Bridging the Gap between TCPA/Palladium and Personal Security

Ahmad-Reza Sadeghi
Ruhr-University Bochum, Germany
sadeghi@crypto.rub.de

Christian Stüble
Saarland University, Germany
stueble@cs.uni-sb.de

Abstract

Microsoft Palladium (Pd) and TCPA are announced to be the next-generation computing platforms, and claimed to improve users' security. However, the public debate on TCPA/Pd is full of skepticism and mistrust about the claimed security enhancements for the users. People are concerned about those features and capabilities of these systems that can be applied to realize Digital Rights Management: they may allow content providers to gain too much power and control over the use of digital content and users' private information. The general negative and sometimes prejudiced response about TCPA/Pd left less space for an unbiased and objective evaluation.

To clarify this situation, we first formulate the security requirements of a "trustworthy" platform on which users' security policies and providers' DRM policies are protected in the sense of multilateral security. Based on the common layered architecture for computing platforms, we analyze at which layers the security policies for users and the DRM policies for providers should be enforced. We then examine to what extent TCPA/Pd fulfill the requirements of a trustworthy platform where our analysis (and speculations) rely on the available TCPA/Pd documentations. Based on our experience in designing and implementing security kernels, we show how one can build and efficiently implement an open-source trustworthy platform using TCPA/Pd hardware.

1 Introduction

A lot has been reported about Microsoft's next-generation secure computing base for Windows (former Palladium or Pd) [9, 14] and the Trusted Computing Platform Alliance[1] (TCPA) [12, 11]. The stated goal of these systems is to improve users' security [13, 20, 25, 24]. However, since their announcement, there is an ongoing public debate about the negative economical, social and technical consequences of these platforms. There are many concerns regarding their capabilities, in particular, in conjunction with Digital Rights Management (DRM)[2]. People are worried about the possibility that TCPA/Pd may give vendors and content providers too much control over personal systems and users' private information: they may allow commercial censorship[3], political censorship, destroy innovation[4] or undermine the General Public License (GPL). Especially, the open-source community seems to resist against TCPA/Pd, mainly because they are more sensitive regarding user security and privacy issues (see [3] and [28] for more discussions on TCPA/Pd).

Thus, we are faced with many open questions and uncertainties about the real capabilities of TCPA/Pd: do these platforms provide users with enhanced security features (e.g., better protection against spam and viruses), or do they protect digital content providers (such as large motion pictures) allowing them to misuse DRM capabilities against users (e.g., violate their privacy) leading us to information dependency and slavery?

There are several reasons for this situation: firstly,

[1] www.trustedcomputing.org

[2] The notion of DRM is often used for systems which control the use of digital content with the goal to protect the (copy)rights of authors and holders of digital properties. For instance, a digital content provider may sell consumers music or video clips which run only on TCPA/Pd and the users are prevented from making (unauthorized) copies.

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

[4] 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.

with a very few exceptions[5], the public discussions were solely dedicated to non-technical issues and led to improper conclusions about the capabilities of TCPA/Pd. Secondly, there exist very few technical documentations on TCPA which can be analyzed so far for the feedback purposes, and nearly nothing technical is available on Palladium. Thirdly, new announcements appear occasionally, attempt to justify these platforms as security enhancements for users but by giving obscure argumentation why they are not appropriate for DRM applications. For instance, in [25] it is claimed that TCPA is to protect user's private data against remote attacks, and since it is not tamper-resistant, it is also not suited to DRM. However, if this is the goal of TCPA, then there is also no need for a new chip since the smartcards already do the job. They have even the advantage that they are only bound to an individual and not to a platform.[6]

To clarify some of these problems, we first define the requirements for a trustworthy platform. By trustworthy we mean a platform which protects both the privacy of users and the rights of digital content providers in the sense of multilateral security. In this context, users are free to accept or reject DRM policies for each individual application without affecting their own security requirements.

Based on a common layered computing platform architecture, we analyze at which layers security and DRM policies have to be enforced. Our analysis relies on available documentations on TCPA/Pd. We then evaluate to what extend TCPA/Pd satisfy or violate these requirements. We come to the conclusion that the hardware features of TCPA and Palladium in their basic form cannot harm the security of the user, in the sense of the terrifying scenarios mentioned above, if the underlying operating system does not support the related scenarios.

Based on our experience in designing and implementing security kernels, we propose design rules that avoid such problematic scenarios. Furthermore, we propose an architecture for a trustworthy platform and show how one can efficiently implement it using modern operating system technology and the TCPA/Pd hardware. For a concrete realization of such a system, we use an existing open-source security kernel [23]. An additional advantage of this approach is that it provides the open-source community with an alternative solution to commercial products.

[5] see, e.g.,[6] and [33]

[6] Another argument in [25] is that there are too many different possible system configurations to be managed by content providers to be able to enforce DRM policies. However, in our opinion, there is no reason why this should be unmanageable. One could define a list of trusted components and configurations and compute the required combinations on demand. Alternatively, providers may support only a small set of allowed configurations, e.g., a secure configuration of common operating systems.

