Taming "Trusted Platforms" by Operating System Design

Ahmad-Reza Sadeghi[1] and Christian Stüble[2]

[1] Ruhr-University Bochum,
Institute for Information and Communication Security,
44780 Bochum, Germany
sadeghi@crypto.rub.de

[2] Saarland University,
Security and Cryptography Group,
66041 Saarbrücken, Germany
stueble@acm.org

Abstract. Experiences of the past have shown that common computing platforms lack security due to architectural problems and complexity. In this context, Microsoft Palladium (Pd) and TCPA are announced to be the next-generation computing platforms, and claimed to improve users' security. However, people are concerned about those capabilities of TCPA/Pd that may allow content providers to gain too much power and control over the use of digital content and users' private information. In this paper, we argue that TCPA/Pd can increase the security of computing platforms by faithfully designing the operating system. Moreover, we discuss how interferences between digital rights management capabilities and end-user security can be prevented. Our results are based on the fact that even with TCPA/Pd platforms the operating system has enough control over the platform to prevent misuse by both content providers and end-users.

We argue that such a trustworthy operating system, that is secure in the sense of multilateral security, can be developed without much effort by efficiently combining the ideas of security kernels and state of the art of operating system technology. We propose a new architecture for a trustworthy security platform that uses TCPA/Pd hardware features in conjunction with an open-source security kernel we have developed. Our security kernel provides backward-compatibility to the Linux operating system. The layered design and its lightweightness allows an easy migration to other hardware platforms like PDAs, mobile phones, and embedded systems.

1 Introduction

The advent of e-commerce and the rapid expansion of world-wide connectivity demands end-user systems that can guarantee authenticity, integrity, privacy, anonymity, and availability. While cryptographic and security research communities have provided solutions to a variety of security related problems, all these solutions depend upon the security of the underlying computing platform.

Existing computing platforms, in particular common operating systems, neither offer appropriate mechanisms to enforce adequate security policies, nor can they be maintained by non-experts. Additionally, they suffer from several vulnerabilities in hardware and software: beside architectural security problems and the inherent vulnerabilities resulting from complexity [31], common operating systems require careful and competent system administration and will still not effectively protect individuals from executing malicious code. This situation brings about the need for developing a new generation of secure computing platforms.

A lot has been reported about Microsoft's Next-Generation Secure Computing Base for Windows (NGSCB, former Palladium or Pd) [10,14] and the specification proposed by the Trusted Computing Platform Alliance[1] (TCPA) [36,35]. The stated goal of these systems is to improve security and trustworthiness of computing platforms [13,27,31,30]. However, since their announcement, there is an ongoing public debate about the negative economical, social and technical consequences of these platforms [3,7,32]. There are many concerns regarding their capabilities, in particular, in conjunction with Digital Rights Management (DRM). People are worried about the potential negative consequences of TCPA/Pd and believe that these platforms may give vendors and content providers too much control over personal systems and users' private information: they may allow commercial censorship[2], political censorship, or destroy innovation.[3] Especially, the open-source community seems to resist against TCPA/Pd, mainly because they are more sensitive regarding user security and privacy issues.

To clarify this situation, we briefly recapitulate requirements of secure end-user computing platforms and evaluate on a technical level to what extend common hardware architectures like PCs satisfy them. After a short review on TCPA/Pd based on available technical documentations, we analyze whether and how they can be used to improve end-user security of existing computing platforms. In this context, we argue that the functionalities provided by TCPA and Pd are completely under control of the operating system. As a consequence, users can benefit from the security features of TCPA/Pd, as long as their operating system is trustworthy. Although this statement seems to be obvious at first glance, the trustworthiness of existing and future operating systems is questionable.

Therefore, we propose a new architecture of a trustworthy security platform that uses TCPA/Pd hardware features in conjunction with an open-source security kernel [28] we have developed. Our proposed architecture demonstrates the following important issues. On the one hand, one can build highly secure

[1] www.trustedcomputing.org

[2] Microsoft, as the vendor of Palladium, is able to remotely delete documents that are locally stored on a Pd-machine.

[3] For instance, word could encrypt all documents using keys only known to Microsoft, making it impossible for rival products (e.g., OpenOffice or StarOffice) to be compatible or undermine the General Public License (GPL).

systems based on TCPA/Pd hardware without much effort. On the other hand, the security concerns mentioned above can effectively be prevented by a careful operating system design.

Furthermore, we propose design criteria to provide DRM features on our security platform that avoid conflicts between DRM policies and end-user security (privacy). In this way, the proposed trustworthy DRM platform fulfills security requirements of end-users and content providers in the sense of multilateral security.

2 The Need for Secure Hardware

In the conventional sense, secure platforms are systems that enforce user-defined security policies to protect them against attacks from inside and outside of the platform, e.g., against other users, remote adversaries and malicious applications such as a virus or worm.

Traditional security targets of secure platforms are authenticity, integrity, privacy, anonymity, and availability. In general, the measures applied to achieve them are information flow control and access control mechanisms [12]. For a secure realization of these mechanisms, the underlying platform has to fulfill appropriate security requirements.

