acTvSM: A Dynamic Virtualization Platform for Enforcement of Application Integrity

Ronald Toegl, Martin Pirker, Michael Gissing

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
{rtoegl, mpirker}@iaik.tugraz.at,
m.gissing@tugraz.at

Abstract. Modern PC platforms offer hardware-based virtualization and advanced Trusted Computing mechanisms. Hardware primitives allow the measuring and reporting of software configurations, the separation of application execution environments into isolated partitions and the dynamic switch into a trusted CPU mode.
In this paper we present a practical system architecture which leverages hardware mechanisms found in mass-market off-the-shelf PCs to improve the security of commodity guest operating systems by enforcing the integrity of application images. We enable the platform administrator to freely and deterministically specify the configurations trusted. Furthermore, we describe a set of tools and operational procedures to allow flexible and dynamic configuration management and to guarantee the secure transition between trusted platform configurations. We present our prototype implementation which integrates well with established Linux distributions.

1 Introduction

The ever evolving PC platform now integrates hardware that offers enhanced security to selected services. The concept of *Trusted Computing* extends the standard PC architecture with trust anchors such as the Trusted Platform Module (TPM) [49]. Also, commodity platforms [23] now provide hardware *Virtualization* with strong isolation of partitions (i.e. virtual machines) and novel mechanisms for safer computing. These hardware primitives can be used to exactly determine the software configuration of a system. A software platform may then collect and report expressive information on its configuration - an important precondition to leverage local integrity enforcement or remote attestation in distributed systems. Unfortunately, there is little available system software employing those technologies.

In recent years, a number of security-oriented virtualization platforms have been proposed or partially prototyped using specialized hypervisors (i.e. virtual machine monitors) to enable remote attestation [32,13], to protect code integrity [33,44] or to isolate critical code at runtime [20,46,16], some also employing a dynamic switch into specially protected CPU modes [45,31,30]. However, there is currently no approach that both supports unlimited types of guest operating systems and takes advantage of hardware security features like strong memory isolation and integrity enforcement.

In this paper[1], we present the acTvSM software platform which demonstrates that by a combination of up-to-date mass-market PC hardware and available open source software an enhanced security level (see Section 4.5) can be achieved. For protection at runtime our design integrates a *dynamic root of trust for measurement* (DRTM) with a virtualization platform offering hardware-isolation of partitions. With this platform, we restrict the execution of critical partitions to trustworthy, integrity enforced configurations. Also, our contribution includes mechanisms that allow the practical application of sealing and remote attestation by providing *deterministic* configuration measurements into the TPM. To this end, we modify the file system structure so that we achieve a separation of non-volatile storage into read-only, administrator-controlled areas, ephemeral in-memory space, and encrypted per-application logical volumes. Considering operational problems, we describe a set of administrative procedures and novel mechanisms which help us retain flexibility in the face of configuration changes and guarantee secure transition between trusted states. Our prototype implementation[2] demonstrates good integration with established Linux distributions and does not add runtime overhead.

Outline The remainder of this paper is organized as follows: Section 2 provides an introduction to Trusted Computing, Trusted Execution and Virtualization technologies. Section 3 presents our platform architecture. We discuss its properties and describe platform operations. Section 4 details the prototype implementation of our architecture and presents performance and a security analysis. In Section 5 we discuss related work. The paper concludes in Section 6.

2 Security Mechanisms of Commodity PC Hardware

2.1 TCG's Trusted Computing

Trusted Computing as it is available today is based on specifications of the *Trusted Computing Group* (TCG). The core hardware component is the *Trusted Platform Module* (TPM) [49]. Similarly to a smart card the TPM features tamper-resilient cryptographic primitives, but is physically bound to its host device.

An integrated circuit contains implementations of public-key cryptography, key generation, cryptographic hashing, and random-number generation. With these components the TPM is able to enforce security policies on hierarchies of secret keys to protect them from software attacks by any remote attacker. A unique Endorsement Key (EK) identifies each TPM.

[1] The design of the platform presented here was first sketched in [35], and the hypervisor integrity mechanisms were presented in [36].

[2] Available for download from [34].

The TPM helps to guarantee the integrity of measurements of software components by offering a set of *Platform Configuration Registers* (PCRs), which can only be written to via the one-way *extend* operation. PCRs are reset to defined values at platform boot. On version 1.2 TPMs, a PCR with index $i, i \geq 0$ in state t may then be extended with input x by setting

$$PCR_i^{t+1} = \text{SHA-1}(PCR_i^t || x)$$

with

$$PCR_i^{t_0} = \begin{cases} \texttt{0xFF}^{20} & 17 \leq i \leq 22 \text{ with static boot} \\ \texttt{0x00}^{20} & 17 \leq i \leq 22 \text{ after dynamic (re-)boot} \\ \texttt{0x00}^{20} & \text{else.} \end{cases}$$

PCRs can be used to exactly document the software executed on a machine by implementing the transitive trust model. Here, each software component is responsible to measure the following component before invoking it. Every caller computes hash values and extends a PCR with the result, before any of this executable code is allowed to run. In the simplest case, which we call *static*, this is done starting from the BIOS, covering bootloader, kernel, and system libraries etc., up to application code. Ultimately, a *chain of trust* is established where the full Trusted Computing Base (TCB), meaning the fundamental base system which needs to be trusted, and configuration of the platform is mapped to PCR values. If such a PCR configuration fulfills the given security or policy requirements, we refer to the system state as a *trusted state*.

The TPM can also *bind* data to the platform by encrypting it with a *non-migratable* key, which never leaves the TPM's protection. An extension to this is *sealing*, where a key may only be used with a specific PCR configuration. Thus, decryption of sealed data can be restricted to a trusted state of the computer. A current state may also be signed and quoted in the *Remote Attestation* protocol [48,13,40]. TPMs also provide a limited amount of non-volatile memory (NV-RAM) to store user- or owner-supplied information.