2 Requirement Analysis

In general, IT-systems involve many different parties having different interests, and thus different security requirements. We distinguish between security requirements of the user or owner of IT-systems, and the requirements of providers of digital content[7]. For our analysis of computing platforms we use the common layered architecture as illustrated in Figure 1. In the following, we differentiate between a secure platform, a DRM platform and a trustworthy platform.

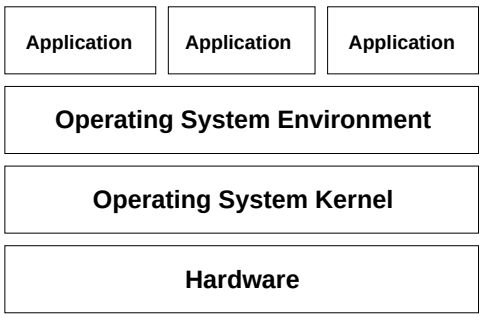

Figure 1: Different layers of a general system architecture.

The *hardware layer* provides the interface of hardware components like CPU, display, storage and other peripheral devices. The *operating system layer* controls hardware resources and implements strategies to share them. The *operating system environment layer* offers services to the application layer to ease the use of the operating system and is either provided as libraries (e.g. the libc in Unix) or as separated user-space processes (e.g., a daemon or the Java virtual machine). At the *application layer* users can execute their applications.

[7] Here, providers represent all parties who possess (copy)rights on digital properties including authors and rights holders.

2.1 Secure Platforms

In the conventional sense, secure platforms are those systems that enforce *security policies* defined by their users or owners 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 to be achieved by secure platforms are privacy, integrity and availability. The measures used to enforce security policies are information flow control and access control, where the latter can be partitioned into two classes, mandatory access control (MAC) and discretionary access control (DAC) [22]. To realize such policies the underlying platform has to provide appropriate mechanisms. We postponed describing them to Section 2.3.

Trust Model. It is assumed that users do not break their own security policy[8]. Further, users trust all hardware and software components that can break their security policy. The set of trusted components is called TCB (Trusted Computing Base). Users do not trust applications in general. Moreover, we distinguish between two types of threats, called Adv_1 and Adv_2. Adv_1 is the general case, where adversaries can access (manipulate or analyze) the hardware. This usually implies tamper-resistance assumptions on hardware. Beside the fact that tamper-resistance is a very strong assumption, today's computing platforms do not fulfill this assumption at all [4]. In Adv_2 it is assumed that the environment protects the hardware (e.g., by locks) in such a way that adversaries can only perform attacks remotely (e.g., by a worm or virus).

Enforcing Security Policies. Security policies can be enforced at different layers. Now we turn our attention to the question at which layer secure platforms should enforce security policies. We discuss advantages and disadvantages of possible approaches. Note that we do not discuss where and how bypassing of policies can be prevented. This is a completely orthogonal aspect which we consider in Section 2.3.

[8]Nevertheless, unexperienced users may break their security policy by mistake.

- **Hardware:** Enforcing security policies at this layer is not very common since certain information about the data, such as encoding or data type information, is not available at this layer. For instance, a harddisk cannot distinguish between a music track and an image.
- **Operating System:** Enforcing security policies at this layer is necessary, because any entity controlling the hardware is capable of controlling all other software components. Typical examples are the protected mode of common CPU's or DMA (Direct Memory Access) which allow the controlling entity to access every physical memory region. However, the enforced policies are restricted to the data types known to the operating system (mainly file, user, process, etc.).

  An important issue to note is that some security policies such as mandatory access control (MAC) have to be realized at this layer, since they have to be applied system-wide.
- **Operating System Environment:** For enforcing security policies the same holds here as for the application layer (see below).
- **Application:** Enforcing security policies at this layer makes sense, because in contrast to the other layers, it has information about the data, such as encoding or the data type. Often it does not make a difference whether the corresponding security policies are enforced by the operating system or the application. The reason is that for many practical uses the application has to be trusted anyway.[9]

Research results in operating system security have shown that secure platforms can be build using modern operating system technology [23, 29, 17]. Optionally, smartcards can be used if the protection of cryptographic keys is of special interest [10].

2.2 DRM Platforms

Digital technology and media offer content providers and producers many business opportunities and users many advantages towards the realization of

[9]As an example, consider a signature application. No matter if the related security policy is implemented at the application layer or at the operating system layer, the signature application can always misuse its secret key.

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*. Note that DRM policies attached to digital works may get very complex, because rights expressions may consist of permissions, constraints, obligations, rights holders, etc. For instance, consider Aisha and Bill who made a certain video (e.g., how to find oil in the dessert). George is only allowed (usage permission) to watch certain parts of the film for a certain number of times (constraint). However, each time, George has to pay usage fee (obligation to pay) to Aisha and Bill (rights holder).

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[10] while controlling the use of works according to the specified DRM policy.[11] The realization of such a trusted viewer requires the combination and interplay of various components to achieve at least some of the desired properties of an ideal "trusted media player".