It is sometimes stated that these security requirements can be fulfilled based on common hardware architectures. However, this is not true, since in the era of smartphones, notebooks and PDAs the untrusted environment does not physically protect the device anymore. However, untrusted adversarial environments require tamper evidence or tamper resistance, which is not provided by common hardware architectures. Even the certain degree of tamper-resistance provided by smartcards (e.g., to protect unauthorized access to cryptographic keys) do not help here, since they cannot offer other important security features such as *trusted path*[4] (see below).

2.1 Types of Threads

We informally distinguish between two types of threats, called Adv_1 and Adv_2.

- In Adv_1, adversaries cannot gain physical access to the device. Thus, attacks can only be performed remotely, e.g., via remote shell, Trojan horse or virus. Further, it is assumed that users do not break their own security policy[5]. An example of Adv_1 is a desktop computer that is connected to the Internet but physically protected by the environment, e.g., a locked room.
- In Adv_2, adversaries have physical access to the device, which requires tamper evidence or tamper resistance to protect its integrity. As in Adv_1, it is assumed that users do not break their own security policy.

[4] This is the reason why smartcard applications based on insecure operating systems can easily be broken.

[5] Nevertheless, inexperienced users may break their security policy by mistake.

2.2 Problems of Common Hardware

Since the functional security requirements to be fulfilled by a secure platform are well known (see, e.g., [17,12,28]), the following discussion focuses on their technical feasibility based on common hardware architectures:

Trustworthiness (R1): The most important requirement is the trustworthiness of all components that can break the security policy. These components are generally called the trusted computing base (TCB). The size and complexity of security-critical implementations directly influences their reliability and thus the trustworthiness of TCBs. As stated in [31] a typical Unix or Windows system (including major applications) consists of about 100 Million lines of code (MLOC). An analysis of a common Linux distribution using *sloc* [39] counts about 55 MLOC and that the TCB counts more than 5 MLOC[6]. According to [31], several studies have shown that typical software consists of about one security critical bug per 1000 lines of code. Thus, we have to assume that more than 5000 security critical bugs are part of the TCB of a typical Linux distribution.

Further methods to increase the trustworthiness of critical components is the evaluation according to the common criteria [12] and to make source code and design decisions publicly available [3,29].

Confidentiality and integrity of application code and data (R2): This requirement should hold during the execution and during the storage.

The former is required to prevent concurrent processes from accessing confidential information. Unfortunately, with common hardware architectures it is very difficult to prevent untrusted components from accessing data. For example, an untrusted process must never be allowed to control DMA-enabled devices[7]. As long as common hardware architectures contain these difficulties, the only solution is to put critical code (e.g., every device driver) into the TCB.

The latter requirement (protection during storage) can be realized in Adv_1 based on existing hardware architectures using cryptographic primitives. However, for this to be realized under the adversary model Adv_2 more effort is required. In theory, it would suffice to have a *complete* tamper-resistant platform including the whole TCB, but the reality shows that this is a strong assumption [5,2]. Thus, common solutions are to encrypt critical data and to protect the master encryption key either by entering it at system startup, or by protecting it by tamper-resistant components.

[6] The Linux kernel 2.4 contains about 2.4 MLOC, but any application that is executed with root rights and libraries used by this applications can directly manipulate the kernel and are therefore part of the TCB. For example, alone the XServer extends the size of the TCB by about 1.3 MLOC.

[7] DMA (Direct Memory Access) allows peripheral hardware, e.g., the video adapter or the sound card, to directly access physical memory circumventing memory protection mechanisms. Thus, malicious modules which control DMA-enabled hardware (e.g., a device driver), can bypass policy enforcing mechanisms.

Integrity of the TCB (R3): The enforcement of security policies can only be guaranteed as long as the TCB is not manipulated. Since existing hardware architectures does not provide mechanisms to protect the integrity of critical components, adversaries can easily manipulate software components of the TCB, e.g., by exploiting design and implementation failures like /dev/kmem or buffer overflows. An important aspect in this context is a secure bootstrap architecture to prevent adversaries from manipulating the TCB when it is inactive. If the adversary has no physical access (Adv_1), a secure bootstrap architecture can for instance be provided by using a read-only boot medium, e.g., a CD-ROM. Obviously, this is impossible under Adv_2. Another important issue in this context is to provide users with a mechanism to securely verify the results of the integrity check [1].

Enforcing least privilege (R4): Many security flaws concerning viruses and Trojan horses result from the fact that common operating systems do not enforce the concept of least privileges. The underlying access control mechanisms, originally designed to separate different users from each other, are based on access control lists (ACL). As a result, every application has all privileges of the invoking user, which makes it very easy for malicious software to bypass confidentiality or integrity policies.

Trusted path to user (R5): A trusted path prevents unauthorized applications from accessing passwords entered by the ordinary user. Further, it is a prerequisite for application authentication to prevent faked dialogs [37,9]. Application authentication also requires a secure application manager that provides the necessary information about the applications, e.g., a unique, user-defined application name.

