running kernel. Therefore it is usable only for later boot phases after the kernel was started. As the kernel has its own memory space that is not accessible by normal processes, we can assume the supervisor inside the kernel as an isolated instance. We can therefore use a measurement system alike the one in the snapshot scenario above, as executed fraudulent code can compromise the system but not affect the kernel internals. We gain the advantage of a simple measurement without prior knowledge of the execution sequence and without modifications to external components.

Conclusion

Either one of the proposed methods is suited to derive a common PCR state from non-deterministic parallel running execution phases or even differently structured boot sequences without raising security related problems. The third method combines the advantages of the first and the second one and is therefore the method of choice in the post-kernel boot phase, if we have access to the kernel to place the necessary modifications. In a real system, we often do not have a sharp differentiation of boot phase and operation phase and can not define a final target as system services are started only on demand. We can handle such situations if we set the time to reach the final target to infinity. The configuration identifier is than interpreted as a policy identifier which we obviously must put into the PCRs at the beginning of the measurement and afterwards enforce that no executed process endangers the expected behaviour.

3.2.3 Establishing Version Control to React on Vulnerabilities

It was specified as a basic requirement to revoke trust from configurations that have shown vulnerabilities. As signatures are digital items which can be easily copied, there is no way to destroy a signature once it is released to the public. Thus, we must transfer some additional information along with the signatures to provide a way to invalidate them artificially.

Instead of raw signatures, we assemble the necessary information to a digital certificate and embed it into a certificate infrastructure (see 2.2). Signature keys are protected by X.509 public key certificates while we use X.509 attribute certificates to link identifiers to entities and system states. The profile of the used attribute certificates is based on the definition given in section 12.1 of ISO/IEC 9594-8:2005 [55] and described in the appendix at A.2.

The certificate infrastructure offers several ways to control the validity of the trust relation expressed by the signature. Including the certificates validity properties and the revocation mechanisms of the infrastructure into the signature verification results in a fine grained version control system, where each individual attestation statement can be declared as invalid at any time. We have to extend the verification component in the selected measurement system to care about this revocation information and treat signatures from expired or revoked certificates as not existing.

As an attacker who can hide the most current revocation status from the system can allege a revoked binary as valid, we must protect the revocation information itself from manipulation, too.

Reliable Revocation on Systems Without a Permanent Network Connection

Without a permanent connection to an authoritative certificate status server, the system must rely on a locally stored datasource to check the validity of a certificate. The certificate revocation list is designed for offline use, but has some pitfalls for the given application scenario, that can be used to successfully attack the system. The list itself suffers the same problem as the raw signatures discussed earlier - without any additional information a valid signature shows only that the list was correct at the time of signing, but no statement can be made about the correctness of the provided information at the actual moment. Usually, the CRL [7] is issued with a validity window to limit the possible latency until an updated version is available at all peers and with it the timeframe for an exploitation of vulnerabilities. As today's computer systems do not provide a tamperproof time source, an attacker can fool the verification system by manipulating the system time, which enables him to install any revocation list that was issued by the correct certification authority at any given point in the past. An attacker just needs an exploit for a known vulnerability, a copy of the vulnerable binary and a copy of a revocation list issued before the vulnerability was discovered. He can use such a copy at any time to undermine the verification system and execute the vulnerable binary in a sane context.

[7] certificate revocation list, see 2.2

Monotonic counters (see 2.3.2), as introduced with version 1.2 of the TPM specification, can provide a partial solution on the problem. Each time a new CRL is received, we increment the value of the assigned counter and create a special binding certificate that links the counter value to the current CRL. The verification system reads the value of this counter from the hardware and looks if it matches the one stored with the CRL to make sure it uses the most current one. As the counters cannot be decremented, it is impossible for an attacker to recreate a system state which accepts an old CRL as valid. A point we cannot prevent is blocking of updates. Our implementation, as given in 4.5, uses the standard operating system to fetch an update and requires a platform reboot to update the CRL binding. A local user can always block this process while a remote attacker might abuse a present vulnerability and block updates to stay undiscovered. If the system is sane and the user cooperates, the presented method provides a sufficient protection to keep the revocation information up to date as far as we can regularly access the network to check for an update.

Reliable Revocation on Systems With a Permanent Network Connection

Systems with a permanent network connection can also use the CRL to judge on the validity of certificates. There is no need to store or protect the freshness of the CRL by the system as the CRL distribution point can be queried for the most recent version whenever necessary. This will guarantee a low latency even if the CRL is renewed before the end of the intended validity period, which might be the case if a vulnerability is discovered. However, instead of downloading a whole CRL and check the certificate in question locally, the use of a certificate status responder produces less load and guarantees the most recent status. Similar to the temporary online system, an already present attacker can block requests to the external verification sources and use this unavailability to hide himself.

3.3 Risk and Vulnerability Management

3.3.1 Vulnerabilities of Protection Systems

Risk management for a deployed computer systems infrastructure means to deal with different types of threats arising from the presence of vulnerabilities. Some of these vulnerabilities are obviously known, while the presence of others is only assumed but proven by experience.

Disclosure of Confidential Information

The protection systems mentioned in 2.1 as well as our certificate based digest creation method rely to a certain extend on the secrecy of information. Obfuscated algorithms, cryptographic key material and authentication credentials can become disclosed as a result of technical or social attacks. The consequence of a disclosure is always a breakage of the given security promises but the number of affected items as well as the efforts which are necessary to compensate the disclosure and bring the system back into a sane state depend on the kind of disclosed information.

Discovery of Technical Flaws

The presence and discovery of technical flaws in software, colloquially called “bugs”, is a commonly known problem and subject of the lifecycle model of Arbaugh [1]. Such flaws usually evolve from misstakes done by humans while writing the code [57]. Some of them are suitable to circumvent security mechanisms and enable an attacker to gain unauthorized access to ressources.

3.3.2 Estimating Risk

In IT security, risk is often defined as the product of the chance to get hit by an attacker and the caused damage when an attack is successful [56]. Both factors are hard to estimate quantitative. The probability is a combination of the attraction of the system and the kind and quantity of exploitable vulnerabilities while the possible damage comprises of different aspects depending on the endangered items.

Why it is Necessary to Rate the Risk

So, if the estimation of a risk is hard to do and offers only vague results, why is it necessary to perform such an estimation? Obviously, it would be best to stop using the system or at least the affected components, if a vulnerability is disclosed. Figure 1 and Figure 4 in [32] show the number of vulnerabilities effective on each single day during the period of a year for a chosen system. Even if the author does not go into detail about the kind of vulnerabilities and it is likely that not all of them are relevant for a given system setup, the numbers show unambiguously that a system is endangered by disclosed vulnerabilities for a considerable part of its lifetime.

The unavailability of a system is usually not an option and also causes damage so we have to find a way to compare the damage resulting from the unavailability against the one arising from a successful exploitation of the vulnerability. To finally come to a decision and either accept the risk of an attack or avoid it at the costs of the unavailability, it is necessary to have at least a qualitative statement to do the comparison.

Accounting Possible Damage

The possible damage arising from a successful attack comprises of two different kinds. After an attack, the technical system must be repaired, which includes the removal of any installed malicious software and fixing the abused vulnerability as well as restoring affected data. The damage calculates from the direct and indirect costs related with this work, which is easy to estimate.

The unavailability of a system, the disclosure of information to unauthorized parties or processes running with manipulated information also cause damage, which is hard to estimate in general.

The direct financial losses are calculable while indirect economical losses or consequences due to the violation of legal issues cannot be expressed in financial terms. The IT-Grundschutzkataloge [56] categorize the expected damage into three levels and uses qualitative statements from six different realms to assign a suitable level. Based on the assumption, that an attacker exploits the given vulnerability we can evaluate this scheme for each protected item on the affected system and assign a potential damage.

Probability for a Successful Attack

The probability for a successful attack is calculated from two sources, the number of attacks and the chance for one attacker to be successful. If an attacker is successful depends on his personal skill and if the attacked target is vulnerable for the exploit he uses. The personal skill and also the number of attackers evolves over time, as the vulnerability is evaluated and knowledge about possible attacks is distributed. Figure 3.1 shows the propability to get attacked using a certain vulnerability based on its evolution over time according to the lifecycle model of [1].

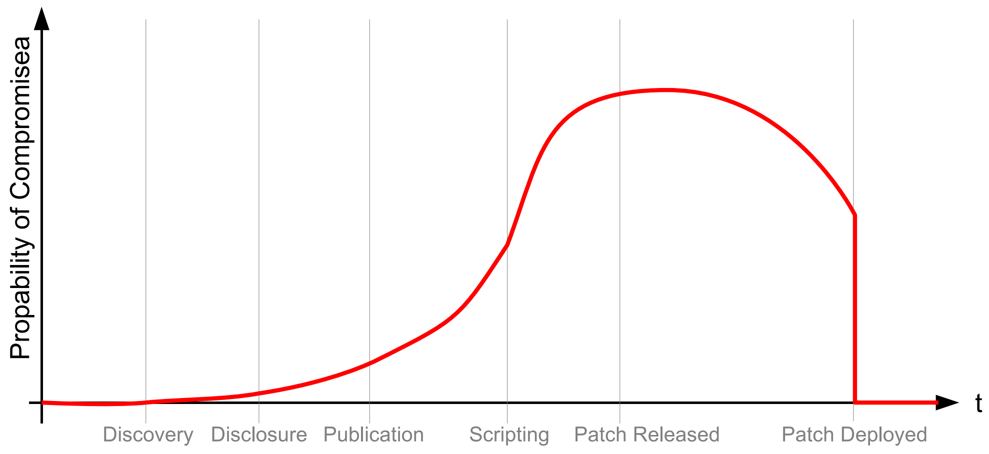

Fig. 3.1: Stages of the Lifecycle with Propability of Compromise

Based on this model, the probability that a certain system is attacked can be divided into four phases. The vulnerability is `born` with the deployment of the defective component but there wont be any attacks until the issue is `discovered`. The model distinguishes `disclosure` and `publication` with respect to the size of the addressed audience. Both events will raise the number of potential attackers and thus the chance that a certain system is affected. Another significant raise is expected, when an exploit reaches the `scripting` state. If we assume, that the distribution of knowledge is viral, the number of potential attackers raises with exponential growth starting with the disclosure. At the moment where the `publication` and `scripting` states are reached, the growth-rate will encounter an unsteady increase.

If the vulnerability is repaired the success-rate of an attack, and thus the related risk, drops to zero but it is likely that the propability decreases earlier if the attackers move on to abuse newer vulnerabilities.

Effects of Countermeasures

The demanded effect of a countermeasure is a decrease of the propability of a successful attack. As repairing the affected component is usually out of the current scope we can primarily just reduce the visibility on the vulnerability for potential attackers. As a result, the number of available targets decreases which lowers the number of possible victims and therefore possible gains for the attackers. If the efforts rise above the expected gains or another attack promises a higher revenue, the attackers will loose interest and invest their resources somewhere else. Besides, a decreasing number of successful attacks might slow down the evolution of their skills.

3.3.3 Managing the Risk

Even if the direct disclosure of confidential information is not based on technical flaws, the management of such cases is within the range of a technical system. Although we do not pursuit such breakages further, the proposed management cycles can be adopted to deal with disclosed passwords or other secrets.

The management of risks arising from the discovery of technical flaws is dominated by two events. The first one is the disclosure of the problem to a suitable audience. Before this disclosure, the system is endangered by a privileged group of people who gained this knowledge in advance. As we do not know about the problem at the moment, we can neither estimate the associated risks nor do anything special against this threat. Fortunately, based on the model of Arbaugh et al., the number of attackers and so the probability to get attacked is low. After the problem came to our attention, we try to estimate the risk and decide if the risk is acceptable or if there are possible countermeasures, which lower the risk to an acceptable level by reducing the chance for an attacker to exploit the flaw. If this is not possible, we have to move valuable items away from the system to limit the damage in case of a successful attack. The estimation of the risk and the decision about the resulting measures must be a recurring process as the probability for a successful attack changes over time.

The second important event is the publication of an official fix for the vulnerability. After repairing the system and ensuring its integrity we can discontinue any intermediate countermeasures and release moved items back to the system. This sounds trivial but raises a major problem as the patch will fix only the original vulnerability but will not detect or remove other modifications which were done while the system was compromised. In certain cases it is not sufficient to verify all binaries on the system as an attacker might manipulate configuration files to modify the behaviour of a legal process to support his demands.

3.4 System Model

To efficiently deal with vulnerable components it is important to estimate how those affect the overall system. In the remainder of this section we outline the assumed system architecture, based on the prototype implementation given in chapter 4, and discuss briefly how the behaviour of each component can affect the security properties of a handled data item. Even if the system model is important for some aspects of the final protection system, it should be possible to adopt the general concept on other architectures, too.