Trust Model. In contrast to the trust model of secure platforms, DRM platforms assume that users are malicious. Both provider and user have to trust the TCB, because it can break both security policies and DRM policies. Users do not have to trust applications, but providers trust their own media players. Here, only threat model Adv_1 holds, because the user (which is untrusted) has physical access to the platform. This implies tamper-resistance assumptions. Past solutions were not satisfactory for general purpose platforms. For instance, a way for distributing software products is to use *dongles*, hardware devices plugged, e.g., into the printer port. The software does not work unless the dongle is installed. However, dongles turned out to be impractical for mass software market. Consumers did not accept them since for each application a separate dongle was needed (see also [2]).

Enforcing DRM Policies. Based on the layered system architecture presented in Section 2.1 (see Figure 1), we now discuss at which layer DRM policies can (or should) be enforced. Again, we do not consider the required mechanisms to prevent that policies can be bypassed. This will be mentioned in Section 2.3.

- **Hardware:** As for security policies, the enforcement of DRM policies on this layer is only meaningful for special purposes, e.g., mp3-player that completely prevent read access to the contained music tracks.
- **Operating System:** Enforcing DRM policies is also possible at this layer, but very critical, since it allows external entities to control the system – a system-wide censorship of files is one of the many possible horrible scenarios [3]. Furthermore, only a restricted number of data types are known at this layer (see Section 2.1) and complex DRM policies, as mentioned in the example above, cannot be realized efficiently. Nevertheless, the operating system has to ensure that applications cannot bypass enforcement mechanisms of the application layer.
- **Environment:** The same argumentation holds as for the application layer (see also Section 2.1).
- **Application:** Enforcing DRM policies at the application layer makes sense for the following main reasons: first, the required information about the used data types are only available at this layer. Second, the policy defining entity can only enforce the policy within the corresponding application and not system-wide. Third, the provider of digital works has to trust its own media player anyway.

2.3 Trustworthy Platforms

DRM policies and security policies often conflict, e.g., if the externally controlled system-wide cen-

[10] 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.

[11] In this context, one of the earliest approaches for efficient distribution of software is *super distribution*. The idea is to make software freely available without any restriction, but, the software would run on a system only if this system have installed the "super distribution technology" (see [19] and [15]).

sorship 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 [3] and [28]).

Therefore, as mentioned in the introduction, we need a platform that satisfies the requirements of both sides on a reasonable level in the sense of multilateral security. To achieve this, we propose to enforce DRM policies only at the application layer, where, as we discussed above, the providers cannot control more than their own media player and the corresponding digital goods. This is by no means a restriction on providers, because the media player applications have to be trusted by providers at this layer anyway. However, the provider must be ensured that the underlying platform layers guarantee that users cannot circumvent the DRM policies. This is what we call a *trustworthy platform*.

We will show in Section 4, after a brief review of TCPA and Palladium, how to efficiently provide such a platform by combining modern operating system technology and the hardware of TCPA or Palladium. However, first we have to discuss the functional requirements that have to be fulfilled by such a trustworthy platform on a more technical level.

- **Confidentiality and integrity of application code and data during execution:**

  The system has to protect application data in the memory during execution of the application, e.g. to prevent concurrent processes from accessing digital goods. The TCPA/Pd documents call this *curtained memory*.

  In the operating system community curtained memory is known as memory protection and can be realized by mechanisms provided by common CPU's. An example is virtual address spaces which can[12] ensure that a process cannot access a memory page of another process. Additionally, all other mechanisms that allow untrusted components to bypass memory protection must be prevented. For example, an untrusted process must not be allowed to access the video memory buffer of the graphic adapter[13] or to control DMA-enabled devices[14].

- **Confidentiality and integrity of application code and data during storage:**

  The system has to protect application data and code when they are persistently stored, e.g., whenever digital contents are written into a file. This is what TCPA/Pd call *sealing*.

  In theory, it would suffice to have a *complete* tamper-resistant platform, but the reality frequently shows that this is a strong assumption [4, 2]. Another approach is to use cryptographic primitives such as encryption and authentication codes. Since applications can easily be broken by re-engineering methods, the keys should not be stored in software. A possible approach for Adv_2 is the derivation of a master key (used for sealing all other keys) entered by the user at system startup.

  For Adv_1 the keys have to be stored into a tamper-resistant module (e.g., a smartcard). This is more reasonable than requiring a complete tamper-resistant platform.

- **Integrity of the operating system and underlying hardware:**

  Systems have to protect the integrity of the operating system and the underlying hardware to allow them to satisfy the requirements mentioned above.

  For instance, one can allow only a fixed and trusted system configuration to be used. The simplest way would be to put all critical parts into a read-only module (e.g., a secure boot-ROM). However, this module must be tamper-resistant for Adv_1. The main disadvantage of this solution is its inflexibility, because it forces users and providers to use a predefined system.

  A more flexible approach allows users to explicitly change the system to be installed, e.g., by modifying the system BIOS after entering a passphrase. Unfortunately, this approach cannot be used if the platform configuration has to

[12] In combination with an appropriate *memory manager*.