Secure channel to devices and between applications (R6): Integrity, confidentiality and authenticity of inter-application communication is required to prevent malicious applications from deceiving honest applications, e.g., by faked process numbers. Further, it has to be ensured that unauthorized components can neither read nor write to buffers of peripheral devices.

Secure Random Numbers (R7): To be able to use cryptographic primitives, a secure generation of random numbers is required to generate secure cryptographic keys. With existing deterministic computer architectures the generation of secure random numbers is very difficult, if not impossible [15,19].

As a bottom line of this section, we can stress that the development of a secure system based on existing hardware architectures under Adv_1 is difficult, but possible. Research results in operating system security have shown that this is also possible in practice [28,34,21]. Under threat type Adv_2 a secure system can by no means be developed without hardware extensions, since tamper-resistance or at least tamper-evidence is required. As we have already stated above, Adv_2 is important whenever the system cannot be protected by the environment, e.g., PDAs, mobile phones, notebooks, embedded systems or pervasive computing. Thus, we require hardware architectures that under Adv_2 offer features which can be deployed to fulfill the above mentioned requirements.

Next, we consider the TCPA and Palladium and briefly discuss what features they provide to overcome some of the shortcomings of the conventional hardware architectures.

3 TCPA and Palladium

Since NGSCB and TCPA are announced to improve user's security, this sections discusses on a more technical level to what extend TCPA/Pd can keep their promises. We review only the architecture of TCPA and briefly outline known differences to Pd. The reasons are that TCPA and Pd provide similar functionality (on technical level) and that only the full TCPA specification is publicly available.

Figure 1 outlines the components of a TCPA-compatible PC platform [35]. Beside the conventional hardware, TCPA consists of two tamper-resistant modules called TPM (Trusted Platform Module)[8] and CRTM (Core Root Trust Module). The operating system supporting these modules is not part of the TCPA specification.

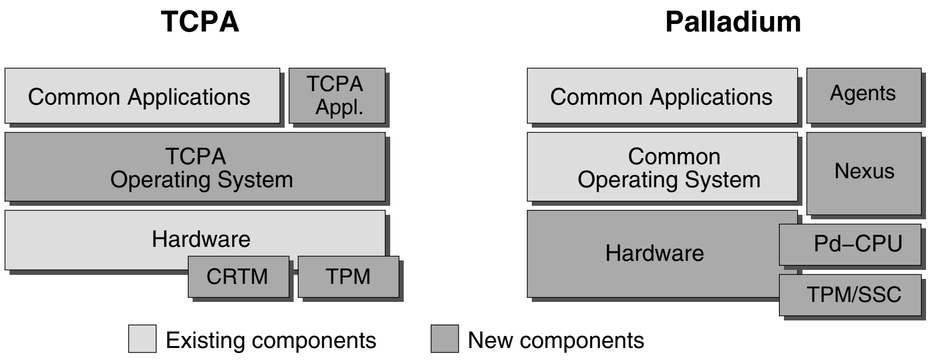

Fig. 1. Comparison between the TCPA and the Palladium architectures. Palladium allows a common operating system to be executed in parallel but requires more changes to hardware components to prevent the common operating system to bypass security policies.

The TCPA specification [36,35] extends the PC architecture by the following functionalities:

– **System configuration authentication (attestation):** The TPM contains a certified and unique signature key that can sign the current system

[8] Some documents on Palladium call it the Security Support Component (SSC).

configuration H stored in protected TPM registers $h_1 \dots h_n$. H is determined by the bootstrap process using a SHA-1 hash function. When the CRTM is invoked on system startup, it determines the BIOS configuration, writes the result into h_1 and invokes the system BIOS which itself determines the configuration of the boot sector, writes it to h_2, invokes the boot loader and so on [40]. This procedure is continued until the operating system is loaded.[9]

- **Sealing:** The integrity preserving mechanism of TCPA does not protect the TCB directly, but ensures that data that was encrypted under a configuration H cannot be revealed under another configuration H'. This is realized by cryptographically binding relevant information (e.g., the system configuration and the identifier of the invoking application) to the data to be encrypted. Note that the TPM cannot distinguish between different applications. Thus, the application identifier (e.g., a hash value of the application code) has to be provided by the operating system.
- **Protection of cryptographic keys:** The identifying signature key and the keys used to seal data are protected by the TPM which they never leave. Instead, the TPM itself provides cryptographic functions to operate on the cryptographic keys.
- **Secure random numbers:** Beside cryptographic functions used to determine and sign the system configuration and to seal data, the TPM contains a secure random generator to generate secure cryptographic keys.

As we will discuss in Section 4.3 these features can be deployed to satisfy some of the main requirements on secure hardware we listed in Section 2.2.