3.4.1 Bootloader and Initial Kernel

Bootloader

At the end of the hardware initialisation of the TCG enabled platform (see 2.3.3) the boot process reads the bootloader from the selected storage media. Due to legacy constraints, the initial code is very limited in size and most bootloaders today are multistage loaders to overcome this limit. The size limit and some other resource limitations will become important later in the implementation.

The main objective of the bootloader is the detection and execution of the initial kernel of the operating system which requires knowledge of the binary format and the data structures on disk. Possible further options of the bootloader are unimportant for our objective as long as they are not suited to break the chain of measurements. Even if the bootloader does not contribute to the codebasis of a finally started system, its integrity is an essential matter to assure a proper measurement and startup of the kernel image.

Kernel Image

A lot of security related researches promote microkernel architectures to enhance a system's security but the systems examined for this work use monolithic or hybrid kernels, where at least parts of the hardware drivers run in kernel mode.

The kernel is the core part of the operating system and provides the interfaces to the hardware of the system and dispatches resources and computing time to the different processes. Therefore, it has unlimited access to all information floating in the system and its behaviour is critical for security relevant aspects. The kernel processes are executed in a dedicated part of the system, the kernelspace, which has special privileges and is shielded from the rest of the operating system. All other processes run in the userspace and can communicate with the hardware only through the kernel's interfaces.

Kernel Modules

Kernel modules are a trick to stick with the structure of a monolithic kernel while keeping the initial kernel image small. Kernel modules dynamically extend the kernel with new functions and can be added and removed while the system is in operation. As the modules are directly loaded into and executed within the kernelspace, we cannot protect the initial kernel from possibly fraudulent behaviour of a module. This makes it obligatory to verify the code of modules before it is added to the kernel and, if its trust statement does not match the one of the running system, either adjust the system's trust statement or deny its execution.

As a module can modify the kernels internal structures while it is loaded, we cannot restore the trust statement after removing the module from the kernel.

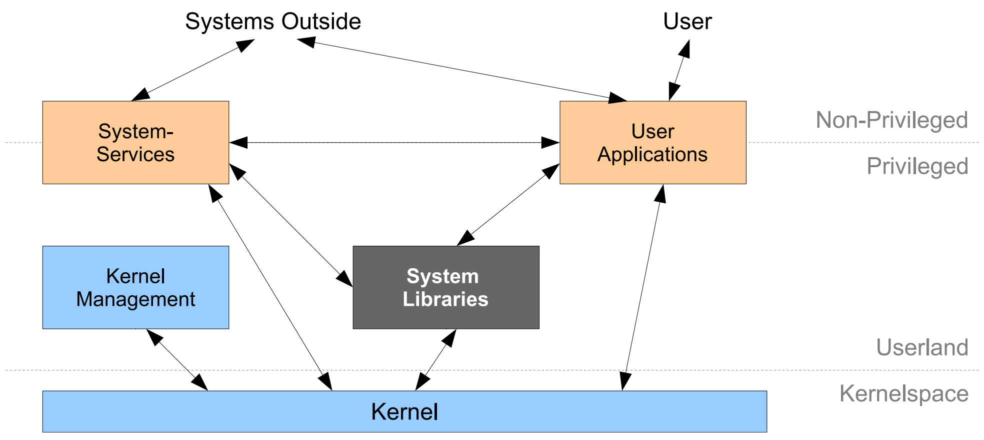

Fig. 3.2: Segmentation of Processes into Different Groups

3.4.2 Runtime Environment

The processes of the running operating system can be divided into several different groups exchanging data with each other as shown in figure 3.2. We will explain how the security properties of processes in the different groups will affect the overall security and what are the implications for our protection system. The verification of a component always implies that the current trust statement either allows the execution of the component or can be adjusted to match the new situation.

System Daemons Performing Management Services

These processes are mostly harmless as they cannot affect the data between an application and the kernel. The influence on the kernel is limited and well defined by the interfaces the kernel offers to this process. If the exposed functionality can affect given integrity assurances, the verification of the component is done with respect to the trust statement of the kernel.

System Daemons Providing Supporting Services

The daemons providing supporting services communicate directly with the user application and can, depending on the kind of provided service, affect assured properties. We can group such services into two subgroups, depending on the direction where information goes. If the application requests data from the service, the only possibility to influence the integrity is in providing a false answer. If such an answer is used in a policy calculation, for example getting the current time to check if a time window matches, the integrity is affected and the service must be integrity checked. Types of the second subgroup receive data from the service which will obviously break any assurance about confidentiality or integrity if the receiving party does not behave according to given requirements.

The necessity to include these components into the verification procedure therefore depends on the kind of information and the concrete assumptions of the trust statement.

Library Linking

The concept of shared libraries is very common to current systems and the relation between application and library can be compared with the one of kernel and modules. The code included from a shared library becomes part of the application itself and runs in the same processing environment, therefore we need to ensure that a linked library shares the trust assumptions of the application.

User Applications

The user application is the primary interface to handle requests and usually deals with protected and unprotected content items as well as key material and rendered content. The application is like a central hub, moderating the whole workflow of data from its origin to its final rendering and can manipulate or leak this information at any point in the process chain. Therefore it might affect any kind of protection target which makes it a very precarious component. Fortunately, the association with data items is very clear and easy to determine as the application itself deals with the protected items and offers only a defined set of operations. This allows a direct comparison of the demanded protection targets of a particular data item against the properties of the application.

The difficulty is given by the dependency of applications on other parts of the system as mentioned in the previous paragraphs. One challenge is to determine if a certain interaction with another component affects the security of the handled data or not.

Additional Issues with Inter-Process Interference

Besides regular communication between two processes via formerly agreed interfaces the process management of the operating system allows two other kinds of interference, which were neglected during the explanations above. The POSIX Signal interface is used by the operating system to terminate and notify processes and is usually implemented by each program. On the examined operating systems, signaling is controlled as an ordinary resource access and thus allows a user to send signals to all processes running under his user account. Such signals can be used to kill a process or to initiate a refresh of associated resource bindings which might result in a denial of service but can not be abused to gain access to any process data. To steal or manipulate data one must utilize the second method of process intercommunication, process tracing. By calling `ptrace` on Unices or `DebugActiveProcess` on Windows, the current process becomes the new parent of the target process and gains full control over memory, registers and code of the child. On the systems reviewed in our work, attaching to a process is controlled by the same access conditions like other resource access, which means a discretionary access control based on the process owner on these systems. In our protection system the trust statement of an individual component is an essential value, which can be circumvented using process tracing to gain uncontrolled access to protected data.

Special attention is necessary for processes running with escalated privileges as such can interfere with processes of other owners and therefore manipulate all daemons and applications on the system.

3.5 Adjustable Protection through a Multilevel Security Model

3.5.1 Multilevel Security Model

The multilevel security approach originates from military and intelligence services where it is common to have a strong hierarchical structure of trust and access clearances. The model consists of a set of security levels which form a partial ordered set and assigns each subject and object to one of those levels. Access is possible within the same level and crossing levels is considered to be allowed along the safe direction. Even if this sounds like a trivial condition, the classification if a direction is safe or not, depends on the type of access and the assumed security property, namely read/write and confidentiality/integrity. Therefore, a technical system must be in the position to get knowledge of these attributes on every access.

Ensuring Confidentiality in a MLS

The model of Bell and LaPadula [8, 9] was the first one to model a MLS system and cares about the confidentiality of data. Access to an item usually includes all operations possible with this type of object which are basically read, write and execute. As long as subject and object stay on the same security level, we do not have any problems, but if we cross the border of a level, we quickly run into trouble in certain situations. Alice, who has the highest level A, can access information on level A but then publish it to an item on level B. Bob, who has access on level B, will now gain access to information which is classified for level A without proper permission. The Bell-LaPadula Model invents the so called "star-property", which disallows writing to lower levels for all subject except special trusted subjects, which must ensure that all information that is passed to a lower level meets this clearance level. The second rule in the Bell-LaPadula is the "Simple Security property" which provides the obvious - a subject is not allowed to read information on higher levels.

Ensuring Integrity in a MLS

Two important models that deal with integrity of data are the models of Biba [2] and Clark-Wilson [4]. Biba reverses the rules of Bell-LaPadula and states that no object is allowed to read from lower levels or write to higher ones. Violating one of these rules enables a person on a low security level to publish false information to higher levels. The model of Clark-Wilson however cares about preserving integrity in a transaction. The central requirement is, that each transaction leaves the system in a state which is consistent with the system policy. This target is achieved through an enforcement component, that maintains a listing of all transactions a subject is allowed to perform on a certain object. For example, the access control mechanism used in the SELinux [13] extension is based on this model.

3.5.2 Adoption of the MLS Model for Risk Management

Classification of Subjects and Objects

Even if the originating field of MLS models is the military sector, the motivation behind the classification is similar to our risk estimation. An object is classified depending on the expected consequences of its unauthorized disclosure. Our risk estimation, as explained earlier in this section, is a measure for such consequences and can therefore be used to do the classification.

The role of the subjects in our adoption is not assigned to the users but to components of the system, whereas the security level is determined from two aspects.

First, we define an upper bound for the security level of a certain component which reflects the propability that this component is involved in a security breakage. The criterion for such a ranking can be for example the reached “Evaluation Assurance Level” according the Common Criteria or ITSEC standard [9]. At the moment when a vulnerability is disclosed, the security level of the component is reduced based on the criticality of the vulnerability. One challenge to the system is the secure storage and evaluation of this security level during all relevant computations to prevent a system from alleging a higher security level than it actually got assigned.

Embedding the Trust Level into the Proposed Measurement System

To embed the multilevel security approach into the proposed system we must first find a method to attach the determined security levels to subjects and objects and second modify the enforcement facilities to obey this new criterion.

As long as the core operating system, which contains the access enforcement and maintains the security levels, is sane, we can simply implement this as a feature into the used management system. To guarantee the security of the protected items, we must assure that no physical accessible information has a security level higher than the one of the protection system itself. This might impose a problem if we share data between system configurations with a different security level and will surely become a problem if a vulnerability affects the security level of the access control and enforcement system.

We can efficiently assign and protect the security level assignment of all subjects by including them into the used integrity certificates as an additional attribute. To reliably represent the value of the core systems security level we use one of the PCRs of the TPM.

The assignment of the security level to the objects is combined with the protection system, so we just need to pass the security level when we create a new file. We therefore need no protection mechanism or even storage for the security level of a file, as the storage system is unable to open the file if it is in a wrong security level. A detailed description of the protection system is given in the following.

3.5.3 Content Protection by Encryption

General Considerations

Encryption is widely used already today to protect data from unauthorized access. To protect a group of individual items on a filesystem we can distinguish three modes of operation. Volume based encryption schemes, like dm-crypt [44], present a standard filesystem with unencrypted files inside the authorized domain and show only random data to an outsider. The system depends on only one key and performs encryption usually based on storage blocks between the virtual filesystem layer and the hardware. As the data is available as plain files once the volume is mounted as a filesystem, this method does not provide any protection in case of a compromise from within the running system.

Another widely used method is content encryption where files are stored unprotected on a storage media but have an encrypted payload. The encryption process and the key management is left to the application while the operating system is not involved. The protection primarily resists against an intruder as long as he cannot extract the keys from the applications key management because the data is visible only in an encrypted form to him.

If we combine features from both mechanisms, we end up with a file based encryption on the level of the operating system. Like in the volume based scenario, key management and encryption is done transparently for the application by the operating system but an attacker, who circumvents the normal access control, cannot gain any valuable information from the file as long as he does not have the correct key. The challenge for the operating system here is to distinguish if a file access results from a legal user request or an attacker which requires the inclusion of process information into the access decision.

TPM-based Key Management

In combination with our new digest creation method we can use the TPM as a secure keystore to manage encryption keys and make them available within a defined system configuration. As mentioned earlier, a volume-based mechanism can successfully prevent a compromise by running a modified kernel or accessing the disk from outside the system but fails if an attacker compromises a running system. If the system detects the attack, it can delete the volume key from memory and lock the TPM by changing the PCR values but if the attacker gains access to the key or blocks the TPM update earlier, all protected content is endangered. Besides these problems, support for different security levels requires using separate volumes and keys for each level.

We have multiple possibilities to manage the keys in a file based approach. First, we can use a global key for all files which we put under the TPM. This stops an attacker from direct access to the files once he is inside the system but if he gains access to the global key we again loose control over all content. Like in the volume based scenario we need at least different keys for each supported security level.