[13] A piece of untrusted software that can *read* these buffers has full access to digital content, and can therefore bypass DRM policies. If *write* access is possible, the platform cannot provide an application authentication mechanism any more.

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

be authenticated to external entities, because users may maliciously modify the installed system.

A possible solution is to 'bind' digital content to a specific system configuration. Binding in this context means that encrypted documents can only be decrypted by using the same system configuration as used for encryption. The advantage of this approach is that both user and provider can use any system they trust or use different systems on the same hardware platform. This is the way how TCPA and Palladium work (see Section 3.1 and 3.2).

- **Platform authentication:**

  The platform has to be able to authenticate itself to its user and external entities.

  Authenticating the platform (or, at least, to visualize changes) to the user is important in both Adv_1 and Adv_2 to prevent that adversaries modify or change critical components[15]. An important issue in this context is providing the user with a mechanism to securely verify the results of the integrity check [1].

  Authentication to external entities is required, for instance, to convince providers of the correctness of the system configuration.

- **Trusted path to user:**

  The platform has to ensure the confidentiality of user's input, e.g., preventing unauthorized access to passwords. Further, the platform has to provide a mechanism that allows users to authenticate applications, e.g., to prevent faked dialogs [30, 8].

  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:**

  The platform has to guarantee integrity, confidentiality and authenticity of inter-application communication to allow applications to enforce their own policy, e.g., to identify other applications. Further, the platform has to ensure that untrusted components can neither read from nor write to buffers of peripheral devices.

- **Reliability:**

  Another important issue is the size of trusted critical components. As stated in [25] a typical Unix or Windows system (including major applications) consists of about 100 Million lines of code (loc) and contains lots of security-related bugs ([25] mentions about one per 100 LOC).

[15] While Palladium and TCPA allow an authentication of external entities based on the provided keys (see also Section 3.1 and 3.2), there is no mechanism provided that allows users to check whether their system is correct or not. Sealing alone does not help here, because an adversary may install a malicious operating system that behaves like the original one (at least for some time, e.g., until the user has entered a passphrase).

As a bottom line of this section, we can stress that security and DRM requirements are not mutually exclusive. Quite the contrary is the case: nearly all required security measures overlap. Therefore, by strictly separating the enforcement of DRM policies and 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.

2.4 Consequences of DRM

If DRM is enabled, providers are capable of enforcing any kind of policy within their specific media player. This is independent of any design and is up to the users whether they are willing to use a DRM system or not. This implies, for instance, that a media player can censor the documents that it controls and that a TCPA/Palladium version of word 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 regulations by law.

The important aspect about a trustworthy platform is that it allows users to freely decide whether to accept or to reject applications that use such policies. If they do perform censorship, the provider has to mention it in the purchase agreement. Especially, it allows users to remove such an application without consequences for the other applications or system components.

3 TCPA and Palladium

In this section, we briefly review the architecture of TCPA and Palladium and possible ways of enforcing DRM policies. Moreover, we shortly compare their main characteristics.

3.1 TCPA

Figure 2 outlines the required components of a TCPA platform [12, 11]. Beside the conventional hardware, TCPA consists of two tamper-resistant modules called TPM (Trusted Platform Module) and CRTM (Core Root Trust Module) and an operating system that supports these hardware modules.

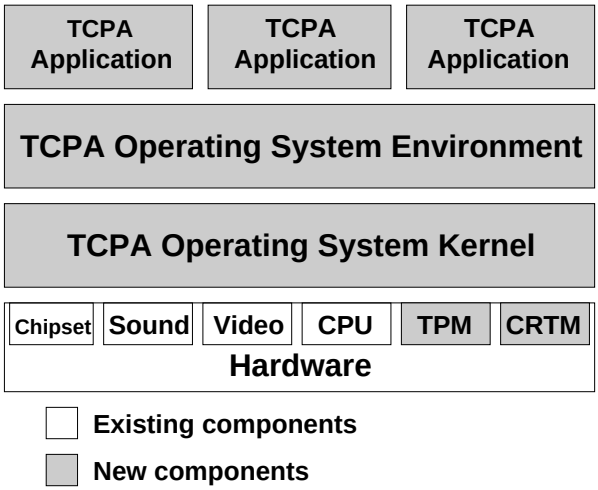

Figure 2: The TCPA architecture.

The main tasks of the CRTM is to initialize the whole system when it is turned on, and to authenticate the system BIOS (Basic Input Output System) and the hardware configuration (installed adapters, etc). The TPM performs mainly the platform authentication and sealing by using an internally stored certified signing key (of a secure signature scheme) and an encryption key.

To authenticate the platform to external entities[16] (e.g., providers of digital content), the TPM sends to the challenging entity signed information about the currently active system configuration. This allows the provider to accept or reject the user's system configuration.

[16]Note that the TCPA specification does not consider authentication to the local user. This security relevant feature has to be additionally provided by software mechanisms.