From the existing (non-technical) descriptions on Palladium [11,14,33] one may derive an architecture as outlined in Figure 1. In contrast to TCPA, Palladium allows an existing operating system to be executed in parallel to a new kernel called *nexus*[10], which itself executes critical applications called *agents*. The architecture allows the conventional operating system to be executed without the nexus, and that the nexus can be loaded later. Therefore, determining the system configuration in Palladium is simpler, since agents only depend on the nexus kernel and the underlying hardware. Compared to TCPA, Pd requires more hardware changes than TCPA: first, a new CPU is required which allows the nexus to be executed. Second, to prevent interferences between the conventional operating system and the nexus, nearly every hardware device has to support Palladium by providing a "nexus-mode".

A major benefit of Palladium is an extension of the mainboard chipset that allows the nexus to control DMA cycles [11,33] (see Section 2). Since TCPA does

[9] The functionality of the bootstrap process is similar to those described in [8].

[10] This is done by adding a new mode of operation that allows the nexus to protect itself (and applications for which the nexus acts as the operating system) against the conventional operating system. A possible implementation of the extension is to add another CPU protection ring, e.g. r_{-1} *below* protection ring r_0 [6] and give it capabilities to hide memory pages and process structures to protect itself and critical applications from code executed on r_0 or above (the conventional operating system).

not contain such a functionality, appropriate drivers and hardware devices have to be part of the TCB which enormously increases its size and complexity.

As a result, we can outline that the functionality provided by TCPA and Pd is completely under the control of the operating system. This moves the question, whether end-users can trust TCPA/Pd platforms or not, to the question, whether they can trust their operating systems. Therefore, the next section proposes an architecture of a security platform that uses the features of TCPA/Pd to increase the overall operating system security without allowing to realize those capabilities of TCPA/Pd people are concerned about.

4 Towards Personal-Secure Systems

As we have discussed in Section 2, many requirements to build secure computing platforms are difficult to fulfill using existing hardware architectures, especially, if the adversary has physical access to the system's hardware (Adv_2). Moreover, we discussed that TCPA/Pd offer prospective properties to build secure systems as long as the underlying operating system can be trusted by the user. In this section we propose an architecture for a secure computing platform in the sense of Adv_2 using TCPA/Pd in conjunction with a security kernel called PERSEUS we have developed. First, we shortly explain the architecture of this security kernel.

4.1 The PERSEUS Security Architecture

PERSEUS[11] [28] is a minimal open-source security kernel providing all operating system services required to protect security critical applications executed on top of it. The main idea is to let common operating systems run as applications on the top of the security kernel, since the past has shown that secure operating system architectures live in the shadow if they are incompatible to a common operating system. Therefore, a tamed[12] operating system (Client OS) provides users with a common user interface and a *backward-compatible interface* to reuse all uncritical standard applications and services.

One main design goal of the PERSEUS security architecture was the realization of a minimal and therefore manageable, stable and evaluable security kernel for conventional hardware platforms such as IBM-PC and mobile devices like PDA's and smartphones. This requirement was fulfilled by extracting only security-critical operations and data to the security kernel. The Client OS still provides all uncritical operating system services like a filesystem, network support, etc.[13] Our decision to use this hybrid architecture is motivated, among others, by the following two facts: Firstly, the development of a completely new

[11] www.perseus-os.org

[12] Currently, a slightly modified user-space Linux adapted to the L4 microkernel, L4-Linux [20], is used.

[13] The hardware access used to provide this high-level services is controlled by low-level services of PERSEUS.

secure operating system that provides backward-compatibility is too costly. Secondly, improving the existing operating systems, e.g., like SE-Linux [26], are too inflexible[14]. The high-level PERSEUS architecture is illustrated in Figure 2.

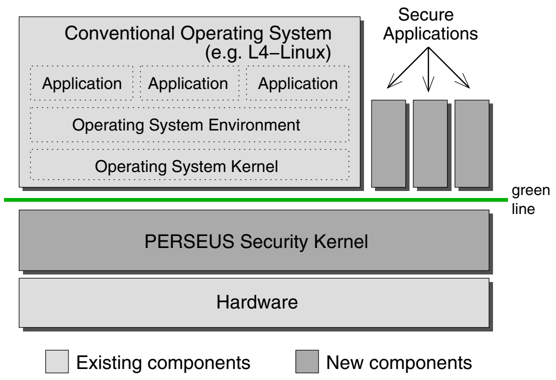

Fig. 2. An overview over the PERSEUS architecture. The green line indicates the border between trusted and untrusted components.

The PERSEUS security kernel is based on the efficient L4 microkernel [25,18] that provides (i) a hardware abstraction, (ii) elementary mechanisms, and (iii) flexible policies that are required to control and protect critical system services, device drivers and resource managers as separated user-space processes (a so-called multi-server system). This prevents that errors of one component can affect others, ensures that only authorized processes can access the hardware and guarantees that only the microkernel itself is executed in the supervisor mode of the CPU.

Also security-critical applications and the conventional operating system are implemented as separated user-space processes which can only communicate to each other or to the underlying hardware using services provided by the secure kernel. This allows PERSEUS to enforce its own user-defined security policy independent of the conventional operating system.