Using one key per file is the superior approach but also has the highest impact on the resource consumption of the key management system. The benefit is a very high protection level even in the case of compromise, as the attacker must query the TPM for each individual file. Attacks on a raw copy of the disk are also of very low value, even for very powerful attackers, as one revealed key is usable for only one particular item. Compared to a disk based encryption this decreases the value of a broken key dramatically. Usage of the TPM also provides a physical upper bound for the number of files an attacker can leak in a defined amount of time, as the number of key operations of the TPM is limited. Furthermore, if we detect the attack and lock the TPM, the attacker is unable to access arbitrary files even if he extracted key material before.

TPM-based Key Provisioning for a MLS system

Providing the keys for a MLS system through the TPM faces us with a well known problem. Assuming we have three distinct levels in our system called A,B,C where A is the highest level and C the lowest and we want to represent the current trustlevel by a PCR value. If we now seal the keys for objects with security level C to the PCR value of the systems level C, we loose access to these objects from its superior classes A and B. We evaluated two different concepts two work around this problem, both are based on the sealing function of the TPM.

A sealed item can be protected not only by the current PCR configuration but also requires an authentication secret to reveal the information. We use this secret to distinguish the different security levels and exclude the PCR used for the security level from the sealing precondition. The secret used for each level is also protected by the TPM but this time the PCR assigned to the security level is used

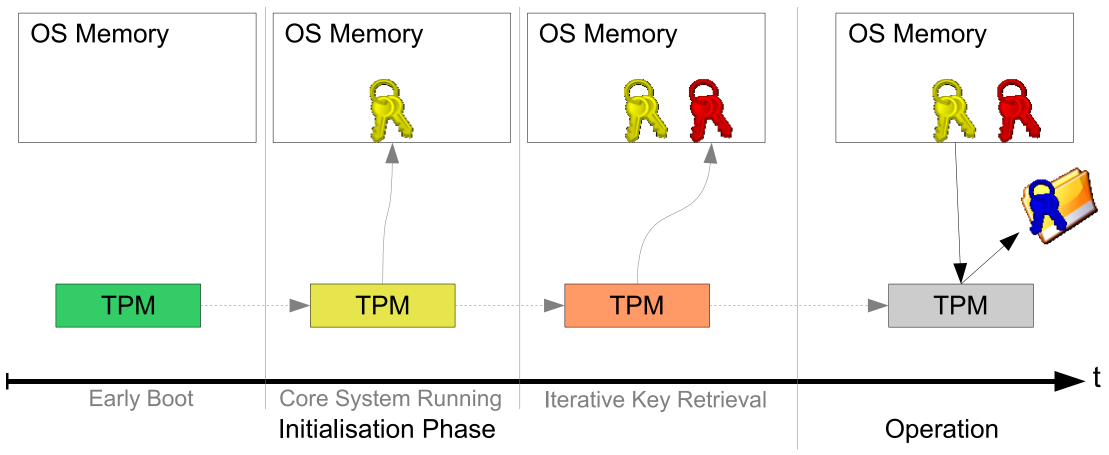

Fig. 3.3: Three-Step Key Retrieval for System with Three Levels

$$key_A = f(PCR_{Baseconfig}, secret_A);$$

$$secret_A = f(PCR_{Baseconfig}, PCR_{SLevel}).$$

As we cannot jump between the different PCR configuration but need to have the keys of all relevant levels at hand for encryption, it is necessary to iterate through the PCR configurations at least once, retrieve the secret keys of the level and store it in the operating systems memory. To be able to iterate through all inferior levels, we need to daisy-chain them starting at the highest level

$$PCR_{LevelA} = f_{PCR}(PCR_{Init} + Identifier_{LevelA});$$

$$PCR_{LevelB} = f_{PCR}(PCR_{LevelA} + Identifier_{LevelB});$$

$$\dots$$

$$PCR_{LevelN} = f_{PCR}(PCR_{Level(N-1)} + Identifier_{LevelN}).$$

The key retrieval for a system whose core system is certified for the middle out of three levels is shown in figure 3.3. The bootloader, which is the first component involved in the new measurement system, reads the security level of the measured component from its certificate and writes the identifiers for this level and all superior levels subsequently to the PCR, starting with the highest one. If during the remaining boot phase a component is executed, which has a lower security level than the present one assigned, the missing identifiers are also written into the PCR. Afterwards the PCR value represents the concatenation of all identifiers starting from the highest level down to the level of the currently measured component.

At the moment where the main operation system containing the access control components is started, the second phase is initiated. The enforcement facility needs access to all keys from the current and all inferior levels, so it reads out the secret for the current level $secret_{LevelN}$ from the sealed storage, stores it in a protected place and writes $Identifier_{Level(N+1)}$ to the PCR to retrieve $secret_{Level(N+1)}$. This procedure is executed until the secret keys for all levels are read.

The weak point of this setup is the disclosure of the stored secrets. If an attacker manages to extract the secret for a high level once through a vulnerability, he can abuse a vulnerability even in lower levels to gain access to the high level data, reusing the stolen secret. Our alternative approach, as shown in figure 3.4, eliminates this problem at the price of higher management costs.

Instead of accessing keys from an inferior level we publish keys to all superior levels at the time of their creation. This eliminates the necessity to access more than one PCR configuration during the systems runtime and the current system state can be kept in the PCR and used to

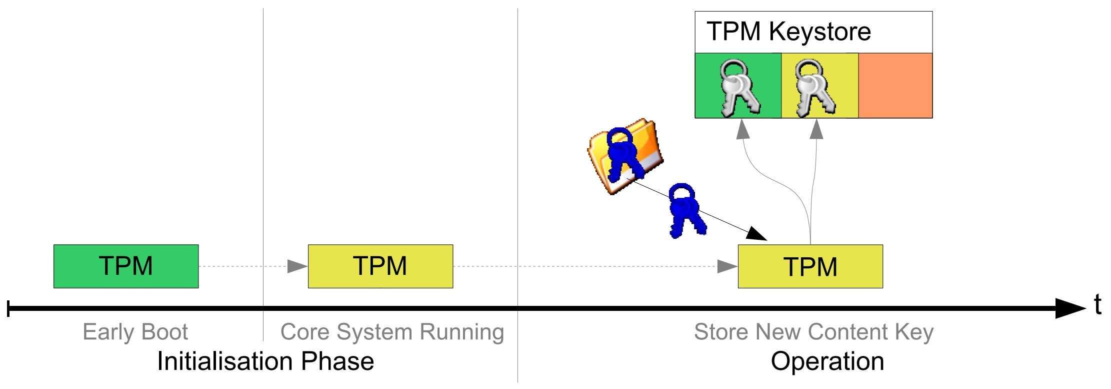

Fig. 3.4: Key Duplication for System with Three Levels

bind data under the TPM. Setting the PCR to the current system level is done as described above as it allows a safe degradation of the level during the boot phase.

The sealing function allows us to specify the expected PCR configuration at the time of decryption independently from the present state, which we use to seal the key for all demanded levels. This requires knowledge of the PCR values of all superior levels, which is easy to maintain as this information is accessible and its secrecy not necessary for security. This approach does not create a security problem, as no secret data which can be abused to break the level barriers is kept in the system. The practicability of this method depends heavily on the number of used security levels as it directly influences the necessary overhead regarding storage capacity, key management and computation power. A minor issue is the fact that we loose the ability to determine the security level of an object simply through its key, which can become problematic as a false assumption about the security level can leak data to a lower level.

Chapter 4

A Sample Implementation

This chapter will show, how we can put the ideas given in 3 together to build a protection system based on a tamperproof root of trust. The Gentoo Linux Distribution [16] was chosen as a sample platform for this work and the examples and explanations are given with respect to the proof of concept implementation we did on such a system. The concept does not regulate all details of a finally useable system and sometimes leaves different options to choose from. Within this chapter, we make decisions on those choices to stay as close as possible to established technical and organisational structures, used by the chosen implementation base. We expect a multilevel security system with three security levels A to C, as already used in the examples in 3.5.3, as a well-balanced compromise between manageability and protection, and therefore use such a configuration. Nevertheless the concept is applicable to every system based on the model given in 3.4 and should work with modifications on other models, too.

In the introductory section, we present and shortly explain a feasible hierarchy for the keys used in the verification system, based on an organisational model where hardware and software components are contributed by different vendors. The necessary workflow how to react on a reported vulnerability is also outlined. In the end of this section, we give a technical description how we handle keys and certificates in our implementation.

The sections two, three and four explain the different phases to bring up and run the system. The first phase reaches from the platform start up to the load of the operating system kernel. Phase two is responsible to bring up the operating system's initial process `init` and guard the setup of system daemons and management services. The third phase implements the actual protection system and moderates the access requests of users to the resources and does the necessary risk management and key handling. In the last section we describe the parts of the system which are responsible to maintain the revocation information.

4.1 Key Hierarchy of the Verification System

4.1.1 Hierarchy

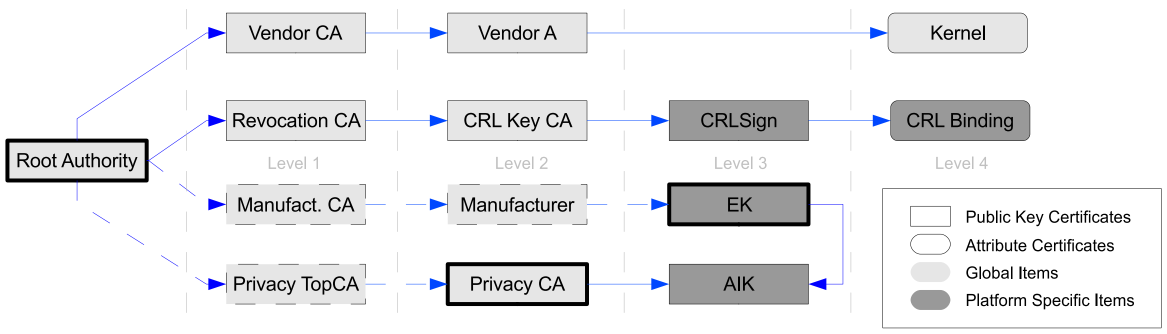

Fig. 4.1: Key Hierarchy of the Verification System

Figure 4.1 shows the hierarchy of keys and certificates used in our implementation scenario. Items with a light background are global items, which are common for all participants while the dark boxes mark items which are individual on each platform. Boxes with sharp edges represent a cryptographic key with a corresponding X.509 public key certificate according to RFC 3280 [30]. Rounded edges denote X.509 attribute certificates, for which we use custom formats based on the ITU-T recommendation for X.509 [55].

Each one of the branches is dedicated to a special type of certification but all have a similar organisational structure that reflects the organisation of the business parties. The topmost node is the common root authority, which is represented by the root key in the verification system. On the first level, directly under the root key, we have a master authority for each of the branches. Each authority possess its own key pair, where the private key is securely stored at the authority. The root authority issues a certificate over the public key, which is than distributed to all participants. On the second level, we have multiple certificates on each branch, which are assigned to the individual business parties one by one. Each single party forms a subordinate authority that can do signatures on behalf of the whole group of participants on this branch. Up to here, the segmentation into several keys has only manageability issues and can be neglected from a functional point of view. The items on level three and four have individual meanings on the different branches and are platform or entity dependant and therefore obligatory.

The branch on top of the figure includes the software vendors and distributors. Using the signature keys shown on level two, each vendor issues certificates of conformity for his distributed binaries. The certificates, shown on level four, are X.509 attribute certificates following our custom schema given in appendix A.2.

The second branch contains the organisational units which drive the revocation system. The `CRLSign` certificate and the corresponding key on the third level are unique for each system and generated during the system initialisation explained in section 4.5 of this chapter. The private key is protected by the local TPM and released only to the revocation management system. The key is used solely to issue the binding certificates, which bind the monotonic counter and the revocation list together. Level four shows the binding certificates, which are also unique to each platform and follow the X.509 attribute certificates schema given in appendix A.1.

While these two certification paths are important for the local protection system and must end up in the root key to make the proposed verification system work, the two structures on the bottom of the figure are used only during remote attestation. The dashed elements in the figure are optional, if the challenger and the PrivacyCA have direct knowledge of the PrivacyCA key respectively the endorsement key of the TPM. However, embedding the PrivacyCAs and manufacturers into the used hierarchy eases the distribution of knowledge.