2.2 Virtualization

Virtualization is a methodology of dividing the resources of a computer into multiple execution environments, by applying concepts such as time-sharing [47], hardware and software partitioning, machine simulation or emulation. Hardware architectures can be designed to offer complete virtualization [37] in hardware and then host unmodified operating systems in parallel. Only recently, the PC platform has been modified accordingly (see [1] for an overview).

Commonly, virtualization is controlled by a singleton *hypervisor*, a superior control entity which directly runs on the hardware and manages it exclusively. It enables the creation, execution and hibernation of isolated *partitions*, each hosting a guest operating system and the virtual applications building on it.

Such a system provides multiple isolation layers: Standard processor privilege rings and memory paging protect processes executing within a virtualization. Hardware support for monitoring all privileged CPU instructions enables the hypervisor to transparently isolate virtualization instances from each other. Finally, the chipset is able to block direct memory accesses (DMA) of devices to defined physical memory areas, thus allowing the hypervisor to control device I/O.

2.3 Dynamic Switch to Trusted State

Modern platforms from AMD [2] and Intel [23] extend the basic TCG model of a *static* chain-of-trust anchored in a hardware reboot. They provide the option of a *dynamic* switch to a trusted system state. In this paper we focus on Intel's *Trusted Execution Technology* (TXT), which we build our implementation on. Similar functionality is provided by AMD's Secure Virtual Machine (SVM).

A so-called *late launch* is initiated by the special Intel TXT CPU instruction `GETSEC[SENTER]`. It stops all processing cores except one. The chipset locks all memory to prevent outside modification by DMA devices and resets PCRs 17 to 22. A special Intel-provided and cryptographically signed *Authenticated Code Module* (ACM) starts a fresh chain-of-trust after setting the platform into a well-defined state. This provides a *Dynamic Root of Trust for Measurement* (DRTM).

Subsequently, a *Measured Launch Environment* (MLE) [26] is first measured and then executed. Piece-by-piece the MLE decides which system resources to unlock and thus cautiously restores normal platform operation. The platform remembers that she is running in "secrets" mode and automatically enforces memory scrubbing whenever a system (re)boot is initiated.

The ACM is also capable of enforcing specific *Launch Control Policies* (LCPs). Here, the ACM measures the MLE and compares it with the trusted LCP stored in the non-volatile memory of the TPM. Changes to the LCP can only be authorized by the TPM owner. Any other, not authorized software configuration is not allowed to continue; the ACM will reset the platform.

3 An Integrated Architecture for Enhanced Security on Off-the-shelf PC Platforms

In the following we present our design of the acTvSM virtualization platform, which is able to enforce the integrity of software services. It is built for flexible operation and allows for the practical protection of commodity software.

3.1 Identification of Platform Security Properties

First, we present the security goals we want to achieve and the leitmotifs we want our architecture to follow.

Defined platform states The main objective of our platform is to guarantee that only well-defined software configurations are executed. The key prerequisite for building a chain of trust that precisely describes such a software state is to start measuring from a well-defined, known secure hardware state. Intel's

TXT with its DRTM provides exactly this. Furthermore, the advanced hardware isolation primitives of TXT platforms allow us to contain (malicious) code so that it cannot alter other partitions' memory (and thus state).

Consistent, expressive and deterministic chain-of-trust In order to document a system's configuration the platform must be able to account for what components are involved in the boot process. We use the TPM provided set of PCRs to guarantee the integrity and authenticity of measurements, but special care must be taken to achieve expressive PCR values. Mapping the intricacies of PC system configurations into a few PCRs is a difficult task: For a deterministic result the measurements must be stable, the number of measurements finite, and their order constant. These properties allow us to calculate the expected PCR values *a priori*. Only then it becomes practically feasible to *seal* data and the desired executable code to *known-good* configurations.

Integrity Guarantees in Trusted States A security sensitive application and its working data should never be stored on platform storage media in plaintext. Instead, we encrypt the file system by default, and only when the platform attains the good running configuration defined by the system administrator, access and modifications are possible. As unsealing can only occur at runtime, this prevents off-line attacks and limits attacks by running maliciously modified software. Transparent encryption implicitly provides us with integrity protection of executables and keys can be managed automatically, as they are protected by sealing.

Attestation friendly An independent remote third-party should be able to request an attestation of the services offered on the platform. Protection of privacy is an essential requirement for Remote Attestation and PCR information should not unnecessarily link the service to the platform manufacturer, owner, operator or user - instead, if an application requires this information it should explicitly and transparently to the user include it in the reporting process. The TXT measurement chain provides this separation of concerns as is starts from the ACM, identifying only the general chipset generation and the following components.

Another challenge is keeping track of known-good system PCR configurations. This proves extremely challenging due to the high number of possible combinations found in today's complex and ever-updated operating systems [18]. However, with virtualization in place the hypervisor can be assigned to perform this task. In our architecture, we choose to do measurements of complete file system images instead. We measure at *file system granularity* and therefore greatly simplify the comparison against known-good values. This allows practical application of Remote Attestation.

Appropriate Usability We do not want to restrict the choice of software the platform may execute. Our platform should allow to install any application and define it as trusted. In its partitions, it may also execute images which contain general purpose software, which does not require or provide elevated security levels. The overhead to maintain configuration integrity and to perform updates should be reasonable. Mechanisms to back-up applications and data must exist.

We believe that appropriate usability is needed for practical applications and so we target professional system administrators, who we entrust with the critical maintenance operations of the platform.

3.2 Proposed Architecture

We now outline a practical architecture for use on contemporary PC platforms. The components used in our architecture can be broken down into different blocks, namely Secure Boot, Base System, Trust Management and Virtualization partitions. Figure 1 illustrates the roles and execution flow of each of these blocks. For implementation details see Section 4.

The **Secure Boot** block is responsible for initializing the system to a predefined configuration. This requires close cooperation of hardware and software. We use an Intel TXT enabled physical platform.