The PERSEUS security kernel currently consists of the following components (see Figure 3):

– **Microkernel.** The only component that is executed in supervisor mode. It contains about 7100 lines of code (LOC) and provides basic process, inter-process communication (IPC), thread, memory and exception support. Furthermore, capabilities to processes, interrupts and I/O ports and a process-level access control mechanism [24,22] is provided.

[14] On the one hand, it is questionable whether, e.g., a trusted path can be provided without many changes to core components. On the other hand, these improvements do not decrease the complexity.

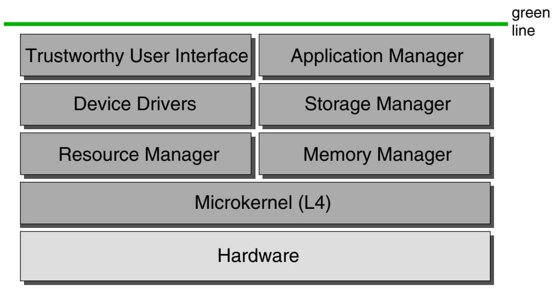

Fig. 3. A more detailed illustration of the PERSEUS security kernel. Again, the green line indicates the border between application layer and trusted computing base.

– **Resource Manager.** The resource manager enforces security policies on low-level hardware resources like interrupts, processes and I/O ports. It consists of about 5600 LOC.
– **Memory Manager.** This module ensures that different services and applications do not share memory pages. The size of the implementation is currently about 654 LOC.
– **Device Drivers.** Because of the security problems with DMA-enabled devices discussed in Section 2, every device driver that accesses DMA-enabled devices (or at least the critical parts) has to be isolated in the secure environment to be protected against malicious modifications. The size depends on the size of the original drivers.[15]
– **Trustworthy User Interface.** The trustworthy user interface provides a *trusted path* by controlling the hardware of input devices (keyboard and mouse) and the video adapter. The implementation ensures that only the active application receives user input and exclusively controls a small region of the output device used for application authentication. For larger screen resolutions (e.g., PC's), a minimal implementation of a secure window manager is provided. The current implementation has a size of about 5400 LOC.
– **Application Manager.** The application manager controls installation and removal of security critical applications based on code signing. It enforces user-defined installation policies and assigns predefined sets of permissions to new applications. The implementation is currently in beta-state; we expect an overall size of about 5000 LOC.
– **Storage Manager.** This module encrypts/decrypts memory pages of other applications/services and persistently stores them using the filesystem of the Client OS. Its size is currently about 500 LOC.
– **Secure Booting.** Because of lack of hardware support, secure booting is currently only provided marginally. The storage manager uses cryptographic

[15] This is especially a problem of common PCs, because they can have a huge amount of DMA-enabled devices which enormously increases the size and complexity of the TCB. For mobile devices like PDAs, this is not so important, since they have a relatively fixed hardware architecture.

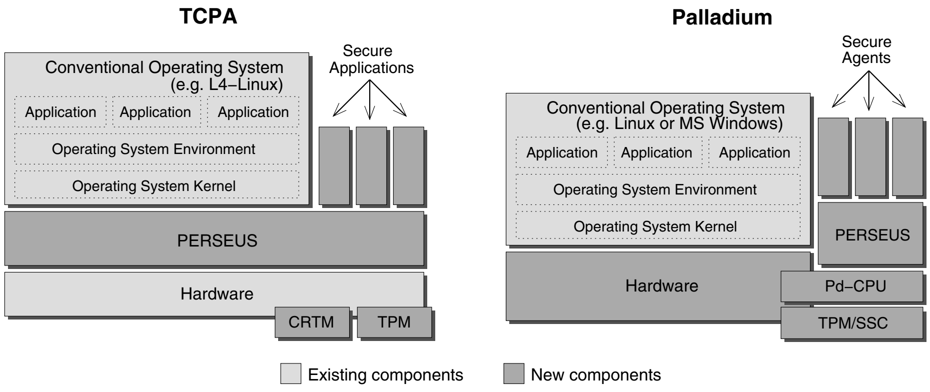

Fig. 4. When the PERSEUS architecture is combined with TCPA or Pd hardware, the resulting architectures are somewhat similar. The Pd approach allows the reuse of every conventional operating system, but requires more changes to the hardware.

key derived from a passphrase entered by the user at system startup. Additionally, all critical components can be stored onto a read-only medium, e.g., a CD-ROM.

– **DGP.** DGP is a PERSEUS application that provides users compatibility to PGP [41]. Security-critical data, e.g., cryptographic keys, never leave the PERSEUS application, since also security-critical operations are performed by the PERSEUS service. Additionally, DGP provides a trusted viewer/editor to verify, e.g., information to be signed, and to enter confidential data.

For more information on PERSEUS and to download the current source code, see the PERSEUS homepage[16].

4.2 Improving Security with TCPA/Pd

To obtain a secure platform in the sense of Adv_1 and Adv_2, we can extend the PERSEUS security kernel to support TCPA and Pd. The latter, however, requires that Microsoft publishes technical specification about Pd, as promised [14]. Since PERSEUS separates critical from uncritical components, the resulting architectures look very similar (see Figure 4).