4.1.2 Handling of Vulnerability Reports

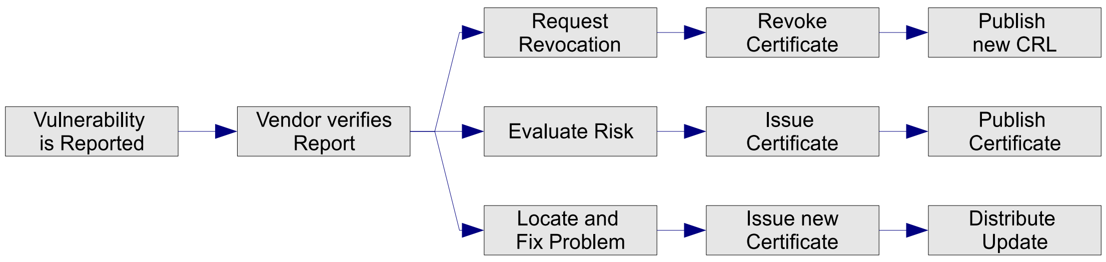

Fig. 4.2: Workflow to React on a Reported Vulnerability

If a vulnerability in a certified component is reported, the workflow shown in figure 4.2 is started. An important organisational issue is the publication of a suitable representative, who receives the reports. We do not want to discuss this issue further, but assume that the report is received at a central instance. Before any further actions are taken, it is at the responsibility of the vendor who issued the certificate, to verify the correctness of the report. If the presence of a vulnerability is confirmed, the further processing splits into three branches. The first and most urgent duty, is the revocation of the certificate and the distribution of updated revocation information to prevent the usage of the vulnerable component. Even if we have different signing authorities with their own keys, and it would be covered by the specification that every signer creates his own CRL, it is sensible to accumulate all revocations in one central list to save processing time and network load of the CRL freshness checks and updates. Therefore, the authority responsible for the “wrong” signature requests its revocation at a central authority, which then issues the new revocation information.

Based on the evaluation of the risk which results from the vulnerability, it might be feasible to issue a new certificate with a lower trust level, to allow further usage of the vulnerable component at least for data with low protection needs. This step is optional and not necessary at all, if a fixed component is available fast enough.

To finally receive a fix for the vulnerable component, it is necessary to allocate resources to prepare a fix and afterwards distribute the update component together with a new certificate to the end users.

4.1.3 Technical Organisation

Root Key and Intermediate Certificates

The intermediate certificates to verify the vendor keys are a small number, which we can efficiently store in a central place in the file system. The root key, and a subset of the intermediate certificates to verify the kernels certificate, are necessary already before the system is operational, so we must put them into a place that is reachable by the bootloader.

The openssl toolkit [10] organises the certificates simply as files in a special directory and uses a hash over the issuer name to speed up seek operations. The toolkit is an accepted and proven solution that suits our needs and we therefore decided to reuse its concept.

Certificates which are necessary in a verification step but not present on the system can be fetched from a central LDAP repository, which is provided as a part of the defined infrastructure. The connection information to reach the LDAP server can either be included into the certificates or globally defined by configuration options.

Entity Certificates

The entity certificates are stored aside the certified entity using the extended attributes of the filesystem. This provides us fast access to the certificates during the verification procedure and makes an additional management system unnecessary.

As the used implementation of the bootloader does not support reading from the extended attributes, we additionally store the certificates which are needed before the kernel is started as normal files in the configuration directory of the bootloader. The relation between a file and the corresponding certificate is defined by keywords in the configuration file of the bootloader.

CRL Binding Certificate

The CRL binding certificate must be accessible by the bootloader and is therefore stored as individual file and referenced by a keyword, too.

4.2 Boot of the Initial Kernel

When a conventional system is powered up, the Basic Input Output System (BIOS) looks for a bootable media and executes the initial bootloader. Through one or more stages the operating system is loaded into the computers memory and executed. The finally started operating system is in the unfortunate situation that it can not judge on its own history and therefore can not make any assumptions on its own integrity or the hardware it is running on. This section describes a modified boot procedure that reflects the complete boot history through PCR values using the new digest creation method.

4.2.1 Starting the Initial Bootloader

Starting at the Core Root of Trust Measurement

The core root of trust measurement (CRTM), a component invented by the TCG, measures itself and afterwards measures and loads the remaining parts of the BIOS. Further components of the system hardware are also measured before finally the code in the master boot record (MBR) of the boot media is measured and executed. The MBR contains the first 512 bytes of the bootloader code, which are called stage1 in our example. As shown in table 4.3, the measurement value of these 512 bytes is written into PCR8. Further information on the measurement of the hardware components is given in 2.3.3.

PCR Index	Usage
PCR 0-7	Hardware
PCR 8	Grub stage1
PCR 9	Grub stage2
PCR 10	root key
PCR 11	CRL
PCR 12	kernel & policy identifier
PCR 13	trust level identifier

Fig. 4.3: PCR Assignment

The measurements of the hardware and of the MBR are done by components which are controlled by the platform manufacturer and not accessible for our work. We therefore could not implement an alternative measurement method here.

Loading the Initial Bootloader

Our prototype uses the GRand Unified Bootloader GRUB [17], a mutli-stage bootloader that supports a large spectrum of filesystem and kernel formats. The extended version Trusted-GRUB [52] adds routines to measure its own stages as well as the files of the loaded operating system into the PCRs in a TCG conformable manner.

The boot process and the associated code is split into three parts. The first part is the code from the master boot record, which is already measured by the TCG extension of the BIOS and is considered trusted. This code, named stage1, locates, measures, and executes the beginning of the upcoming stage2. Due to technical limitations, stage1 is unable to access data beyond the first sector of stage2 which makes it necessary to split stage2, at least logical, into two parts. This results in a stage2 binary, where the first 512 Bytes, that are measured and executed by stage1, contain again all code that is necessary to measure and execute the remainder of stage2. Within the TrustedGRUB project these two measurements are realized using the TPMs built-in hash functions and due to the very limited available size, it seems infeasible to implement a more sophisticated measurement without having additional trusted resources.

While stage1 and the beginning of stage2 are mainly for circumventing technical difficulties in running the bootloader, the remainder of stage2 contains the actual bootloader system. The stage2 code of the TrustedGRUB extension already contains measurement routines to calculate and write the hashes of the kernel image into the PCRs. We modified these routines to use our certificate-based digest creation methods which enables a system maintainer to exchange the kernel image without changing the associated digest value.

4.2.2 Setting up the Verification System

To verify the signature of the kernel, we first need to setup the verification infrastructure, which consists of the root key and the current revocation list.

Loading the Root Key

The public part of this root key is now read from disk and its fingerprint is written into PCR10. This does not guarantee anything on the used key and there is no way to do a verification of it, but as the PCR value depends on it, sealing operations will fail and a challenger will receive an unexpected value if the key changes.

For the remaining processing stages we must ensure, that always the key stored and measured is used and that no one can change the stored key.

Loading the Revocation Information

In a standard X.509 public key infrastructure, the correctness of a revocation list is verified by its signature and a check on the validity interval. We explained in 3.2.3 why the absence of a network connection and an authoritative timesource can be used by an attacker to blind discovered vulnerabilities from the system and outlined an approach how to prevent such an attack. The central components of this rollback protection system are the CRL binding certificate, the corresponding signing key `CRLSign` and the TPM-based monotonic counter mechanism. To prevent any influences on this protection system from a compromised operating system, the manage-

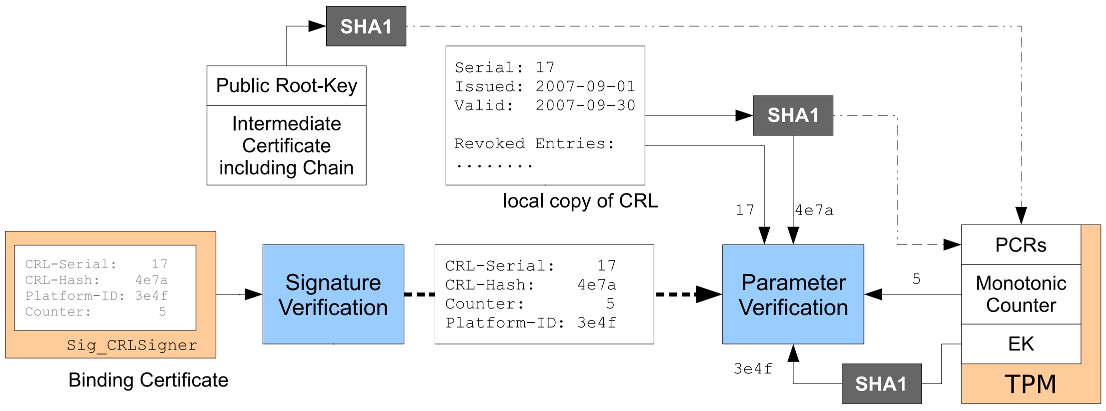

Fig. 4.4: Validation of the CRL Binding Certificate

ment of these binding certificate is done by a dedicated system, which we explain in 4.5 at the end of this chapter.

The binding certificate is a X.509 attribute certificate based on the custom profile given in A.1. The holder information is set to the SHA-1 hash of the CRL binary so the certificate is linked to a dedicated instance of the CRL. The certificate issuer field denotes the platform on which it was created by the hash of the public endorsement key while the serialnumber of the certificate reflects the value of the monotonic counter at the time of creation. As the monotonic counter is increased whenever a new CRL is invented to the system, this relation indicates if the present certificate is the most current one. The certificate is signed using the `CRLSign` key, which is unique for each platform and is verified by a set of certificates ending in the used root key as shown in figure 4.1.

Figure 4.4 gives a schematic view of the necessary steps to actually load the revocation information into the verification system. After checking the signature of the binding certificate against the previously loaded root key, the hash value of the CRL is calculated and verified against the holder field. The current platform state is expressed by the public part of the endorsement key EK and the value of the monotonic counter. Both items are read from the TPM and matched against the issuer field respectively the serial number field. In addition, we check if the current system time matches the validity window, which is necessary to have a reliable time reference for checking all further certificates. It is not necessary to validate the signature of the CRL itself, as this is done by the update system when the binding certificate is issued. If the validation of the binding certificate fails, we destroy the representation of the root key by writing an error value to PCR10.

As the given system can only prevent a rollback but not guarantee absolute freshness of the used CRL, we write its hash value to PCR11 to give a remote challenger the chance to detect its version.

4.2.3 Boot of the Initial Operating System

Load and Measure Kernel and Initial System

After setting up the verification system, we load the code of the kernel-image from the storage media into the systems memory and calculate its hash value. Before we execute the code, we check for a certificate and determine the resulting trust statement of the system.

Verification and Trust Assignment

As the kernel comprises of only one single file, which we can measure in a single step, we use the file-based method with a global key as explained in 3.2.1 and assemble the necessary management information into an attribute certificate according to our custom profile given in Appendix A.2. The `holder` field is set to the kernels binary SHA-1 hash value and provides the linkability between certificate and binary. At this stage, the second important item in the certificate is the attributes section, which contains two entries for the kernel certificate. The first entry designates the identifier, which should be used as the digest value for this special file while the second entry assigns the trust level. The `issuer` contains a pointer to the signers key, which is one of the vendor keys as depicted in 4.1. After we have verified, that the `holder` information matches the hash value calculated while loading the kernel into memory, we check the signature of the certificate. We use the `issuer` information from the certificate to select the correct signing key, check the signature, and verify the certificate chain up to the root key. Besides the verification of the cryptographic signature, we have to check if the system time matches the certificate's validity window and if the loaded revocation list does list it as revoked.

If all verification steps are completed without error, we want the PCRs to reflect the measured entity and the current trust level. The identity of the used file is represented by its digest value, which we extract from the attributes section of the certificate and write it into PCR12. Setting the trust level of the system can require more than one step. We reuse our example from chapter 3 with three trust levels and assume the kernel has trust level B assigned. The PCR value which represents trust level B results from a concatenation of the predefined identifiers for trust level A and B inside the TPM

$$PCR_{LevelB} = f_{PCR}(f_{PCR}(PCR_{Init} + Identifier_{LevelA}) + Identifier_{LevelB}).$$

It is now up to the bootloader to write both identifiers to PCR13. Once written, there is no way to force the PCR back to the state assigned to trust level A without rebooting the whole platform. As the key provisioning depends on the value of this PCR, it is physically impossible for an attacker to gain access to keys for items associated to trust level A.

We now execute the kernel's code from the system memory and pass control to the kernel.

Necessary Modifications to the Kernel

With the execution of the kernel code, the process of the bootloader terminates and its information, including the verified revocation information, is lost. The kernel becomes the central element in the measurement architecture of the running system, and therefore needs access to the components of the verification system. As the first measurement must be done already on the `init` process, before other userspace components can be set up, we must implement the verification system directly into the kernel.