Upon power-on, the platform performs a conventional boot, but does not start an operating system; instead, a Measured Launch Environment (MLE) is prepared and a TXT *late launch* is performed. This is accomplished by cooperation of a bootloader that prepares the system and the hardware-vendor provided Authenticated Code Module (ACM). The precise, desired software configuration can be specified by the administrator in the form of two policies stored in the TPM. The Launch Control Policy (LCP) is evaluated by the ACM and specifies which MLE is allowed to be executed. The MLE is configured with a Verified Launch Policy (VLP), which contains known-good values for measurements of the system kernel and its configuration. For the remainder of the paper we assume that the platform is built around the Linux Kernel. The temporary in-memory filesystem environment `initramfs` contains code and data to configure devices and to perform complex initialization operations, especially PCR extend and unsealing, early in the boot process. Thus, a secure boot is performed into a hardware guaranteed state and the chain of trust is extended over the kernel and its configuration. If the measurements do not match the expected values provided by the policies, the platform will shut down. Else, the startup code unseals the cryptographic keys needed to mount the file system which contains the Base System. Also, an unbroken chain-of-trust is ensured by measuring the filesystem image of the Base System into a PCR before it is mounted.

Together with the kernel, the **Base System** takes the role of the hypervisor. It manages the hardware, provides virtualization services to virtual applications and protects their integrity at run-time. As a critical component, it must be a mature, reliable technology and actively maintained with regular security updates. To support deterministic PCR measurement, the Base System file system must be read-only. A seemingly contradiction is the requirement of a read-write file system for Linux platform operation. To solve this dilemma, we merge a ephemeral file system with the measured image. As changes to the Base System do not survive a reboot of the platform - except by explicit patching or system update (see Section 3.3) - this ensures robustness of the base system image to malicious modifications.

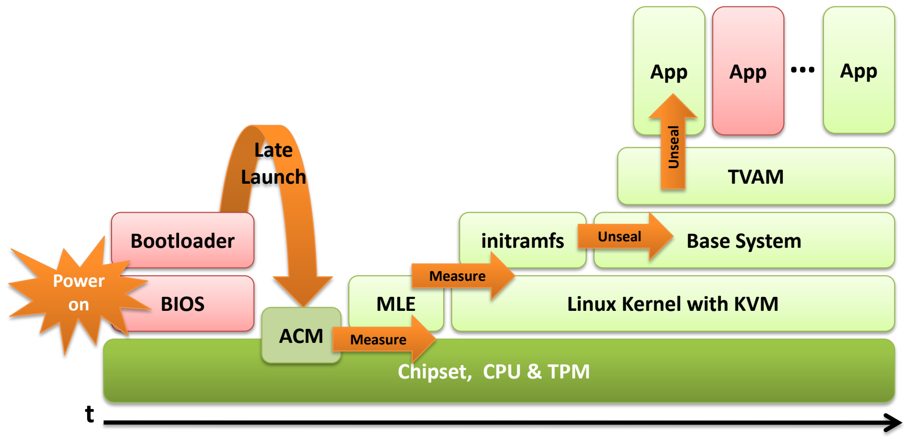

Fig. 1. Overview of the interaction of the main components of the platform. Trusted components are green, untrusted are red. The timeline indicates the different phases of platform boot and operations.

Management of the virtual applications itself is done by a component called TVAM, the **Trusted Virtual Application Manager**. It is the central component which manages the import, initialization, startup, cleanup etc. of virtual applications. The images only exist encrypted on storage media. The cryptographic key to decrypt and run a specific application is sealed to a specific system (PCR) configuration. TVAM therefore unseals key data, and if the system is in exactly the trusted configuration, TVAM is able to startup a virtualization partition, mount it and execute the desired application. Another task of TVAM is to calculate future trusted states, prepare the necessary policies and to reseal data and keys.

Virtualization Partitions may host any system normally running standalone. This can be a unmodified out-of-the-box Linux or Windows system, or a special purpose system. Multiple partitions can be started in parallel, connected by network interfaces. This allows to set-up multiple services. TVAM is capable of continuing the chain-of-trust into the application layer by measuring virtual partition images before start. However, if stable PCR measurements are expected the image must be read-only with read-write storage provided externally.

3.3 Operating the Platform

In the previous section, we outlined the basic components for booting the system and starting one or more partition. We now identify and describe basic platform operations which allow the initialization and long-term maintenance of our setup.

Installation In order to start from an initial trusted state the software needs to be distributed on a trusted medium. A root of trust for installation can be ensured by read-only media such as CD-ROM and appropriate organizational measures. Once booted from it, the installation routine wipes the system's hard-disk(s) and installs a default configuration of the platform. Immediately, the measurement values for the MLE, the platform's filesystem, kernel and initial ramdisk are calculated and appropriate policies are stored in the TPM. Already in this early stage, the platform is ready to do its first reboot into trusted mode.

Update and Application Mode After a successful late launch, the platform runs in *Update Mode*, where it waits for maintenance commands. In Update Mode the platform is capable of upgrading software packages. At the end of this process, the same procedure is triggered that was run during installation. If no external login attempt is received, the platform switches into *Application Mode*. In this mode TVAM unseals the decryption key of application images and mounts them. TVAM then extends a PCR to document the state transition and finally starts the applications.

Base System Update and Resealing In Update Mode the platform is capable of upgrading software packages of the base system. A remote administrator can login and update the system as needed. After that, the base system image is assembled, compressed and linked in the bootloader menu. As all data blobs holding the decryption keys for the application partitions are sealed to the old platform state, they must be (re)sealed to the new one. If the kernel or its `initramfs` was updated, a new VLP is written into the TPM. This also applies to a MLE update, where a new LCP needs to be written.