The only difference is that Pd allows the trusted kernel to control the conventional operating system by extending the CPU hardware, while the TCPA approach reuses exiting protection mechanisms but requires the conventional operating system to be modified.

[16] www.perseus-os.org

Table 1. Fulfilling security requirements by the functions of the proposed architecture

	R1	R2	R3	R4	R5	R6	R7
Random Generator		x	x				x
Sealing		x	x				
Attestation			x				
CPU		x	x				
Microkernel	x			x		x	
PERSEUS	x				x		

4.3 How Existing Security Problems Are Solved

The combination of the security functions provided by a minimal, highly secure operating system kernel and those provided by TCPA/Pd hardware allows us to offer a security platform that fulfills all security requirements that have been demanded in Section 2.2. Table 1 outlines which properties resp. functions of the proposed architecture are deployed to fulfill the corresponding requirement.

The trustworthiness of the proposed architecture is improved, since the reuse of an existing operating system allows us to keep the size and the complexity of the TCB very small (currently, about 25.000 LOC). Moreover, the attestation functionality of the underlying TCPA/Pd hardware allows also external systems to trust our architecture. Application code and data is protected during runtime by existing CPU memory protection mechanisms (e.g., virtual address spaces). Persistently stored data is protected by the extended Storage Manager that uses sealing to ensure that users cannot bypass security mechanisms by booting another operating system. The same mechanism is used by the bootstrap architecture to protect the integrity of the TCB. To allow reusing of existing device driver implementations, DMA has to be tamed as, e.g., proposed in [23,16]. Obviously, the proposed architecture cannot directly solve security problems of viruses, since viruses exploit security flaws on the application layer (e.g., a Word macro-viruses). Nevertheless, the proposed architecture reduces the potential damage caused by such attacks by enforcing its own security policy and separating code and data of different applications from each other. For example, the Client OS cannot access data of another PERSEUS application. A trusted path is provided by the TCB that controls a subset of the user interface and prevents Trojan horses attacks. Since secure IPC is the only communication mechanisms of the underlying microkernel a secure channel to devices and between applications can be provided. Finally, the random generator of the TCPA/Pd hardware is used to securely create cryptographic keys.

To be able to use the security features provided by our proposed architecture, security-critical applications have to be adapted to the interface of PERSEUS. To achieve this efficiently, we only need to move the security-critical data and the operations performed on this data to a compartment protected by PERSEUS. For example, consider a signature application: the critical data to be moved to the protected compartment is the signing key. The security critical operations are the generation of the signature and, to prevent attacks performed by an insecure Client OS, the verification of the document to be signed (trusted viewer).

5 Multilateral Secure DRM Platforms

Digital technology and media offer content providers and producers many business opportunities and users many advantages towards the realization of new electronic marketplaces. In this context, trading digital goods over open networks, such as the Internet, plays an important role. However, all these technological possibilities for comfortable handling and trading digital goods face us also with challenging problems regarding copyright protection of digital properties. *Digital Rights Management (DRM) platforms* are often considered as systems that should realize the appropriate environment for trading digital works while protecting the rights of authors and copyright holders. This is what we call a *DRM policy*.

To enforce DRM policies, one actually requires an ideal *trusted media player*, also called *trusted viewer* enabling only the authorized users to "view" (watch, listen to, edit, execute, etc.) digital works[17] while controlling the use of works according to the specified DRM policy.

5.1 Consequences of DRM

DRM policies and (user) security policies often conflict, e.g., if the external copyright control or system-wide censorship contradicts with a locally defined availability requirement of a user, or if a software product can prevent the installation of competitive products. While users are interested in unrestricted use, copying and even redistributing digital works, the providers want to protect their digital goods against misuse to limit financial damage. However, providers are not always the victims: they may also become the delinquents, and misuse DRM policies against users (see also [4] and [32]).

In DRM systems, providers are capable of enforcing any kind of policy within their specific media player. This is independent of any technical design and has many implications: For instance, a media player can censor the documents that it controls and a DRM-wordprocessor can encrypt its documents in such a way that other products like open-office cannot read them. If users desire to use DRM-enabled platforms and if they accept the underlying DRM conditions, then the only possibility to prevent issues such as censorship is the regulation by law.

5.2 Towards Trustworthy DRM Systems

Although the potentially negative consequences of DRM platforms mentioned above cannot be prevented on a technical level, a careful design of the TCB

[17] Note that the media player can control access to information only within the platform. Users may still be able to make unauthorized copies by using cameras or audio recorders.

(which now has to be trusted by users and providers) can prevent most of the dangerous consequences regarding user security. Most functions for controlling the system from outside, for instance a system-wide censorship, require a reference monitor located at the operating system layer or below. To enforce DRM policies such a system-wide reference monitor under control of content providers is not required, since the context to enforce DRM policies is only available at the application layer (the media players).