If a provider wants to enforce a DRM policy, it encrypts the corresponding digital content, a list of allowed system configurations and a list of the identifiers of the allowed media players using the encryption key of the TPM. To use digital content the corresponding application has to invoke the TPM to decrypt it. The TPM reveals the content to the application only if the current system configuration and the invoking application are contained in the lists. Sealing is performed in a similar way: if an application invokes the TPM to seal data, the system configuration, the application identifier and the data are encrypted and signed by the TPM.

A TCPA system measures the system configuration by observing the whole bootstrap process[17]. An authentication chain is created starting with the CRTM which will extend the BIOS of today's PCs [11]. The CRTM writes the authentication data of the hardware to a protected area of the TPM. Then the TPM hands over the control to the next instance of the boot process, e.g., the boot sector. This procedure is continued until the operating system is loaded. A more detailed technical overview is given in [33].

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.

3.2 Palladium

Unfortunately, no technical information about Palladium has been published. From the existing non-technical descriptions one may derive an architecture as outlined in Figure 3.

The main components of Palladium are a tamper-resistant Palladium-CPU, Palladium-devices a security chip called Security Support Component (SSC), a Palladium motherboard chipset and a Palladium kernel called *nexus*. In contrast to TCPA, Palladium allows a conventional operating system to be executed in parallel to the nexus[18].

[17]The functionality is very similar to those described in [7].

[18]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 [5] and give it capabilities to hide memory pages and process structures to protect itself and

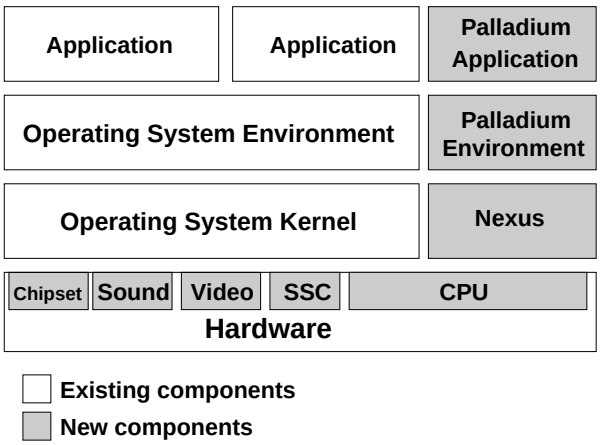

Figure 3: The Palladium architecture.

Similar to TCPA, Palladium can authenticate the platform and can bind digital works to system configurations. Since applications which depend on the nexus do not depend on the conventional operating system, it follows that the platform authentication does only depend on the nexus and the underlying hardware. This leads to a simpler mechanism for measuring the system configuration than in TCPA, since only the configuration of the nexus has to be measured by the underlying platform.

System authentication, sealing and the enforcement of DRM policies is similar to the mechanisms described in Section 3.1.

3.3 Comparison

In our opinion, the functionalities provided by Palladium and TCPA are similar, at least on a technical level. TCPA provides a basis for developing DRM applications on top of it, and leaves it to developers whether and how to provide backward compatibility to existing operating systems. The backward compatibility on top of TCPA may be reached by the following approaches:

- One can extend an existing operating system by TPM-supporting features and develop, e.g., a TCPA-Windows or TCPA-Linux [19]. The disadvantage of this approach is obvious: Both Windows and Linux have monolithic kernels containing critical and uncritical components without protection mechanisms between them. Therefore, one bogus or faulty part (e.g., an unimportant device driver) can compromise the whole system allowing adversaries to circumvent security and DRM policies. The existing and continously announced patches and exploits signify the problems of such kernels. We do not expect them to become more stable in the future since their internal structure has to be frequently adapted due to new hardware and user requirements.

- Alternatively, one can securely separate critical and uncritical parts by developing a small and stable kernel that executes a conventional operating system and DRM applications in parallel. The resulting architecture would be similar to those suggested by Microsoft, except that it can be provided with existing operating system and hardware technology [23, 16]. This is a very promising approach which we will explain in more details in Section 4 to overcome the concerns about TCPA and Palladium.

In contrast to TCPA, the Palladium solution allows a conventional operating system to run in parallel to the DRM kernel (nexus). This backward compatibility is achieved only through new hardware components. This concerns the CPU design for Palladium and new designs of hardware extensions like a Palladium-enabled graphic adapter or a Palladium-enabled sound card: they have to prevent the conventional operating system from *reading* the data which is written into the adapter buffer by DRM applications. Although the available material on Palladium does consider this requirement, it is not clear whether Palladium has forseen mechanisms that prevent the conventional operating system from *writing* into the buffer of such devices. As we remarked earlier, this is an important security requirement to authenticate applications for protecting the user against faked dialogs [30, 8].

Both TCPA and Palladium provide only a subset of the requirements discusses in Section 2. These are platform authentication and protection of data integrity and confidentiality. The mechanisms for realizing these requirements are passive mechanisms and rely on the operating system. The other requirement must be fulfilled by the operating system. Therefore, the terrifying scenarios described in the introduction can occur only if the operating system supports the corresponding policies. Nevertheless,

critical applications from code executed on r_0 or above (the conventional operating system).