Small Updates The full update procedure outlined in the previous paragraph may be cumbersome for minor configuration changes such as a change of a static IP address in some configuration file. Instead, a "patch" facility allows the remote administrator to provide a patch file on a separate partition. The authenticity of the patch is guaranteed by a cryptographic signature added by the administrator - the certificate to validate it is contained within the platform image. Upon next full system update these patches are automatically integrated into the system. This mechanism allows for small bug fixes and easy distribution of pre-configured system platform images for instance in homogeneous datacenters, where machines only vary in small configuration details.

Application management TVAM provides a comprehensive feature set for the management of application images. This allows the import, backup and deletion of images, along with the required encryption and TPM (un)sealing operations. The simplest way to install an application image is to import a raw disk image. This allows the convenient creation and customization of virtual applications on remote hosts. TVAM will create an encrypted storage space and copy the raw data to it. The access secret is sealed to the trusted platform configuration and stored. Since our architecture can handle multiple secrets for each encrypted disk, the administrator can also provide a backup secret, which can be stored securely, i.e. on a smart card. This allows the administrator to decrypt the image even in case the TPM would be damaged or failed. This flexibility is also useful for migrating an image to a different machine.

External Reboot A remote way to force a reboot of the platform is an important feature for administration. Intel TXT capable platforms are usually combined with a set of features called Intel Active Management Technology (AMT) [24]. AMT allows, amongst other features, externally triggered reboots as well as serial console redirection via a network connection. A reboot will cause the platform to boot into a trusted state, regardless of the runtime state it was before. In some environments this may be warranted at a regular basis to limit the attack surface.

Trusted Administration Channel The Administrator can access the platform via SSH in Update Mode. Remember that a policy update process requires the TPM owner password. Before the administrator provides this password, she a) must confirm that she is connected to the right platform and b) that the platform is in the correct Update Mode configuration. The first constraint demands that the client must verify that the server always presents the same trusted public key. Second, we seal the SSH daemon's private key to Update Mode. If no external log-in attempt is received, the platform switches into *Application Mode*. This is performed as follows. The SSH daemon is stopped and its private key is removed, and the PCRs are extended to document the state transition and prevent further access to the TPM sealed blobs. Finally, applications are started.

4 Implementation

We assembled a prototype of the platform described in Section 3. The following subsections give an in-depth description of implementation details and our storage management concept and provide a security analysis.

4.1 Components

In theory, any Linux distribution can be customized for our architecture. In practise, a substantial number of patches and changes were needed to assemble a working prototype and available Trusted Computing utilities needed modifications.

Secure boot is accomplished by using a standard bootloader (GRUB[3]) along with `SINIT` and `tboot` [25]. `SINIT` is Intel's implementation of an ACM, while `tboot` is Intel's prototype implementation of an MLE (see Section 2.3). Upon boot GRUB loads SINIT, tboot, the kernel and its `initramfs` into memory and executes tboot, which sets up the ACM and then late-launches into it. The authenticity and integrity of the ACM code is guaranteed under an Intel private key, of which the public part is hardwired into the chipset. The ACM's task is then to measure the tboot binary and compare it to the LCP. Tboot takes over and continues to measure the kernel and `initramfs` and compares them against the VLP. Once the integrity of the kernel and its environment has been assured, control is passed to it and the standard boot process continues. Customized 64-bit ports of tools from IBM's *TPM-utils*[41] provide the PCR extend and unsealing capabilities in the initial ramdisk (`initramfs`) environment.

[3] `http://www.gnu.org/software/grub/`

In our architecture, we use a customized Linux operating system augmented with the *Kernel-based Virtual Machine* (KVM) [38,28] hypervisor module, which is fully maintained in mainline Linux. KVM can run multiple virtual machines on x86 CPUs equipped with virtualization mode extensions. It extends the Linux Kernel to offer, besides the existing Kernel and User modes, an additional Guest process mode. Each virtual machine offers private virtualized hardware like a network card, hard disk, graphics adapter, etc. Those virtual devices are forwarded to *QEMU* [6], a fast software emulator. QEMU can emulate all standard hardware devices of a x86 platform, including one or several processors. For the Base system, we use packages from the x86_64 *Debian Linux* lenny[4] release. It acts as the host for the virtualization partitions. This is a pragmatic choice for a robust base system, as Debian is known for its emphasis on a stable code base, frequent security releases and conservative approach to adding immature features. To support current Trusted Computing and virtualization hardware we need to add selected packages from the Debian testing tree. For example, only Linux kernels 2.6.32 or newer integrate Intel TXT support and drivers for chipsets that implement it. Scripts for installation, initial ramdisk management and rebuilding of the Base System image are taken from Debian and customized to our needs. The system bootstrap scripts for creation of distributable and bootable CDs for initial installation are taken from GRML Linux[5], a distribution specialized for system administrators.

The runtime architecture of our platform is outlined in Figure 2. After successful boot, the architecture supports several virtual applications. Raw hard disk storage space and the hardware TPM are assigned to the emulated hardware provided to each virtual application.

The implementation of the Trusted Virtual Application Manager in the Base System is scripted in the Ruby language. TVAM offers operations to define new trusted system states and re-seal the base system and applications to them. It also adds and removes system states from the list of trusted states in the launch policies. It allows to import an application image into the platform, start, stop, list and remove virtual applications. The more complex operations such as calculating trusted states, sealing, unsealing or policy creation and storage in TPM NV-RAM are performed by *jTpmTools* and *jTSS* from IAIK's "Trusted Computing for Java" project [34].

After boot, PCR 17 holds the LCP and details on the MLE [26]. PCR 18 contains the measurements (hashes) of MLE and kernel module and the respective command lines (4 hashes). PCRs 19-22 contain the elements described in the VLP - the modules defined in the bootloader configuration (kernel, `SINIT` module, `tboot` binary and `initramfs`). The read-only image of the Base System is extended to PCR 14. PCR 13 is assigned for the measurement of virtual applications. We use PCR 15 to indicate the transition from Update to Application Mode. These state descriptions can be used to seal data, especially file system encryption keys, to known good values in all phases of the boot process.