The modifications to actually perform the measurements are placed into the memory mapper functions of the kernel and become part of the kernel binary while root key and revocation list are stored in separate files and read from disk on every startup. It is important to ensure, that root key and revocation information as loaded by the kernel are identical to the ones used by the bootloader and represented via the PCRs. As it is likely, that the kernel needs to load a driver module to access the TPM, the TPM might not be available early enough to verify the loaded items. To ensure that the items loaded and used by the kernel match the ones logged into the PCRs, we pass their hash values from the bootloader to the kernel using the kernel commandline and compare the values on load.

4.3 System Startup

The first element in the system startup phase is the initial system process `init` which is started by the kernel. It starts the kernel management and system services and executes configuration scripts to setup the systems devices.

As the exact configuration and execution order of those items may vary between different system configurations, we can not use the file-based measurement approach from 3.2.1 here but use a state-based one with an infinite execution time as discussed in 3.2.2 instead. The particular identifier of a started entity does no longer matter and the corresponding attribute in the certificates is obsolete and replaced by a policy identifier. The policy identifier used in the measurement as reference can be either implemented directly into the kernel or passed to the system as a parameter and is extended on top of the kernel's digest into PCR12.

4.3.1 Starting the Initial Process `init`

The initial process `init` becomes the parent of all further processes running on the system. It therefore has full access to their process structures while they are executed, so it is mandatory to include it into the trust level evaluation. Even if the init process is a very special process in the system, it also goes through the memory mapper functions and we can handle it the same way as all other processes. After the kernel has setup all of its internal structures, it reads the executable code for the init process from a file on disk, maps it into the memory and calculates the SHA-1 hash value of it. The certificate is read from the extended attributes section of the files inode and its holder information matched against the hash value calculated from the loaded code. Finally, the signature of the certificate is checked, using the root certificate and the revocation information which was previously loaded into the kernel. If the cryptographic verification succeeds, the policy identifier is read from the certificate and compared against the one used by the system. Before we can execute the loaded code, we determine its trust level from the certificate and, if necessary, adjust the trust level of the system accordingly.

4.3.2 Running Configuration Scripts

Configuration scripts are started by the init process to load necessary kernel modules and start the system daemons, which is explained in detail later, but also perform several other tasks like mounting file systems or initialising hardware devices. Our system and trust model is based on the assumption, that the trust level of an operation is determined only by the trust level of the involved processes.

As configuration scripts usually do not result in a long-lived process and do not get involved into a trust level sensitive operation, it is unnecessary to measure them.

4.3.3 Extending the Kernel Space with Kernel Modules

Immediately after the boot process has finished, the data inside the kernel space is determined by the kernel image which has been already measured by the bootloader and thus contains only trusted code. The concept of kernel modules allows the dynamic extension of this initial codebase during runtime. Once such a module was loaded into the kernel, it is fully incorporated into the code running in the kernel space and becomes part of the kernel. We therefore must ensure that loaded modules match the current policy and trust level of the running system. Adjusting the trust level downwards can lead to a violation of protection targets if the system is currently processing data with a higher trust level demand. To avoid such a violation, we maintain an internal table about all currently used resources and their trust level and either force the system to free affected resources before loading the module or deny its inclusion.

The actual implementation of the module measurement is based on the concept given in the article “Signed Kernel Modules” [36] which we modified to use our certificate based approach.

4.3.4 Execution of System Daemons

System daemons are basically normal applications and communicate with the kernel or other daemons via defined interfaces. As their code is executed outside the kernel space, they do not affect the security properties of the kernel directly. Nevertheless, it is necessary to know about their security properties, as soon as they get involved into an operation on protected data. We therefore take a measurement whenever a daemon is executed and store the results of the verification to use it in later decisions.

The kernel usually mediates access to its own interfaces as well as communication between processes through a discretionary access control model based on user and group membership. We extended this model with the attribute trust level, which is set to the result of the performed verification, and evaluated by a new ruleset for every access decision. The rules dictate, that a system daemon can only bind to the kernel or communicate with another system daemon, if they share the same trust level. The trust level of the kernel is the one of the core system, which is the highest possible level of all other processes, too. Decreasing the trust level of a daemon is possible if it currently has no binding to resources which contradict the decrease.

A detailed description on application measurement is given in 4.4.1 in the next section.

4.4 Protection System in the Application Space

The key feature of the invented protection system is the trust level based access management which obviously requires knowledge about the trust level of the requesting application. As the application can not interfere with the core operating system and influence the access control system, we assume the kernel is capable of blocking access requests from applications to resources with higher trust levels and it is not necessary to adjust the physical protection mechanisms depending on the applications trust level.

4.4.1 Trust Level of Applications

Initial Measurement on Startup

The measurement code is placed into the memory mapper functions of the kernel which is a perfect place as every code passes by just before it gets loaded into the system memory and executed. At the moment when the launch of the application is requested by the user, the code's SHA-1 hash value is calculated while it passes through. We extract the pointer to the inode on the disk of the loaded binary from the internal process structures and use it to load the corresponding certificate, which is stored aside the binary using the extended attributes feature of the ext3 file system. We perform the usual verification steps on the certificate and extract the trust level from it, which is then stored in a lookup table using the id of the started process as key. As parts of the operating system can inherently access and manipulate process structures, and are finally responsible to enforce the rules based on the trust level, it is useless to set the trust level of an application higher than the one of the core system.

Dependency on Configuration Options

In our trust model, we assume that the trust level of a process is determined by the binary from which it is created but does not depend on any configuration options passed to it via the command line or a configuration file.

On current systems, there are several system components where this assumption does not hold. For example, an attacker with access to the password database file can easily add a new user or modify the password of an existing one. This will not result in a compromise of the system itself and is not detected by our verification systems, but enables an attacker to access protected resources.

As the configuration options are usually platform dependant and must be adjustable by the local administration staff, we can not solve this problem with global property certificates based on the hierarchy given in 4.1. At the moment, we can not provide a feasible, general solution and therefore leave the implementation of countermeasures to the affected applications.

Dependency on Shared Libraries

If the application depends on shared libraries, we will see their code passing through the memory mapper shortly after executing the application and can detect the origin of the request through the id of the calling process. As the library code gets included into the application, it fully contributes to the security assumption. Each library is measured in the same way than the application itself and the trust level in the lookup table is set to the lowest value of all measured items.

Communication with System Daemons

System daemons provide a multitude of functionalities and it is impossible to make a general decision if a communication with them is suited to violate a security property. A solution to this problem would require a detailed analysis of the kind of information that flows between application and daemons and a policy that defines which kind of information needs which kind of protection. As current systems can not provide such information, we have no handy solution for this problem and therefore must consider each communication as a potential risk for our protection properties. This leads to the very rigid requirement, that we either block access to system services, if their trust level is below the one of the application or decrease the application's trust level which might be impossible due to opened resources.

4.4.2 Protection by Encryption

The basic protection mechanism of our system is an item based encryption using the eCryptfs [29] filesystem. Each file gets encrypted using a unique key which is sealed to the TPM under the selected trust level.

Sealing Precondition

The seal of each key is bound to the PCRs 0 to 10, 12 and 13 which represent information about the used root key, the identifier of the kernel, the used policy, and the systems trust level as seen in table 4.3. As long as we assume that no vulnerabilities are present in the files of the core system, there is no way for an attacker to gain unauthorized access to the protected resources even with full physical control over the platform.

Key Provisioning

We presented two different methods to manage the keys for such a system in 3.5.3. Within the boundary conditions of the presented implementation with three trust levels, the management overhead using the key duplication method shown in figure 3.4 seems to be an acceptable price for the gained resistance against a key leakage. The publicly available eCryptfs code already comes with support for TPM based keys so we do not have to spend any additional efforts to enable access to existing resources. The available code also handles the creation of new keys which we modify to populate the key at the different trust levels as foreseen in the proposed key management mechanism.

Before we can seal the key, we need to know the parameters for the sealing operation. These are the selection of PCR registers and their expected values, a storage key from the hierarchy, and an optional passphrase, which we will omit in our case. The registers we assign to the seal are fixed and their values are publicly known, as they are used in the remote attestation, so their publication does not impose a security problem. Nevertheless we must protect this information against changes, as using the wrong registers or wrong values might grant access to the sealed information in an unintended platform state. Finally, we need to know all trust levels at which the key should be available as each one has its own value of PCR13. As we pointed out in 3.5.1, the publication of data across the border of a trust level has effects on integrity and confidentiality of the information. We therefore think, that it is a bad idea to allow the user an arbitrary selection of the trust level. For the moment, we take the trust level currently assigned to the originating application, but we will have a brief discussion on this topic a bit later.

We now take the newly created key and seal it to the target trust level and all superior levels using the collected information. Besides, we use the key to encrypt the identifier of the target trust level and write the plain and encrypted form to the header of the eCryptfs file as shown

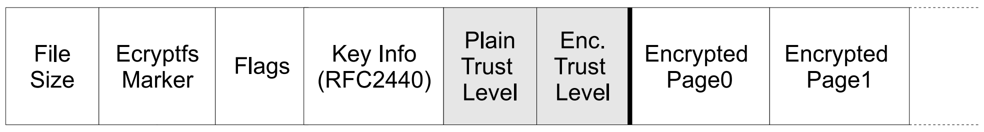

Fig. 4.5: Structure of the eCryptFS Fileheader with added trust level fields (grey)

in figure 4.5. Afterwards, we pass the key to the original code to encrypt the payload. The information about the trust level serves two purposes. The plain version is used only to speed up the decision, if a file can be decrypted on the current trust level or not. Instead of doing an expensive search for the key stored under the TPM, it is sufficient to check if the identifier is known to be accessible from the current configuration. An attacker might change the entry but can launch at least a denial of service attack by either forcing unuseful key searches or preventing access to the file even if the trust level is sufficient. The encrypted version is useful, if we need to reliably know the trust level from which the item originates.

4.4.3 Resource Access by an Application

Read Access to an Existing Storage Item

The decision wether an application can access an existing item on disk depends on the assigned trust level of the application and the requested resource. How to determine the trust level of both was mentioned in the previous paragraphs and the comparison of both is a simple operation. If the preconditions are fulfilled, the access control accepts the request and advises the protection system to load the key and offers a file handle to the decrypted information.

Write Access to a Storage Item

The decision to allow a write access to an item is more difficult than for a read. The problems arise from the fact, that read across the borders of a trust level is possible and that the trust level assigned to a running application does not reflect the nature of the handled content. We can divide the possible situations into two groups, whether the data item is stored with the same trust level or not.

If we want to store an item on the same trust level as it was opened from, we are faced with two kinds of problem. First, we must track the used resource or at least store the trust level of it, to assign the correct trust level when writing the item. Furthermore, we must ensure that the application can not open items from a higher trust level within the same processing space and transfer information from it into the other resource. We can provide the expected by either manipulating the application itself to deal with the trust level property internally and obey these rules or we readjust the trust level of the application to match the one of the opened resources on every read. This will prevent any transfer of material between trust levels but also reduces the convenience of the user, as it forces him to close and reopen the application, if he wants to access items from different trust levels.

If a user deliberately wants to transfer information from one trust level into another, the operation will affect either the confidentiality or the integrity of the information. According to Biba [2], the integrity is affected, if we move information to a higher trust level. Due to a technical feature of the sealing mechanism, we can deal with such situations. Each seal stores information about the system state when it was created and we therefore gain knowledge about the system's trust level at the time of the write operation. If integrity is a concern, we additionally demand that the selection of the target trust level is done by user interaction using a component running on the system's trust level. This way we ensure that the trust level recorded in the seal reflects the circumstances under which the trust level was selected.

Moving information downward, endangers its confidentiality (Bell-LaPadula [8]), which we could not compensate in the case of a security relevant breakage. We assume the legal user as a trusted party who is able to judge on the protection demands of processed information and therefore allow him such a transformation of trust levels without further conditions. We also need no additional measures against fraudulent attackers, as they first need to access the information on the originating, higher level and there is no benefit for them to lower the trust level after they already have access.

4.5 Maintenance of the Revocation Information

The ability to react on discovered vulnerabilities depends on the revocation mechanism included into the X.509 infrastructure. The availability of the most current revocation information at the verification system is critical for an adequate reaction in either operation phase of the system.

4.5.1 Timeliness in a Networked System

While the system is operational and has access to an authoritative source of revocation information like a CRL distribution point, a service supporting the online certificate status protocol or the like, we can simply query this source for every single revocation request respectively store and refresh the information inside the protected memory of the verification system. Besides the verification of all newly started applications, it is also necessary to check if the certificate status used to evaluate running processes changes. This becomes especially important for the core components on systems with large intervals between two reboots.