Therefore, we propose a multilateral secure operating system based on the PERSEUS security kernel that allows content providers to enforce DRM policies by guaranteeing that users cannot bypass enforcement mechanisms provided by the application layer, e.g., by patching media players. This can be provided by the sealing mechanism offered by TCPA and Pd, but requires a TCB that cannot be modified afterwards.[18] The enforcement of system-wide security policies, e.g., access control between applications and their data or the decision on which application are to be installed, can be kept under control of local users. This is by no means a restriction on providers, because the media player applications have to be trusted by providers at this layer anyway. The PERSEUS kernel is extended by services that provide sealing and attestation functions to be used by the application layer. This prevents users from bypassing DRM policies, e.g., by rebooting another operating system after the system was authenticated, since applications can still *bind* their contents to a specific system configuration. To allow applications to enforce their own access control policy, the Application Manager offers an application identification mechanism based on hash values.

This is what we call a *trustworthy DRM platform.* The important aspect about trustworthy DRM platforms is that they allow users to freely decide whether to accept or to reject applications that use DRM policies. Especially, it allows users to remove such an application without consequences for the other applications or system components.

The resulting architecture of a multilateral secure DRM platform is similar to the one outlined on Figure 2 and Figure 3. The only difference is that the TCB (all components below the green line) has to ensure that users cannot (i) manipulate application code and (ii) access application data. For instance, the user interface service must not allow other applications to create screenshots, and the storage manager resp. the memory manager have to strictly separate data of different applications. All these requirements are still fulfilled by the existing implementation of the PERSEUS security kernel.

As a bottom line of this section, we stress that security and DRM requirements are not mutually exclusive. Therefore, by strictly separating the enforcement of DRM policies and (user-defined) security policies, it is possible to provide a platform that allows external providers to enforce their policies without allowing them to misuse these mechanisms against users (privacy issues).

[18] Device drivers or Linux' kernel modules are negative examples. Microkernel systems that provide system services by user-space processes are more promising in this regard.

Many potential dangers of DRM systems debated in the press (see, e.g., [3, 32]) can be efficiently avoided using the proposed architecture of a trustworthy operating system. The proposed architecture provides only a minimal set of operating system functions, allowing different conventional operating systems to be developed (or ported) on top of it.

In this context, an important political and social issue to be considered is the use of an open-source security kernel that is not under control of one (or a few) vendors, avoiding monopolies. This may solve certain conflicts, since DRM platforms have to be trusted by both users and content providers.

6 Conclusion

In this paper, we discuss the capabilities of Microsoft Next-generation Secure Computing Base for Windows and TCPA based on the available documentation. Based on common security requirements we discuss why common hardware architectures in untrusted environments are unable to provide users with adequate security properties if the adversary has physical access to the device. Examples of untrusted environments are the use of mobile devices like smartphones, PDAs, notebooks and applications in pervasive computing.

Our analysis of TCPA/Palladium shows that their hardware architectures can be used to improve end-user security, but we also conclude that the DRM capabilities of TCPA/Palladium can be misused against users, if the underlying operating system does not prevent it. Therefore, we propose an open security architecture that relies on TCPA/Palladium hardware and an open-source security platform called PERSEUS we have developed. TCPA/Pd support can be implemented without much effort, providing the open-source community an alternative to commercial TCPA/Pd products. With the proposed architecture we also demonstrate that highly secure systems can profit from the features of TCPA or Palladium. However, the border between security and censorship is small and the community should observe further developments in this area carefully.

Since there is a need for DRM platforms, we emphasize that a careful design of a DRM kernel, what we call a trustworthy DRM platform, can prevent most of the negative consequences on the operating system level. The philosophy behind it is the strict separation between the enforcement of DRM policies and personal security policies. This allows user security policies and DRM policies to coexist and gives users the freedom to accept or reject a DRM application without further consequences. Furthermore, we have evaluated which negative consequences cannot be prevented on a DRM system on a technical level.

References

1. A. Alkassar and C. Stüble. Towards secure IFF — preventing mafia fraud attacks. In *Proceedings of IEEE Military Conference (MILCOM)*, 2002.
2. R. J. Anderson. *Security Engineering — A Guide to Building Dependable Distributed Systems.* John Wiley & Sons, 2001.