[4] `http://www.debian.org/releases/lenny/`
[5] `http://grml.org/`

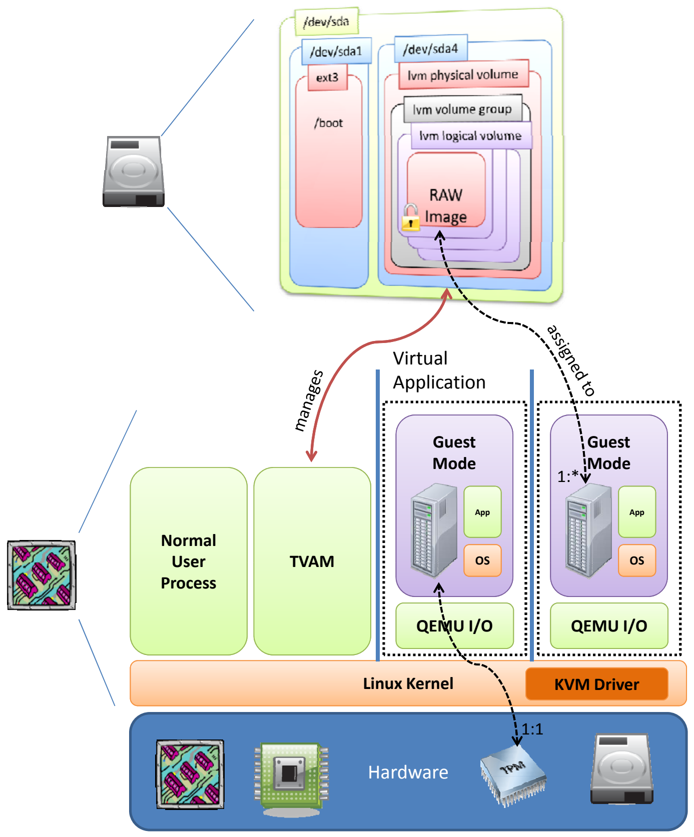

Fig. 2. The hardware platform consists of RAM, CPU, TPM and harddisk storage. KVM/QEMU runs applications providing isolated memory and defined storage areas. TVAM is the process that manages the base system configuration and set-up, start and tear-down of applications. The hardware TPM can be forwarded to applications.

4.2 Disk Storage Management

We use a complex disk layout with different file systems to create a measurable platform. Such a on-disk structure is automatically generated at initial installation and maintained by the TVAM. An exemplary layout of our architecture's structures is depicted in Figure 3. In the example, the system harddisk is `/dev/sda` and divided into two partitions.

The first partition (`sda1`) contains a read-write filesystem hosting all the components necessary for the platform boot process. This encompasses the bootloader, `tboot`, `SINIT` and Linux kernel plus associated initramfs images. The remainder of the harddisk storage (`sda4`) is allocated as a Logical Volume Man-

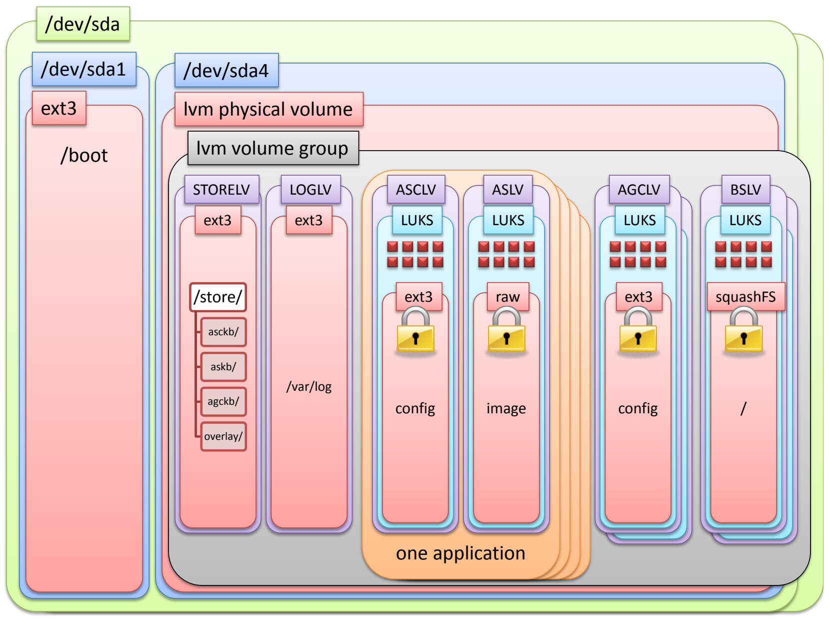

Fig. 3. The detailed disk layout of the platform. All trusted code and data of the base system and of the virtual applications is stored on logical volumes (LV) and sealed to PCR states. See Section 4.2 for a detailed description and explanation of the acronyms.

ager (LVM)[6] dynamically managed space, which is assigned to a single LVM volume group. This volume may be assembled from multiple physical volumes distributed over multiple physical disks, enabling online resizing and RAID technologies.

The LVM managed volume group contains both plaintext and encrypted logical volumes (LVs). These block devices are transparently encrypted with Linux kernel's `dm-crypt` subsystem. Each LV contains either a file system for use on the platform, or raw storage which is assigned to, and managed by, virtual partitions. `dm-crypt` encrypts block devices with symmetric AES keys, called masterkeys. Each encrypted LV contains a Linux Unified Key Setup (LUKS) [19] partition header, which is responsible for key management. LUKS provides 8 key slots that encrypt the same masterkey with different access secrets. If a client knows one correct secret for one of the key slots, it may decrypt and access the data. This abstraction layer adds flexibility and provides several advantages: The masterkey can be generated with high entropy, independent of user input or password memorization. Key slot secret changes do not require re-encryption of the whole block device. Key slots can also be assigned to maintenance processes, such as backup. The plaintext logical volume *STORELV* contains the TPM

[6] `http://sourceware.org/lvm2/`

sealed data structures holding the primary access secrets for the encrypted logical volumes.