4.5.2 Preserve Updated Information Across Reboots

To preserve the updated information after a reboot cycle, we need to store it in a non-volatile storage. As already mentioned in 3.2.3, a special challenge for our system is the protection of this revocation information against a rollback attack and we already gave an outline for a suitable protection mechanism using the TPMs monotonic counters.

Updating the Binding Certificate

Updating the revocation information consists of five steps. The updated information is fetched from a distribution point and afterwards verified for authenticity and checked if it is newer than the data already stored locally. If the checks succeed, the monotonic counter is incremented and a new certificate is issued, linking the incremented counter value and the new revocation information together. The final step is the distribution of the new information to the verification system which is done in our implementation proposal by replacing two files in the configuration filesystem of the bootloader.

Requirements on the Certificate Issuance Process

One central requirement of our architecture is the sane recovery of the verification system after a compromise of the main operating system which includes a guarantee on the freshness of the revocation information. This makes it mandatory, that the potentially compromised system is not involved in the creation of the mentioned binding certificate, as otherwise an attacker can tamper with the revocation information. Therefore, we must delegate the certificate issuance to an external process.

External Signing Authority

Using an external signing authority seems a feasible approach but fails due to limitations in the current TPM specification and the TCG Software Stack TSS. To issue a new certificate, the authority needs to know the current value of the monotonic counter and a reliable feedback if the demanded incrementation was successful. Unfortunately, reading the counter value from the TPM is possible only as a plain number without any proof of origin or correctness. The TSS allows a secure connection of a remote party and also a proof of the local platform state via remote attestation, but assumes that the TSS is trusted and does not tamper information between the communication channel and the origin. This assumption is contrary to our demand to keep the operating system out of the certification process.

The transport session feature of the TPM allows the verification of a whole communication session by a signature and can be modified to create a provable channel between the monotonic counter and the remote signing authority. However, this requires a large deviation away from common practices and the current specification and will reveal a lot of platform internals to the signing authority and therefore affect privacy concerns. Besides, such a protocol requires either a network connection or a costly exchange of several messages via a portable data storage.

Local Signing Authority

A local signing authority avoids all these problems and provides a convenient way using current standards. The complete process for issuing a new binding certificate does not require any human interaction, if the location of the update information is well known and we can simply pack the whole signing authority into its own miniature operating system which runs independently from the main system. On legacy platforms, this requires that we shutdown the original system and boot into the updater system while we can run both in parallel if the platform provides a virtualisation layer with strong isolation.

The signature on the binding certificate must be verified through the root key of the system and therefore we need a certificate chain from the used signing key up to the root authority as shown in figure 4.1. Besides, we must avoid any usage of the signing key outside the updater system, as this would allow a forgery of the revocation information. In the upcoming paragraphs, we give a detailed setup procedure how to create, store and certify the signature key, so the demands are fulfilled.

4.5.3 Setup of the Local Signing Authority

To effectively protect the signature key used for the CRL binding certificates from abuse, we use a TPM based key bound to a PCR state which is only reachable by a legal CRL management system. The management system itself consists only of a small kernel and a ramdisk image, so we can efficiently measure it using the snapshot verification approach directly from the bootloader.

After the initial deployment of the management system we must create a key and obtain a certificate for it from a "CRL Key CA". The bootloader is configured to perform the measurement the same way like on a normal system start, but omits the check of the revocation list, as it does not exist at this moment. As soon as the system is running, a key bound to PCR0 to PCR12 is created using the TPM_CreateWrapKey command. As the key is supervised by but not stored inside the TPM, we have to export the encrypted key packages and store it in some non-volatile storage outside the system. The system image itself must be read-only as otherwise the hash will change with the key update thus blocking access to the key.

We now initialise a monotonic counter of the TPM and create a first, empty, binding certificate and write it into the certificate storage area used by the bootloader. Afterwards, we create a certification request over the key to obtain an intermediate certificate linking our newly created, platform specific signing key to the global root key. The whole certification request object is then signed by the TPM_Quote command using the endorsement key or a previously created attestation identity key, including the values of the PCR0 to PCR12. As the CRL update system has no network connection, we cannot dispatch the request directly to the superordinate signing authority and thus store it along with the key blob into a dedicated storage area.

Afterwards, we shutdown the CRL management system and start another system which provides network support. This system does not necessarily have to be trusted or use the TPM at all and just has to forward the certification request to a signing authority of type "CRL Key CA". Based on the signature of the outer envelope, the authority can verify that the request was created using an AIK or EK of a genuine TPM while the list of PCRs reveals that the CRL management system was running at this time. The authority now issues the intermediate certificate for the platforms "CRLSign" key and sends it back to the requestor, where it is written into the well known location of the verification system and closes the gap in the certification chain.

Chapter 5

Analysis of Gains and Efforts

This chapter contains mainly a summary of the conceptual work and the sample implementation, as already presented in the two former chapters. In the beginning, we evaluate to which extend the features of the given implementation match the expected objectives stated in 1.6.1. The middle section summarises the efforts spent on the implementation of the proposed protection system, both on the technical level and from an organisational point of view. In the last section, we finally examine the exposure of the new components against local and remote attackers. We rate the propability for an exploitable vulnerability based on the added code and estimate the consequences and the efforts to fix the problem, if such an exploit really happens.

5.1 Achieved Improvements

The main objective of this work is a protection system for digital items based on the Trusted Platform Module's encryption abilities, which is useable under the assumptions made by Arbaugh's lifecycle model. A subordinate objective is the enhancement of privacy and manageability aspects in remote attestation scenarios. This section shows the reached coverage of both objectives.

5.1.1 Protection of Data on the Local System

Superseding Obscurity by TPM-based Protection

As shown in 1.3, the protection of digital items nowadays mainly relies on obfuscation which is provably not a secure method. The functions provided by the Trusted Platform Module to seal data or keys to a defined system state are suited to protect a system from changes made to system components stored on disk. The certificate-based digest creation method, described in 3.1, eliminates the system immanent deficiencies given in 1.5.3 and enables us to use the TPM under the influence of Arbaugh's lifecycle model [1]. With the TPM as secure key storage, we can avoid the problems given in 1.3.2 arising from a fraudulent user and replace obscurity by provable cryptographic procedures.

Managing Risk on Disclosed Vulnerabilities

The multilevel security access model provides us with a comfortable tool to regulate access to resources based on the currently assumed trust state of the core system and individual components. The recurring discovery of new vulnerabilities in used components make the vulnerability exposure, and thus the trust assumption, a transient value over time. If a vulnerability is disclosed, it can be necessary to adjust the trust statement given to an individual component or the whole system. The invented measurement procedure supports such an adjustment by revoking and reissuing property certificates. The trust level in applications is enforced in software by the access control system, which does not impose a problem as long as the core system is not affected by a vulnerability. If the trust level of the core system needs to be decreased, the changes will be visible in the platform configuration registers which are used by the key provisioning system. This breaks the sealing precondition for all items on higher trust levels and therefore makes access to them physically impossible.

Limited Damage in the Case of a Compromise

In the unfortunate case, when an attacker exploits a known or unknown vulnerability and gains unauthorized access to the system, the invented protection system limits the possible damage in several ways.

First of all, we must distinguish if we have a partial compromise in the application space or a full compromise of the core system. In the first case, the attacker can only access items which are on or below the trust level of the compromised application. This will efficiently block each access by additionally installed malicious software, as such does not possess a valid certificate and therefore gets no trust level assigned. If the core system is compromised, the logical access control of the system becomes ineffective, and the attacker gains access to all items, even if he uses untrusted software. Compared to a system, where the operating system's access control is the only effective instance, we still have two limitations enforced by the TPM. First, the item based encryption makes a bulk copy of data items useless while the low operation speed of the TPM limits the number of decryptable items per time. The second advantage is only effective, if the system's trust level was lowered based on a risk management decision prior to the compromise. In such cases, the degraded trust level recorded in the TPM will reliably prevent access to all items on trust levels higher than the one assigned to the vulnerable system.

5.1.2 Benefits in Remote Attestation Scenarios

Benefits for the Challenged Party

The effects on the privacy of the challenged party by a remote attestation are an unsolved problem in the scope of the current TCG specification. The problematic part in the attestation is the misuse of the submitted measurement values, which reflect detailed information about the used software and hardware stack. As elucidated in 1.4.2, this knowledge can be abused to track or discriminate the user or, in the worst case, to identify vulnerable components and launch a well targeted attack. The digest creation method invented in this work condenses the submitted information and reduces the variety of possible combinations, making it impossible to draw conclusions about the actual running system configuration. Assuming a representative large amount of users, a challenger is no longer in the position to either identify a particular platform or even link subsequent transactions to the same origin.

Simplified Evaluation of Attestation Results

The decreased number of possible configuration values does not only enhance the privacy of the challenged platform but also provides a benefit for the challenger, too. It is no longer necessary for him to deal with hash values of individual binaries and perform complex calculations to find legal combinations thereof. Instead he just has to evaluate if he accepts the combination of root key, running kernel, policy identifier and trust level as suited for his purpose. To ensure that the remote system is not running vulnerable software unintentionally, he must additionally check the timeliness of the used revocation information.

5.2 Efforts

The first part of this section briefly summarises the efforts, which are necessary to implement the system as described in chapter 4. The efforts for an implementation on other system platforms are considered to be comparable. In the second half, we outline the organisational and technical efforts which are necessary for a working certificate management.

5.2.1 Implementation at the Target System

Bootloader

The TrustedGRUB project [52] already provides methods to calculate the SHA-1 hash of files, an interface to write to the platform configuration registers of the TPM and the ability to read files from a storage media. Starting from here, we are missing functions to parse ASN.1 structures and to verify RSA signatures. Suitable code can easily be extracted from other projects. As the bootloader has only few dependencies on other components of the system, we do not have to spent much efforts on compatibility issues here.

Runtime

The necessary changes to components of the runtime system have a considerably large footprint but can be done with reasonable efforts as the data flow model of the system allows us to concentrate all necessary changes inside the system's core components. The efforts to implement the item encryption and the binary measurement are comparable low as we can reuse existing code and profit from existing interfaces.

The access decision component however requires a complex system, which can trace dependencies between different processes and manage all information necessary to control the trust levels of applications and items. As such functionality is not foreseen in the systems, we have no fixed interfaces but must modify components of the kernel directly. The necessary programming work is noteable but does not raise any technical or conceptual problems.

Vulnerability Management

The periodic update of the locally stored revocation information is not a complex task and easy to be implemented. A larger effort is necessary to implement the creation of the binding certificate, as such a component does not exists yet. However, functional modules for the different steps in the workflow are available and assembling them requires just some programming experience.

5.2.2 Certificate Management

Organisational Issues

The organisational cornerstone behind the certificate-based digest creation is the common policy document and its acceptance by all involved authorities. The policy contains one rulesets to define how we assign a trust level to an item and second, information about the criteria that must be meet when sharing an identifier. Depending on the number of involved parties and the desired granularity of the used rules, the complexity, and with it the necessary efforts, of such a policy document can become quite large. However, a system with only one trust level is easily created using existing processes and relations and already enhances the reliability and effectiveness of the protection system significantly.

One dominating factor for the quality of our protection system is the time elapsing between the disclosure of a vulnerability and the revocation of the affected certificate. A fast deployment of a fixed component will satisfy customers and keep the damage caused by the unavailability of resources low. Both requires the permanent availability of skilled developers to first verify and than prepare a fix for the vulnerability and, where appropriate, define the parameters for an interim certificate about the affected component. The remaining steps from the workflow shown in figure 4.2 do not require skilled personnel and run mainly without human interaction.

The necessary efforts to build a suitable certificate lifecycle management system are dominated by the preventive allocation of resources to react on new vulnerabilities.

Technical Implementation

Neither the workflows nor the cryptographic components of our proposal are uncommon or a new challenge from a technical point of view. Systems to perform resource allocation and coordination during software development as well as vulnerability tracking are widely deployed today but some efforts might be necessary to interconnect the systems of the involved parties.

The management of certificates with all surrounding elements is a typical task for a PKI-infrastructure and several solutions are out on the market providing the necessary functionality. Workflows and organisational structures are predetermined by this concept and the rules given in the policy document. The handling of key material and the operation of critical infrastructures anyhow is already part of the daily work of software distributors. We therefore do not have to expect noteworthy efforts or complications here.

5.3 New Threats Arising from the New Components

The newly invented components raise new threats, which we want to point out and evaluate in this section. We assume that, where applicable, any interference with external processes is prevented by the operating system. Therefore the only relevant possibility to compromise one of the components is a vulnerability which can be exploited by manipulated input data. To evaluate the possible threats, we analyse the points where user provided data enters the system and judge on the propability for an exploitable problem within the data path, based on the amount and complexity of added code.