[19] e.g., see www.research.ibm.com/gsal/tcpa/

care must be taken that also future designs of hardware make only use of passive mechanisms.

According to the available documentation, it is unclear to us how TCPA/Pd could better protect users against viruses and spam than existing tools, since both cannot be prevented using integrity preserving mechanisms. TCPA/Pd may only reduce the damage caused by viruses by protecting the confidentiality and the integrity of application code using the provided sealing mechanism. However, this feature can only be used by TCPA-enabled applications. Existing applications (especially the conventional operating system in Palladium) remain as insecure as before.

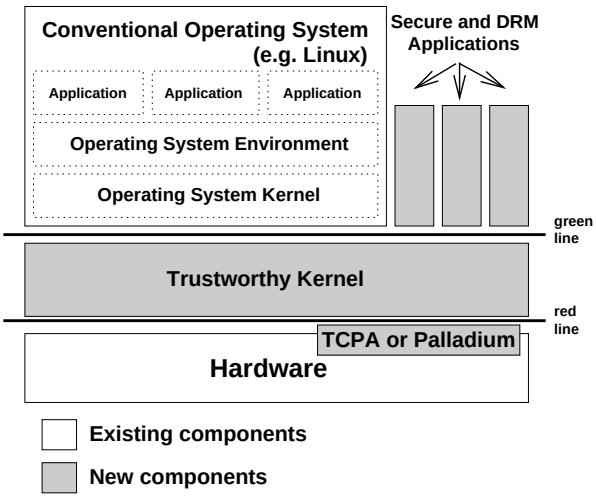

Figure 4: An overview of the proposed architecture for a trustworthy platform. The green line indicates the border between application layer and trustworthy platform.

4 Towards a Trustworthy Platform

In this section, we show how to provide a trustworthy platform that fulfills the requirements of users (security) and the providers (DRM) using modern operating system technology. The idea is, as already discussed in Section 2.3, to implement the DRM policies at the application layer based on any hardware platform which supports DRM such as TCPA and Palladium. To preserve the security of the user (against misuse by providers) the communication between the application layer and the hardware layer is controlled by a trustworthy layer (kernel). This kernel provides the appropriate environment where users can define their own policies. Further, the implementation of the trustworthy platform ensures that users cannot circumvent the DRM policies that they have already *accepted*. This requirement is fulfilled since core components of the the trustworthy platform cannot be manipulated by the user at runtime.

The architecture we propose is outlined in Figure 4. As one can see, DRM applications and secure applications (e.g., a digital signature) can run in parallel on top of the trustworthy platform (above the green line). To be backward compatible to existing platforms, it would be desirable that the trustworthy platform offers the appropriate binary interfaces of a conventional operating system.

To fulfill the requirements we discussed in Section 2.3, the trustworthy kernel provides the following features:

- It protects *confidentiality and integrity of application code and data during execution* by controlling the memory protection mechanisms of the CPU, and by providing a *memory manager* which strictly separates memory pages of applications. Further, the trustworthy kernel controls all other hardware components that are able to bypass the memory protection mechanism. For this, the security kernel ensures that only its components can (i) access memory buffers of hardware components (e.g., of the video adapter or the sound adapter) and (ii) control the hardware components which are able to perform DMA.

- It protects *confidentiality and integrity of application code and data during storage* by a sealing interface and a boot manager. The sealing interface allows applications to use the sealing functionality provided by the underlying hardware. The boot manager is a module which protects application code during persistent storage.

- It protects the *integrity of the operating system and the hardware* when the system is off. This is done implicitly by using the sealing mechanisms as described in Section 2.3. Additionally, the *integrity of the operating system* is protected during execution, since the trustworthy kernel is the only component that runs in protected mode.

- It allows *platform authentication* by using the appropriate features of the underlying hardware (see Section 3). To prevent the leakage of information about the platform configuration against user's will, a response to an authentication challenge is only generated if the user agrees.

- It provides a *trusted path to users* by controlling common user I/O components like keyboard, display and mouse. Further, the trustworthy kernel controls a protected region of the display for application authentication purposes, e.g., displaying the application's name.

- *Integrity and confidentiality* of the communication channel between applications is provided by the underlying IPC (Inter-Process Communication) mechanism. *Authentication* of applications is provided by an additional service, e.g., by a mapping between local process identifiers and authentication codes.

- An *Application Manager* controls the applications to be installed and enforces a user-defined security policy. Possible instantiations would be the use of code signing, source-code analysis, object analysis, or proof-carrying code [27, 26, 32, 21]. The application manager uses the cryptographic features of the TCPA/Pd hardware to be able to decrypt DRM applications and then generates unique identifier for applications (e.g., an authentication code for the TCPA/Pd hardware and a unique identifier for the authentication by users). Additionally, the application manager allows the user to set system-wide access control rules (e.g., allow two applications to communicate or not).