As a running Linux system requires some writable file system, the root "`/`" file system of the platform is assembled from multiple layers via `aufs`[7]. Figure 4 illustrates this process which is performed at boot time. Base System Logical Volumes *BASELV* contain compressed read-only `squashfs` images which contain binaries and configuration files of the Base System. They are measured by `initramfs`. `aufs` merges this `squashfs` with an in-memory `tmpfs` to provide writable, but ephemeral storage. In addition, we copy administrator signed configuration patches from *STORELV*, after verification of the signature. Thus, we create a runtime read-write file system which is based on authenticated images with robust and deterministic hash values. In Update Mode, we can even create these images and pre-calculate their expected measurement values *in situ*. This unique feature enables our platform to update from one trusted configuration to another without exposing sealed data.

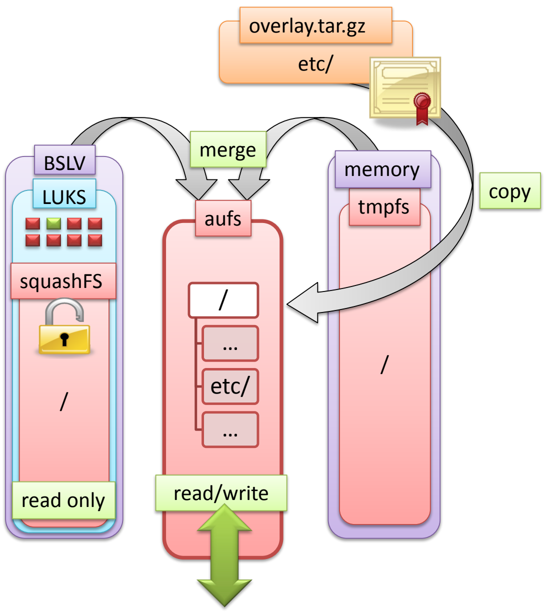

Fig. 4. The writable platform root file system is merged at boot time from a read-only static image and an in-memory file system. An authentic configuration patch for minor adjustments is integrated.

In the scenario of a service that encompasses multiple partitions running in parallel, i.e. a webservice front-end, a database partition and a firewall parti-

[7] `http://aufs.sourceforge.net/`

tion, configuration files for such application groups are stored in the Application Group Specific Configuration Logical Volume *AGCLV*. The configurations specify which virtual partitions are to be started, which platform resources are assigned to them, and how they interconnect.

Each Application Specific Logical Volume *ASLV* contains one virtual partition application image. The configuration in the AGCLV specifies if additional logical volumes containing Application Specific Configuration data *ASCLV* (or simply more mass storage) are attached. The *LOGLV* keeps system logs.

4.3 Performance

The prototype size of the Base System installation image is 402 MB. We use a HP dc7900[8] with Intel Core 2 Duo E8400 CPU clocked at 3GHz, 4GB RAM, Intel Q45 Express Chipset and Infineon TPM 1.2 as reference platform. Installation and setting up the encrypted file systems on such a system is done in just under 15 minutes. The measurement and integrity enforcing mechanisms of our platform are only performed at boot time, which takes 57 seconds from power-on to displaying the login-prompt. The added delay by the late launch at boot time is 5 seconds. The Base System kernel boot time and runtime performance of applications is not different to what can be expected from a Linux/KVM platform with hard disk encryption done in software and QEMU device emulation and not influenced by our measurement and boot-time enforcement process.

4.4 Shared TPM Access

Several different services may access the singleton TPM. The Base System needs the hardware TPM only at start-up. After that, it can be assigned to application partitions, one at a time. We enhanced QEMU to allow byte stream forwarding to the `/dev/tpm` interface in the base system in a non-blocking manner. Consequently, one virtualized partition can run Trusted Computing applications at a time. More general approaches have been proposed to solve multiplexing of the hardware TPM for multiple partitions, for example virtual software TPMs [7,43]. Full support in the acTvSM platform is planned for future work.

4.5 Security Analysis

The previous sections gave an overview of the workings of our platform. In this section we reflect on several of the security design features, their consequences and countermeasures.

[8] Note that in our experiments some other hardware platforms suffered permanent damage during late launch due to BIOS bugs.

Attacker Model As we base our platform on TCG's TPM and Intel's TXT technology our assumptions on the adversary and possible attacks closely follow those of the hardware designers [23]. Attackers will attempt either to modify the state measurements, extract cryptographic materials from hardware, manipulate code execution, or attempt control of the base platform or the applications executing on top of it. Commodity devices can only be assumed to protect against very simple hardware attacks. Hardware security therefore depends on physical access control and diligent operational procedures in datacenters and other deployment scenarios. Thus, we consider software and network attacks only. Further, attackers are also assumed to have control over external network traffic, but are unable to break cryptography [14]. Also, we do not consider side effects of System Management Mode (SMM), the overruling processor mode for low-level system event management by the BIOS. In contrast to TCG's security model we fully trust the system administrator to handle and back-up secrets and key material of the platform and of applications.

Security Analysis Our platform does not protect all data on the system at all times. However, the TPM protects the sensitive cryptographic data (keys) that the platform uses to guarantee the integrity of itself and of the applications. The platform fully utilizes the hardware based PCR mechanisms that protect measurements in the chain-of-trust of the components booted. Thus, a malicious boot with a following attempt to start a virtual application will fail if any part in the chain of trust was modified. Therefore, our platform can ensure that a trusted system state is reached after boot.

However, the general time-of-check-time-of-use [8] problem remains: As the security of a program is in general undecidable, successful runtime attacks cannot be ruled out. Still, they are much more difficult to perform in our architecture when compared to systems without TXT based integrity enforcement.