5.3.1 Bootloader

The bootloader consumes two types of user provided data, binaries and certificates and revocation information. The only new processing step, which happens directly on the loaded binaries, is the calculation of their SHA-1 hash value. Due to the low complexity and small footprint of this step, there should not be vulnerabilities in this data path.

The handling of the certification information is far more complex and results in a large implementation footprint, but the nature of the handled data reduces the possible intrusion points. In a first processing step, the binary data is decoded which demands a well formed ASN.1 structure. Implementation problems in the ASN.1 parser component are possible but due to its simplicity unlikely. The second step in the processing chain is the signature verification where the most likely candidate for an exploitable vulnerability is the multiprecision arithmetic. A strong inspection of this code should abolish possible implementation mistakes here. Beyond this step, the presence of vulnerabilities is likely, but an abuse through manipulated data packets is nearly impossible, as packets without a valid signature will be rejected before they can reach the critical points. If forgery of a signature becomes possible, it is useless to care any longer about the integrity of the verification system as the attacker can create arbitrary certificates.

Even if a compromise seems to be unlikely, the dramatic consequences require extraordinary diligence when creating this component. A vulnerability might lead to the execution of arbitrary code while the TPM still assumes the operation of a trusted system. As the bootloader itself is measured by the hardware, it is impossible to exchange the code without loosing access to the protected data.

5.3.2 Infrastructure for Revocation Information

Binding Certificate Creation

The management system responsible for the creation of the binding certificates is a fully autonomous and isolated system. The only way to interfere with it, is the provided update information, which is processed like previously described in 5.3.1 for the certification information during the bootloader phase. The assumptions made there, regarding the location and propability of a vulnerability are similar for the management system. What differs are the possible consequences, in the case of a compromise.

Even if an attacker manages to exploit a vulnerability and gains full access to the management system and the local CRL signing key, the worst thing he can do, is the creation of an empty revocation list. To convert this into an attack against protected data, he further needs a copy of a vulnerable binary with a valid certificate and sufficient access to install this defective component. This enables him to circumvent, at least parts of, the local protection system but can not hide the presence of the vulnerable component from a remote challenger.

Although we assume an exploitable vulnerability a very unlikely event, we take precautions to react on it. Fixing the problem is as easy as deploying a new image for the management system and perform the initialisation procedure given in 4.5.3. To prevent an attacker from rolling back to the old image, its binding certificate must be invalidated. As the vulnerable image is installed on many different systems, where each one has its own certificate, a revocation of the individual binding certificates would cause a very huge revocation list. A more efficient way to handle

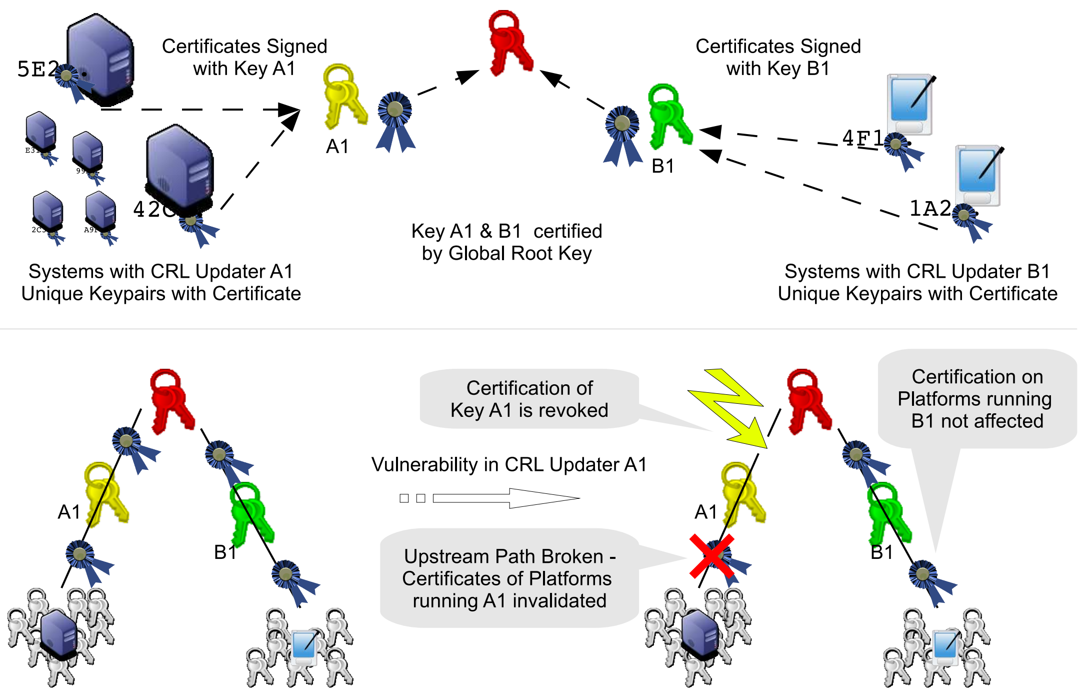
5E2
Certificates Signed with Key A1
A1
42C
Systems with CRL Updater A1 Unique Keypairs with Certificate
Key A1 & B1 certified by Global Root Key
Certificates Signed with Key B1
4F1
1A2
B1
Systems with CRL Updater B1 Unique Keypairs with Certificate
A1
B1
Certification of Key A1 is revoked
Vulnerability in CRL Updater A1
Upstream Path Broken - Certificates of Platforms running A1 invalidated
Certification on Platforms running B1 not affected
A1
B1

Fig. 5.1: Hierarchy of Keys for the CRL Updater Systems and Reaction on a Compromise

such a situation is possible, if a key hierarchy as shown in figure 5.1 is used. Each version of the updater system has its own signature key assigned and all certificates for the individual platform keys are signed with it. If all certificates belonging to a particular image version need to be revoked, we simply revoke the authority key which renders all subordinate keys invalid.

Blocking Updates of the Revocation Information

A denial of service attack against updates of the revocation information is relatively easy to conduct by cutting down the network connection to the authoritative servers. It does not create any harm itself but as soon as a vulnerability is discovered, an attacker can abuse the public knowledge about it to infiltrate the system. As the system does not participate on the new knowledge about the vulnerable component, it willingly hands out the protected items to the attacker.

5.3.3 Digest Creation for System Entities

Whenever an application is started, its code is measured using a SHA-1 function to later find the corresponding certificate and finally create the digest.

Measuring the Binary Code

The hash calculation is done inside the kernel space and therefore a potential candidate for a compromise of the kernel space using a crafted binary. However, as the hash calculation destroys the structure of the code, and we can safely assume that the hash calculation routines used in the kernel are not prone to a buffer overflow attack or the like, we do not expect a vulnerability in this context.

Creating the Digest

To create the digest from the calculated fingerprint, we have to load and verify the corresponding certificate. A compromise of the kernel space is possible in anyone of the involved subsystems which are several filesystem layers, the ASN.1 parser or the signature verification code. As already explained for the bootloader and the CRL management system, vulnerabilities beyond the signature verification are usually not exposed to an attacker.

The used filesystem functions are already present in the kernel, so if they are affected by a vulnerability, it is likely that other subsystems are also usable to break into the kernel. For a judgement on the certificate parsing and verification functions, we can reuse our argumentation used on the bootloader in 5.3.1. We can therefore assume, that the digest creation code does not introduce new threats into the system.

5.3.4 Access Control System

The access control system is the linchpin of the protection system and the file encryption routines offer the most critical path for an exploitable vulnerability. The data provided to the encryption routines can be arbitrarily chosen by the attacker and it is therefore possible to send special input which exploits a given vulnerability. However, as the encryption and filesystem code is not very complex, the propability for a vulnerability in these parts is likely not higher than in the other parts of the core system. The additional risk introduced with the changes made can therefore be neglected in terms of the overall risk. On the contrary, the invention of the TPM as key storage still offers limited protection in the case of a compromise, as illustrated in 5.1.1.

Chapter 6

Conclusion and Outlook

6.1 Review of Efforts and Gains

The implementation of the given system approach is possible with feasible efforts on the technical as well as on the organisational side. As shown in 5.2, a lot of structures already exist in toady's software landscape, which reduces not only the development costs but also lowers the risk of new vulnerabilities. The gained enhancements must be reviewed differently, depending on whether we take a software based protection system or one already using the TPM as reference.

Regarding systems with TPM support, we receive a nearly full coverage of the expected objectives. The improvements brought to the system with the new methods successfully handle the consequences of the lifecycle model as good as possible. The only limitation, we did not totally overcome with reasonable efforts, is the timeliness of the revocation information. As the implicated problems of this insufficiency were also present in the original solution, it does not create a new threat and therefore does not count as a negative effect. Even if the proposed new methods do not invent noteable additional propabilities for a vulnerability into the system, the chance for an exploitable vulnerability in the core system, that affects the TPM based protection system, remains a likely event. In an overall estimation, the multilevel security approach reduces the possible damage while the revocation system shortens the exposure time of known vulnerabilities so the overall risk of the system gets decreased.

Compared to a system which used obscurity, the judgement is ambiguous, as a formal evaluation means comparing apples and oranges. Better than a formal evaluation seems a comparison of the necessary efforts to circumvent the protection system. Assuming that the confidence we put into the used cryptographic methods holds, the efforts to break into a TPM protected system are equal to the ones necessary to break a relevant component of the core system. As such a breakage is only possible using a vulnerability, the estimation of efforts is closely related to the quality the code. For an obscurity system, the efforts can be accounted as the work we have to spent to reveal the obscure secret by either social attacks or reverse engineering. It is impossible to make a general statement about those estimations and therefore we can not recommend our system proposal as the preferred solution for any usecase. On the contrary, there might be situations, where the underlying system is of such a bad quality, that an obfuscation system is the better choice. We owe this effect to the dependency of cryptographic solutions on the reliability of the used operating system, while an obfuscation system always tries to shields itself from all influences.

When we make such a trade off, we should not underestimate the role of exposure time and the management aspects. As we have discussed in 2.1.3, it is impossible to react on a breakage of an obfuscation mechanism, while a vulnerability in a system component can be handled using our new methods. So even if the necessary efforts to reveal the obfuscation system are larger, it might be a more interesting target for an attacker than the abuse of a short-lived vulnerability.

Besides the problem of vulnerabilities in the used operating system, many available hardware implementations of the TCG concept are prone to two kinds of physical attacks. On many implementations, the TPM is an individual component and the wires of the LPC bus, which connects it with the rest of the system, are accessible from outside. With some modifications, it becomes possible to send a reset signal to the TPM, using the LPC reset wire, without triggering a reset of the platform itself. The result is a running platform with a freshly initialised TPM, which we can now set to reflect whatever values we want [51]. A second problem arises from insufficient protection of the BIOS, which contains the core root of trust measurement. Even if the specification demands a sufficient protection against unauthorized changes for the relevant parts of the BIOS, not all vendors obey these requirements. The work of Kauer [34] describes a successful attack against a current platform, which is sold as TCG conformable, by making changes to the BIOS code. Both attacks are possible to conduct without special equipment and knowledge and allow the forgery of values stored in the PCRs and therefore cause a collapse of the security base. It is needless to say, that a successful attack undermines our protection system as there is no chance to discover or prevent such a compromise neither from inside the running system nor from outside by a remote attestation.

6.2 Impact on User's Habits and User Acceptance

The major obstacle for the effective deployment of an active risk management system is its impact on the user's habits. The price we have to pay during a phase with a raised vulnerability exposure is the unavailability of certain items. Users might not accept the necessary changes of their habits to deal with such protective measures. The result would be either uncooperative behaviour with negative effects on the completion of business tasks or an intentional misbehaviour on the classification of objects. In the first case, the additional damage due to such secondary effects might raise the overall damage caused by the unavailability over the damage assumed in the case of a compromise. As a consequence, blocking access to the item is no longer the preferred choice. In the other case, information is not protected according to its real trust level but only by the one selected by the user, and an attacker might gain access to those items, even if the risk management system performed correctly.

To avoid those social problems, a well-founded education of the users is necessary to raise their awareness for security issues. Besides, a fast response and the immediate availability of a fix for discovered vulnerabilities will shorten the unavailability period and therefore keep the impact on the user habits low.

6.3 Future Developments

A very beneficial step would be the adoption of several aspects of the proposed concept into the next generation of TPM modules. An implementation of the proposed certificate-based digest creation method directly into the hardware should not require much efforts but makes the advantages usable for a larger audience. Especially the state-based digest creation, as described in 3.2.2, has some associated risks which can be eliminated by an implementation inside the TPM. If we finally can manage the availability of a reliable timesource and have the certificate revocation check inside the tamperproof hardware, we can even improve the problem with the revocation information's latency.