3. R. J. Anderson. Security in open versus closed systems — the dance of Boltzmann, Coase and Moore. Technical report, Cambridge University, England, 2002.
4. R. J. Anderson. The TCPA/Palladium FAQ. `http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html`, 2002.
5. R. J. Anderson and M. Kuhn. Tamper resistance – a cautionary note. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce* [38], pages 1–11.
6. J. L. Antonakos. *The Pentium Microprocessor*. Prentice Hall Inc., 1997.
7. W. A. Arbaugh. Improving the TCPA specification. *IEEE Computer*, pages 77–79, Aug. 2002.
8. W. A. Arbaugh, D. J. Farber, and J. M. Smith. A reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 65–71, Oakland, CA, May 1997. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.
9. N. Asokan, H. Debar, M. Steiner, and M. Waidner. Authenticating public terminals. *Computer Networks*, 31(8):861–870, May 1999.
10. A. Carroll, M. Juarez, J. Polk, and T. Leininger. Microsoft "Palladium": A business overview. Technical report, Microsoft Content Security Business Unit, August 2002.
11. A. Carroll, M. Juarez, J. Polk, and T. Leininger. Microsoft "Palladium": A business overview — combining microsoft windows features, personal computing hardware, and software applications for greater security, personal privacy and system integrity. White paper, Microsoft Windows Trusted Platform Technologies, July 2002.
12. Common Criteria Project Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation*, Aug. 1999. Version 2.1, adopted by ISO/IEC as ISO/IEC International Standard (IS) 15408 1–3. Available from `http://csrc.ncsl.nist.gov/cc/ccv20/ccv2list.htm`.
13. M. Corporation. Building a secure platform for trustworthy computing. White paper, Microsoft Corporation, Dec. 2002.
14. M. Corporation. Microsoft "Palladium" technical FAQ. http://www.microsoft.com, Aug. 2002.
15. D. E. Eastlake, S. D. Crocker, and J. I. Schiller. Randomness requirements for security. Internet Request for Comment RFC 1750, Internet Engineering Task Force, Dec. 1994.
16. L. Fraim. SCOMP: A solution to the multilevel security problem. In *IEEE Computer*, pages 26–34, July 1983.
17. M. Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Co., New York, USA, 1988.
18. A. Gefflaut, T. Jaeger, Y. Park, J. Liedke, K. J. Elphistone, V. Uhlig, J. E. Tidswell, L. Deller, and L. Reuter. The SawMill multiserver approach. ACM SIGOPS European Workshop, Sept. 2000.
19. P. Gutmann. Software generation of practically strong random numbers. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, USA, Jan. 1998. USENIX.
20. H. Härtig, M. Hohmuth, and J. Wolter. Taming linux. In *Proceedings of PART'98*. TU Dresden, 1998.
21. H. Härtig, O. Kowalski, and W. Kühnhauser. The BirliX security architecture. *Journal of Computer Security*, 2(1):5–21, 1993.
22. T. Jaeger, K. Elphinstone, J. Liedtke, V. Panteleenko, and Y. Park. Flexible access control using IPC redirection. In *Hot Topics in Operating Systems (HotOS VII)*, pages 191–196, Rio Rico, AZ, Mar. 1999.

23. B. Leslie and G. Heiser. Towards untrusted device drivers. Technical Report UNSW-CSE-TR-0303, School of Computer Science and Engineering, Mar. 2003.
24. J. Liedke. Clans and Chiefs. a new kernel level concept for operating systems. Working paper, GMD, 1991.
25. J. Liedke. Towards real micro-kernels. *Communications of the ACM*, 39(9), 1996.
26. P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. Technical report, U.S. National Security Agency (NSA), Feb. 2001.
27. C. Mundie, P. de Vries, P. Haynes, and M. Corwine. Microsoft whitepaper on trustworthy computing. Technical report, Microsoft Corporation, Oct. 2002.
28. B. Pfitzmann, J. Riordan, C. Stüble, M. Waidner, and A. Weber. The PERSEUS system architecture. Technical Report RZ 3335 (#93381), IBM Research Division, Zurich Laboratory, Apr. 2001.
29. E. S. Raymond. The cathedral and the bazaar. `http://www.openresources.com/documents/cathedral-bazaar/`, August 1998.
30. D. Safford. Clarifying misinformation on TCPA. White paper, IBM Research, Oct. 2002.
31. D. Safford. The need for TCPA. White paper, IBM Research, Oct. 2002.
32. B. Schneier. Palladium and the TCPA. `http://www.counterpane.com/crypto-gram-0208.html\#1`.
33. S. Schoen. Palladium details. `http://www.activewin.com/articles/2002/pd.shtml`, 2002.
34. J. S. Shapiro, J. M. Smith, and D. J. Farber. EROS: a fast capability system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 170–185. Kiawah Island Resort, near Charleston, Sout Carolina, Dec. 1999. Appeared as ACM Operating Systems Review 33.5.
35. Trusted Computing Platform Alliance (TCPA). TCPA PC specific implementation specification, Sept. 2001. Version 1.00.
36. Trusted Computing Platform Alliance (TCPA). Main specification, Feb. 2002. Version 1.1b.
37. J. D. Tygar and A. Whitten. WWW electronic commerce and Java Trojan horses. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce* [38], pages 243–250.
38. USENIX. *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, Oakland, California, Nov. 1996.
39. D. A. Wheeler. More than a gigabuck: Estimating GNU/Linux's size. `http://www.dwheeler.com/sloc/`, June 2001.
40. Wintermute. TCPA and Palladium technical analysis. `http://wintermute.homelinux.org/miscelanea/TCPASecurity.txt`, Dec. 2002.
41. P. Zimmerman. *The Official PGP User's Guide.* prz@acm.org, 1994. The MIT Press In press. More in `http://www.pegasus.esprit.ec.org/people/arne/pgp.html`.