Within each partition, priviledged system software supports normal user applications which communicate to the outside. Security history suggests that (almost) any application can be exploited remotely and taken control of. The malicious elevation of application privileges to system privileges may follow. Horizontally, such a compromised partition will be contained by the Intel VT feature. Vertically, attackers could further attempt to jail-break from the KVM/QEMU environment, the security of which is not easy to assess: KVM itself is a rather small piece of code encouraging security evaluation, but QEMU, which provides the emulation for devices associated with a KVM instance is more complex. Further, QEMU runs as standard user process in the base system and is not hardware isolated. Naturally, the more devices are emulated by QEMU, the higher is the possibility of a bug which would allow tampering.

If an attacker succeeds as far as obtaining root privileges in the Base System layer, access to the encrypted application file images currently mounted at that time is possible, as the required key material is kept in kernel memory. Still, a security break-out would not be able to permanently modify the Base System, since updating the secure boot chain to a new configuration requires a rewrite of the policies in the TPM, which can only be performed with TPM owner permission. However, the owner password is neither stored on the system nor transmitted in clear text. The next reboot will either restore a trusted state or fail.

Therefore, our platform empowers the administrator to freely define what is known-good and trusted. It also clearly defines the boundaries of each service, so that those exposed interfaces can be inspected meticulously. We believe that this allows an structured approach for forming an informed opinion on the security of specific configurations and enforcing it in practice.

5 Related Work

Security critical code can be isolated using hardware security co-processors [3], like the IBM 4758 [15]. An early example of extending the trust from dedicated hardware security modules into applications is given in the Dyad System [50]. AEGIS [4] is an early mechanism to support secure boot on PC platforms assuming a trusted BIOS. The Enforcer platform [29] and IBM's Integrity Measurement Architecture [42] show how to integrate TCG-style static measurements into the Linux environment. While this collects precise information, it does not always allow to identify a limited number of possibly good configurations. Instead of individual files, file system images have been used to transport user software and data with SoulPads [10] or Secure Virtual Disk Images in grid services [22] between platforms.

Early demonstration of a system that makes use of the advantages of virtualization for security and trust are PERSEUS [33] and Terra [20]. The Nizza virtualization architecture [46] extracts security critical modules out of legacy applications, transferring them into a separate, trusted partition with a small TCB. None of these platforms relied on the TPM.

Microsoft's now apparently inactive NGSCB [17] envisioned the security critical Nexus kernel to provide an environment for security critical services, while running a legacy OS in parallel. The EMSCB [16] platform demonstrates TPM-based Trusted Computing on an L4-based virtualization platform. The OpenTC [32] project demonstrated a system based on a static chain-of-trust from the BIOS to the bootloader via Trusted Grub to Xen [5] or L4 hypervisors, and into application partitions measured and loaded from CD images. An example of a trusted installation medium is described in [12]. [13] describe a Xen-based platform which is focussed on Remote Attestation. Schiffman et al. [44] describe an all-layer (Xen hypervisor to application) integrity enforcement and reporting architecture for distributed systems. Cabuk et al. [9] propose to use a software-based root of trust for measurement to enforce application integrity in federated virtual platforms, i.e. Trusted Virtual Domains [11].

Other recent proposals have directly supported hardware virtualization or a *dynamic* switch to a trusted system state. Vasudevan et al. [51] discuss general requirements for such systems. BIND [45] uses AMD's Secure Virtual Machine (SVM) [2] protection features to collect fine grained measurements on both input and the code modules that operate on it so that the computation results can be attested to. Flicker [31] isolates sensitive code by halting the main OS, switching into AMD SVM, and executing with a minimal TCB small, short-lived pieces of application logic (PALs). PALs may use the TPM to document their execution and handle results. TrustVisor [30] is a small hypervisor initiated via the DRTM process. It assumes full control and allows to manage, run and attest multiple PALs in its protection mode, however without the repeated DRTM mode switch costs incurred by the Flicker approach. Intel's P-MAPS [39] launches a tiny hypervisor parallel to a running OS to protect process integrity, hidden from the OS. The hypervisor authenticates code and data areas, which are protected in-place and can only be accessed via well-defined interfaces. LaLa [21] performs a late launch from an instant-on application to boot a fully fledged OS in the background. OSLO [27] is an OS loader module which implements a dynamic switch to a measured state in the OS bootchain on AMD SVM systems.

6 Conclusions

In this paper we present a practical architecture for mass-market off-the-shelf PC hardware that combines Trusted Computing and hardware virtualization to provide a platform that offers integrity guarantees to the applications and services it hosts.

Our architectures achieves a distinct set of characteristics, the combination of which sets it apart from previous results. We take advantage of Intel TXT to shorten the chain-of-trust and make use of the KVM hypervisor to separate service partitions. This choice of virtualization technology frees us from restrictions in the choice of guest operating systems and at the same time isolates memory areas of virtual applications by hardware mechanisms. While other approaches measure the executed binaries only on fragment ([45,31,30]), or process ([39]) granularity, we measure a structured set of file systems. This contribution simplifies the problem of defining known good states and allows to do integrity enforcement in practical scenarios. Besides just sealing to current states, our architecture allows to calculate future trusted configurations, with trusted state transitions that do not expose data. We empower administrators to easily customize and update the platform offered and to enforce these configuration. Our prototype implementation shows that any OS or application can be run inside the partitions, and that commodity Linux distributions can be adapted this way.

Acknowledgments

This work was supported by the Österreichische Forschungsförderungsgesellschaft (FFG) through project acTvSM, funding theme FIT-IT, no. 820848. We thank Andreas Niederl and Michael Gebetsroither for insightful discussions and their help in implementing the prototype and the anonymous reviewers for their helpful comments on the paper.