The movement of the TPM hardware itself directly into the mainboard chipset or the processor is frequently discussed and partially done by several manufacturers already. This will harden or even eliminate reset attacks and probably makes it possible to extend the role of the PCR registers as access credential to other resources of the system.

Besides modifications to the TPM and the used measurement procedures, it is essential to enhance the robustness of the software stack. On the one hand, we have to reduce the number of errors in the code by an improved software development process. Handy technologies are, for example, model-driven code generation and formal verification of code. On the other hand, alternative system architecture models can reduce the impact of an error or reduce the exposure of systems. Hardware-based virtualisation is one of the most current of such concepts and its benefits in conjunction with the TCG concept are currently exploited by the OpenTC project [54]. Last but not least, microkernel based system architectures are already under development for a long time and offer benefits from both worlds as they reduce the impact of errors and make a formal verification easier due to the small size of the components. In the past, the need for interoperability with legacy software and insufficient hardware resources hindered the raise of new architecture concepts but the ongoing demand for security and the cheap availability of high-performance hardware might give them a chance.

Appendix A

Additional X.509 Profiles

A.1 CRL Binding Certificate

```
AttributeCertificate ::= SIGNED{AttributeCertificateInfo}

AttributeCertificateInfo ::= SEQUENCE {
 version    INTEGER { v2(1) },
 holder     {
                digestedObjectType  2 (otherObjectTypes),
                otherObjectTypeID   OBJECT IDENTIFIER (custom definition),
                digestAlgorithm     SHA-1,
                objectDigest        BIT STRING (Hash of CRL-Object)
             },
 issuer      {
                digestedObjectType  0 (publicKey),
                digestAlgorithm     SHA-1,
                objectDigest        BIT STRING (Hash of public EK)
             },
 signature      RSA,
 serialNumber   INTEGER, (Value of the Counter)
 attrCertValidityPeriod AttCertValidityPeriod,
 attributes     {}
}
```

The above definition is based on the definition for attribute certificates as found in section 12.1 of ISO/IEC 9594-8:2005 [55]. The scheme shows only the values we choose to construct our binding certificate and ommits all alternatives. The attributes section is empty, as the fields `holder`, `issuer` and `serialNumber` in the certificate header already contain all necessary information.

The `otherObjectTypeID` is optional in the recommendation and can be ommited in our case, as the content of the field is clearly dedicated. To help a third party to determine the usage of the certificate and to allow a later extension of the system by other holders, a dedicated OID should be retrieved and included in the certificate.

A.2 Property Certificate

```
AttributeCertificate ::= SIGNED{AttributeCertificateInfo}

AttributeCertificateInfo ::= SEQUENCE {
 version          INTEGER { v2(1) }
 holder           {
                    digestedObjectType  2 (otherObjectTypes),
                    otherObjectTypeID   OBJECT IDENTIFIER (custom definition),
                    digestAlgorithm     SHA-1,
                    objectDigest        BIT STRING (Hash of CRL-Object)
                   },
 issuer            {
                     digestedObjectType  0 (publicKey),
                     digestAlgorithm     SHA-1,
                     objectDigest        BIT STRING (Hash of public EK)
                   },
 signature         RSA,
 serialNumber      INTEGER, (Value of the Counter)
 attrCertValidityPeriod AttCertValidityPeriod,
 attributes        Attribute
}

Attribute ::= SEQUENCE {
                         type      OBJECT IDENTIFIER,
                         values    SET OF AttributeValue
                         -- at least one value is required
}

AttributeValue ::= ANY DEFINED BY AttributeType
```

The list of attributes differ depending on the type of the certified item. Formally we have to define custom Object Identifiers and possible values in the used policy document.

List of Figures

Bibliography

[1] William A. Arbaugh, William L. Fithen, and John McHugh. Windows of vulnerability: A case study analysis. *Computer*, 33(12):52–59, 2000.

[2] Kenneth J. Biba. *Integrity Considerations for Secure Computer Systems*, 4 1977.

[3] Ernie Brickell, Jan Camenisch, and Liqun Chen. *Direct Anonymous Attestation*, 2004.
`http://eprint.iacr.org/`

[4] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. *sp*, 00:184, 1987.

[5] Intel Corporation. *Low Pin Count (LPC) Interface Specification*.
`http://www.intel.com/design/chipsets/industry/lpc.htm`

[6] Microsoft Corporation. *BitLocker Drive Encryption Technical Overview*.
`http://technet2.microsoft.com/WindowsVista/en/library/ce4d5a2e-59a5-4742-89cc-ef9f5908b4731033.mspx`

[7] PGP Corporation. *PGP Whitepaper - Revocation made Simpler*.
`http://download.pgp.com/pdfs/whitepapers/Revocation-SLCs_060104_F.pdf`

[8] Len LaPadula David Elliott Bell. Secure computer systems: Mathematical foundations. Technical report, 3 1973.

[9] Claudia Eckert. *IT-Sicherheit*. Oldenbourg, München [u.a.], 2008.

[10] Ralf S. Engelschall. *The OpenSSL Project*.
`http://www.openssl.org`

[11] Dirk Kuhlmann et al. *An Open Trusted Computing Architecture—Secure virtual machines enabling user-defined policy enforcement*.
`http://www.opentc.net/images/otc_architecture_high_level_overview.pdf`

[12] Hewlett-Packard Corporation et al. *Advanced Configuration and Power interface Specification Revision 3.0a, December 30, 2005*.
`http://www.acpi.info/DOWNLOADS/ACPIspec30a.pdf`

[13] The National Security Agency et al. *Security Enhanced Linux*.
`http://www.nsa.gov/selinux`

[14] S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization. RFC 3281 (Proposed Standard), April 2002.
`http://www.ietf.org/rfc/rfc3281.txt`

[15] Seth Schoen (Electronic Frontier Foundation). *Trusted Computing: Promise and Risk*.
`http://www.eff.org/Infrastructure/trusted_computing/20031001_tc.php`

[16] The Gentoo Foundation. *Gentoo Linux*, 05 2007.
`http://www.gentoo.org`

[17] The GNU Project / Free Software Foundation. *GNU GRUB GRand Unified Bootloader*.
`http://www.gnu.org/software/grub`

[18] T. Freeman. *Server-based Certificate Validation Protocol (SCVP)*, April 1999/2007.
`http://tools.ietf.org/html/draft-ietf-pkix-scvp-31`

[19] J. Butler G. Hoglund. *Rootkits - Subverting the Windows Kernel*. Addison Wesley, 2005.

[20] J. Galvin, S. Murphy, S. Crocker, and N. Freed. Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted. RFC 1847 (Proposed Standard), October 1995.
`http://www.ietf.org/rfc/rfc1847.txt`

[21] The Trusted Computing Group. *TCG PC Client Specific Implementation Specification For Conventional BIOS, Version 1.20, Revision 1.00*.
`https://www.trustedcomputinggroup.org/specs/PCClient/TCG_PCClientImplementationforBIOS_1-20_1-00.pdf`

[22] The Trusted Computing Group. *TCG PC Client Specific TPM Interface Specification (TIS), Version 1.20, Revision 1.00*.
`https://www.trustedcomputinggroup.org/groups/pc_client/TCG_PCClientTPMSpecification_1-20_1-00_FINAL.pdf`

[23] The Trusted Computing Group. *TCG Software Stack (TSS) Specifications, Version 1.2 Level 1, Part1: Commands and Structures*.
`https://www.trustedcomputinggroup.org/specs/TSS/TSS_Version_1.2_Level_1_FINAL.pdf`

[24] The Trusted Computing Group. *The Trusted Computing Group*.
`http://www.trustedcomputinggroup.org/`

[25] The Trusted Computing Group. *Trusted Platform Module (TPM) Specifications, Version 1.2 Revision 103, Part 1*.
`https://www.trustedcomputinggroup.org/specs/TPM/mainP1DPrev103.zip`

[26] The Trusted Computing Group. *Trusted Platform Module (TPM) Specifications, Version 1.2 Revision 103, Part 3*.
`https://www.trustedcomputinggroup.org/specs/TPM/mainP3DPrev103.zip`

[27] The Trusted Computing Group. *TCG EFI Platform Specification, Version 1.20, Revision 1.0*, 2006.
`https://www.trustedcomputinggroup.org/specs/PCClient/TCG_EFI_Platform_1_20_Final.pdf`

[28] The Trusted Computing Group. *TCG EFI Protocol, Version 1.20, Revision 1.00*, 2006.
`https://www.trustedcomputinggroup.org/specs/PCClient/TCG_EFI_Platform_1_20_Final.pdf`

[29] Michael Halcrow. *eCryptfs - An Enterprise-class Cryptographic Filesystem for Linux*.
`http://ecryptfs.sourceforge.net`

[30] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280 (Proposed Standard), April 2002. Updated by RFCs 4325, 4630.
`http://www.ietf.org/rfc/rfc3280.txt`

[31] E. Van Herreweghen J. Poritz, M. Schunter and M. Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical report, IBM Research Division, 2004.

[32] Jeffrey R. Jones. Estimating software vulnerabilities. volume 5 of *IEEE Security and Privacy*, pages 28–32, 2007.

[33] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003.
`http://www.ietf.org/rfc/rfc3447.txt`

[34] Bernhard Kauer. Oslo: Improving the security of trusted computing. Proceedings of the 16th USENIX Security Symposium, 8 2007.
`http://os.inf.tu-dresden.de/papers_ps/kauer07-oslo.pdf`

[35] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997.
`http://www.ietf.org/rfc/rfc2104.txt`

[36] Greg Kroah-Hartman. *Signed Kernel Modules*, 1 2004.
`http://www.linuxjournal.com/article/7130`

[37] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.
`http://www.ietf.org/rfc/rfc2560.txt`

[38] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97)*, pages 106–119, Paris, January 1997.

[39] EADS Corporate Research Center Philippe Biondi, Fabrice Desclaux. Silver needle in the skype. BlackHat Europe, 2006.
http://www.secdev.org/conf/skype_BHEU06.handout.pdf

[40] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000.
http://www.ietf.org/rfc/rfc2818.txt

[41] Joanna Rutkowska. Subverting vista kernel for fun and profit. Black Hat Briefings, 2006.
http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf

[42] Ahmad-Reza Sadeghi and Christian Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, pages 67–77, New York, NY, USA, 2004. ACM Press.

[43] University of Michigan Samuel T. King, Peter M. Chen. Subvirt: implementing malware with virtual machines. Security and Privacy, 2006 IEEE Symposium, 2006.

[44] Christophe Saout. *dm-crypt: a device-mapper crypto target*.
http://www.saout.de/misc/dm-crypt/

[45] Karl Scheibelhofer. Pki without revocation checking. 4th Annual PKI R&D Workshop, 2005.
http://middleware.internet2.edu/pki05/proceedings/scheibelhofer-without_revocation.pdf

[46] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[47] Secunia. *Vulnerability Report: Apple Mac OS X*, 2007.
http://secunia.com/product/96/

[48] Secunia. *Vulnerability Report: Gentoo Linux 1.x*, 2007.
http://secunia.com/product/339/

[49] Secunia. *Vulnerability Report: Microsoft Windows XP Professional*, 2007.
http://secunia.com/product/22/

[50] Secunia. *Vulnerability Report: Ubuntu Linux 7.04*, 2007.
http://secunia.com/product/14068/

[51] Evan R. Sparks. A security assessment of trusted platform modules. Technical report, 6 2007.

[52] Marcel Selhorst / Christian Stueble. *TrustedGRUB*.
http://www.prosec.ruhr-uni-bochum.de/trusted_grub.html

[53] trew. Introduction to reverse engineering win32 applications. *Uninformed Journal*, 1(7), May 2005.
`http://www.uninformed.org/?v=1&a=7&t=sumry`

[54] Technikon Forschungs und Planungsgesellschaft mbH. *The Open Trusted Computing (OpenTC) consortium*.
`http://www.opentc.net/`

[55] International Telecommunication Union. *ISO/IEC9594-8:2005 ITU-T Recommendation X.509*, 08 2005.

[56] Hartmut Pohl / Gerhard Weck. *Sicherheit in der Informationstechnik - Handbuch 1*. Oldenbourg, 1993.

[57] David A. Wheeler. *Secure Programming for Linux and Unix HOWTO*. 2003.
`http://www.dwheeler.com/secure-programs/`

[58] Edward N. Zalta. *Stanford Encyclopedia of Philosophy*. 03 2007.
`http://plato.stanford.edu/`

All internet links were last checked in February 2008.