The trustworthy platform mentioned above can be implemented efficiently by a slightly modified security kernel. A concrete existing implementation of such a kernel is PERSEUS[20] [23] which we will shortly describe in the following section.

4.1 PERSEUS Security Kernel

PERSEUS is a security architecture developed based on the functional requirements of the Common Criteria [22]. Its main design goal was to realize a very small[21] (and thus manageable, stable and evaluable) security kernel for conventional hardware platforms such as IBM-PC and personal devices like PDA's. The architecture of PERSEUS is illustrated in Figure 5.

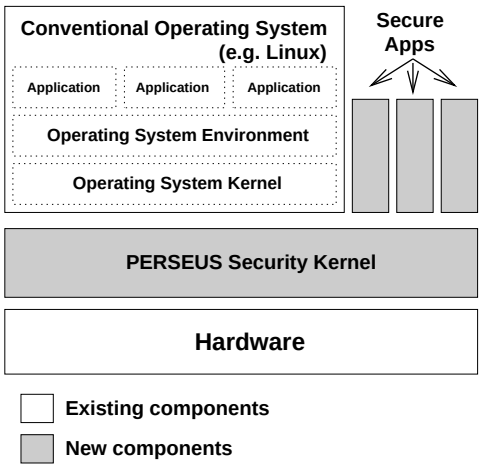

Figure 5: An overview over the PERSEUS architecture.

In contrast to other security approaches in the context of operating systems, PERSEUS acts as a small security kernel that controls all critical hardware resources to be able to protect security critical applications. It is located between hardware and conventional operating system. On the top of PERSEUS a conventional operating system (Linux) runs in parallel to security-critical applications. Controlled by the security kernel, it provides users a common environment to execute uncritical applications.

The PERSEUS security kernel is based on an efficient microkernel and provides all critical system services and device drivers as separated processes (a so-called multi-server system). This prevents that errors of one component can affect others. Security-critical applications and the conventional operating system are realized as separated processes which can only communicate to each other or to the underlying hardware through the security kernel. This allows PERSEUS to enforce its own user-defined security policy.

To obtain a trustworthy platform, the PERSEUS security kernel must be extended with appropriate

[20] www.perseus-os.org

[21] By counting all lines of code (loc) we get the following sizes: microkernel (the only component that is executed in protected mode) 1911 loc, resource-manager 2201 loc, secure services including trusted path, logserver, sealing, utility lib and gui lib about 1650 loc. Thus, the complete TCB (Trusted Computing Base) has a size far below 10.000 loc.

features, e.g., by adding TPM support to PERSEUS based on the source-code provided by IBM[22]. The trustworthy kernel could have the structure as shown in Figure 6.

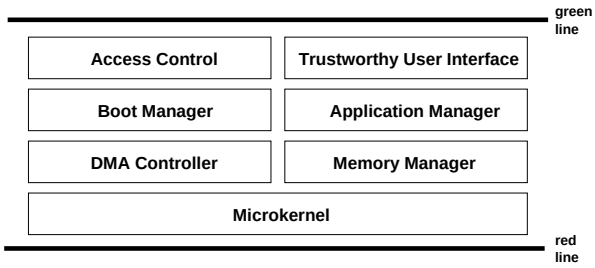

Figure 6: A more detailed illustration of the possible architecture of a trustworthy kernel if it is based on the PERSEUS security architecture. Again, the green line indicates the border between application layer and trustworthy platform.

4.2 Advantages of the Proposed Platform

In summary, the proposed trustworthy platform has the following advantages:

- It allows providers to protect their digital goods by DRM mechanisms and it protects security interests of users.
- It provides a high security level, because it benefits from TCPA/Pd features like DMA-protection, a true random generator and a secure bootstrap architecture.
- It provides backward-compatibility to an existing operating system without the need to change the underlying hardware, as required by Palladium.
- It allows the parallel execution of other proprietary operating systems, e.g., Windows, because it can be used as a nexus in a Palladium environment.
- Depending on the hardware platform, existing protection mechanisms (e.g. DMA control by the motherboard chipset) can be used or implemented in software.
- The size of critical components is very small compared to a conventional operating system like Windows and Linux. This allows an efficient evaluation of critical components, or even a formal verification of the source-code (the microkernel used by the PERSEUS security platform is currently formally verified [18]).
- The distribution under the GNU GPL open-source license allows every user to verify the implementation or compile it with its own trusted compiler.
- The architecture is open and not developed by any particular company preventing a monopoly. Further, unnoticed modifications, e.g., backdoors by any organization are prevented.

[22] www.research.ibm.com/gsal/tcpa/

5 Conclusion

In this paper, we evaluate the capabilities of Microsoft Palladium (recently renamed to "next-generation secure computing base for Windows") and TCPA based on the available documentation. Independent of the concrete hardware, we define general requirements for different types of platforms: (i) security platform where users' security policies are enforced, (ii) DRM platform where content provider policies are enforced and (iii) trustworthy platform which allows users to protect their privacy and allows content providers to protect their copyrights.

We come to the conclusion that TCPA/Palladium can be used as security and DRM platforms. The DRM capabilities of TCPA/Palladium can be misused against users, if the underlying operating system does not prevent it. The philosophy behind the trustworthy platform is the strict separation between the enforcement of DRM policies and 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.

We present an architecture for a trustworthy platform that contains a trustworthy kernel. The kernel can rely on the features of TCPA/Pd hardware to prevent users from violating the DRM policies they have already accepted. Further, it prevents providers from misusing DRM policies against users. The proposed architecture is based on a concrete existing implementation of an open-source security kernel called PERSEUS.

According to our experience in developing security kernels, the trustworthy kernel can be implemented without much effort, providing the open-source community an alternative to commercial 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 the security and the censorship is small and the community should observe further developments in this area carefully.

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. The TCPA/Palladium FAQ. http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html, 2002.

[4] R. J. Anderson and M. Kuhn. Tamper resistance – a cautionary note. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce* [31], pages 1–11.

[5] J. L. Antonakos. *The Pentium Microprocessor*. Prentice Hall Inc., 1997.

[6] B. Arbaugh. The TCPA; what's wrong; what's right and what to do about. http://www.cs.umd.edu/ waa/TCPA/TCPA-goodnbad.html, 2002.

[7] 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.

[8] N. Asokan, H. Debar, M. Steiner, and M. Waidner. Authenticating public terminals. *Computer Networks*, 31(8):861–870, May 1999.

[9] A. Carroll, M. Juarez, J. Polk, and T. Leininger. Microsoft "Palladium": A business overview. Technical report, Microsoft Content Security Business Unit, August 2002.

[10] P. C. Clark and L. J. Hoffman. BITS: A smartcard protected operating system. *Communications of the ACM*, 37(11):66–70, Nov. 1994.

[11] T. T. Committee. TCPA PC specific implementation specification v1.00. Technical report, TCPA Alliance, September 2001.

[12] T. T. Committee. Trusted computing platform alliance (TCPA) main specification v1.1b. Technical report, TCPA Alliance, February 2002.

[13] M. Corporation. Building a secure platfrom for trustworthy computing. Technical report, Microsoft Corporation, 2002.

[14] M. Corporation. Microsoft "Palladium" technical FAQ. http://www.microsoft.com, Aug. 2002.

[15] B. Cox. *Superdistribution, Objects as Property on the Electronic Frontier*. Addison-Wesley Publishing Company, 1996.

[16] H. Härtig, M. Hohmuth, and J. Wolter. Taming linux. In *Proceedings of PART'98*. TU Dresden, 1998.

[17] H. Härtig, O. Kowalski, and W. Kühnhauser. The BirliX security architecture. *Journal of Computer Security*, 2(1):5–21, 1993.

[18] M. Hohmuth, H. Tews, and S. G. Stephens. Applying source-code verification to a microkernel - the vfiasco project. In *Proceedings of the 10th ACM European SIGOPS Workshop*, 2002.

[19] R. Mori and M. Kawahara. Superdistribution: the concept and the architecture. Technical Report 7, Inst. of Inf. Sci. & Electron (Japan), Tsukuba Univ., Japan, July 1990.

[20] C. Mundie, P. de Vries, P. Haynes, and M. Corwine. Microsoft whitepaper on trustworthy computing. Technical report, Microsoft Corporation, Oct. 2002.

[21] G. C. Necula and P. Lee. Safe kernel extensions without run-time checking. In USENIX, editor, *2nd Symposium on Operating Systems Design and Implementation (OSDI '96), October 28–31, 1996. Seattle, WA*, pages 229–243, Berkeley, CA, USA, 1996. USENIX.

[22] C. C. P. S. Organisations. *Common Criteria for Information Technology Security Evaluation (Version 2.1)*. Common Criteria Project

Sponsoring Organisations, 1999. adopted by ISO/IEC as Draft International Standard IS 15408 1-3.

[23] 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.

[24] D. Safford. Clarifying misinformation on TCPA. White paper, IBM Research, Oct. 2002.

[25] D. Safford. The need for TCPA. White paper, IBM Research, Oct. 2002.

[26] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, Feb. 2000.

[27] F. B. Schneider, G. Morriset, and R. Harper. A language-based approach to security. In R. Wilhelm, editor, *Informatics – 10 Years Back, 10 Years Ahead*, volume 2000 of *Lecture Notes in Computer Science*, pages 86–101. Springer-Verlag, Berlin Germany, 2001.

[28] B. Schneier. Palladium and the TCPA. http://www.counterpane.com/crypto-gram-0208.html#1.

[29] 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.

[30] J. D. Tygar and A. Whitten. WWW electronic commerce and Java Trojan horses. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce* [31], pages 243–250.

[31] USENIX. *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, Oakland, California, Nov. 1996.

[32] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient software-based fault isolation. *ACM SIGOPS Operating Systems Review*, 27(5):203–216, December 1993.

[33] Wintermute. TCPA and Palladium technical analysis. http://wintermute.homelinux.org/miscelanea/TCPA Security.txt, Dec. 2002.