References

1. Adams, K., Agesen, O.: A comparison of software and hardware techniques for x86 virtualization. In: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems. pp. 2–13. ACM, San Jose, California, USA (2006)
2. Advanced Micro Devices: AMD64 Virtualization: Secure Virtual Machine Architecture Reference Manual (May 2005)
3. Anderson, R., Bond, M., Clulow, J., Skorobogatov, S.: Cryptographic processors-a survey. Proceedings of the IEEE DOI - 10.1109/JPROC.2005.862423 94(2), 357–369 (2006)
4. Arbaugh, W.A., Farber, D.J., Smith, J.M.: A secure and reliable bootstrap architecture. In: Proceedings of the 1997 IEEE Symposium on Security and Privacy. p. 65. IEEE Computer Society (1997)
5. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles. pp. 164–177. ACM, New York, NY, USA (2003)
6. Bellard, F.: Qemu, a fast and portable dynamic translator. In: ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference. pp. 41–41. USENIX Association, Berkeley, CA, USA (2005)
7. Berger, S., Cáceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: virtualizing the trusted platform module. In: USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium. pp. 305–320 (2006)
8. Bratus, S., D'Cunha, N., Sparks, E., Smith, S.W.: Toctou, traps, and trusted computing. In: Trusted Computing - Challenges and Applications. LNCS, vol. 4968. Springer Verlag (2008)
9. Cabuk, S., Chen, L., Plaquin, D., Ryan, M.: Trusted integrity measurement and reporting for virtualized platforms. In: Chen, L., Yung, M. (eds.) INTRUST 2009. Lecture Notes in Computer Science, vol. 6163, pp. 180–196. Springer (2009)
10. Cáceres, R., Carter, C., Narayanaswami, C., Raghunath, M.: Reincarnating pcs with portable soulpads. In: Proceedings of the 3rd international conference on Mobile systems, applications, and services. pp. 65–78. ACM, Seattle, Washington (2005)
11. Catuogno, L., Dmitrienko, A., Eriksson, K., Kuhlmann, D., Ramunno, G., Sadeghi, A.R., Schulz, S., Schunter, M., Winandy, M., Zhan, J.: Trusted virtual domains - design, implementation and lessons learned. In: Chen, L., Yung, M. (eds.) INTRUST 2009. Lecture Notes in Computer Science, vol. 6163, pp. 156–179. Springer (2010)
12. Clair, L.S., Schiffman, J., Jaeger, T., McDaniel, P.: Establishing and sustaining system integrity via root of trust installation. Computer Security Applications Conference, Annual 0, 19–29 (2007)
13. Coker, G., Guttman, J., Loscocco, P., Sheehy, J., Sniffen, B.: Attestation: Evidence and trust. Information and Communications Security pp. 1–18 (2008), `http://dx.doi.org/10.1007/978-3-540-88625-9_1`
14. Dolev, D., Yao, A.C.: On the security of public key protocols. In: Information Theory, IEEE Transactions on. Stanford University, Stanford, CA, USA (1981)
15. Dyer, J., Lindemann, M., Perez, R., Sailer, R., van Doorn, L., Smith, S.: Building the ibm 4758 secure coprocessor. Computer 34(10), 57–66 (2001)

16. EMSCB Project Consortium: The European Multilaterally Secure Computing Base (EMSCB) project (2004), `http://www.emscb.org/`
17. England, P., Lampson, B., Manferdelli, J., Willman, B.: A trusted open platform. Computer 36(7), 55–62 (July 2003)
18. England, P.: Practical techniques for operating system attestation. In: Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies. pp. 1–13. Springer-Verlag, Berlin, Heidelberg (2008)
19. Fruhwirth, C.: New methods in hard disk encryption. Tech. rep., Institute for Computer Languages, Theory and Logic Group, Vienna University of Technology (2005), `http://clemens.endorphin.org/publications`
20. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A virtual machine-based platform for trusted computing. In: Proceedings of the 19th Symposium on Operating System Principles(SOSP 2003). pp. 193–206. ACM New York, NY, USA (October 2003)
21. Gebhardt, C., Dalton, C.: Lala: a late launch application. In: Proceedings of the 2009 ACM workshop on Scalable trusted computing. pp. 1–8. ACM, Chicago, Illinois, USA (2009)
22. Gebhardt, C., Tomlinson, A.: Secure Virtual Disk Images for Grid Computing. In: 3rd Asia-Pacific Trusted Infrastructure Technologies Conference (APTC 2008). IEEE Computer Society (October 2008)
23. Grawrock, D.: Dynamics of a Trusted Platform: A Building Block Approach. Intel Press (February 2009), iSBN 978-1934053171
24. Intel Corporation: Intel active management technology (amt), `http://www.intel.com/technology/platform-technology/intel-amt/index.htm`
25. Intel Corporation: Trusted Boot (2008), `http://sourceforge.net/projects/tboot/`
26. Intel Corporation: Intel Trusted Execution Technology Software Development Guide (December 2009), `http://download.intel.com/technology/security/downloads/315168.pdf`
27. Kauer, B.: Oslo: improving the security of trusted computing. In: SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium. pp. 1–9. USENIX Association, Berkeley, CA, USA (2007)
28. Kivity, A., Kamay, V., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux Virtual Machine Monitor. In: OLS2007: Proceedings of the Linux Symposium. pp. 225–230 (2007)
29. Marchesini, J., Smith, S., Wild, O., MacDonald, R.: Experimenting with tcpa/tcg hardware, or: How i learned to stop worrying and love the bear. Tech. rep., Department of Computer Science/Dartmouth PKI Lab, Dartmouth College (2003)
30. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: TrustVisor: Efficient TCB reduction and attestation. In: Proceedings of the IEEE Symposium on Security and Privacy (May 2010)
31. McCune, J.M., Parno, B.J., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: an execution infrastructure for tcb minimization. In: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008. pp. 315–328. ACM, Glasgow, Scotland UK (2008)
32. OpenTC Project Consortium: The Open Trusted Computing (OpenTC) project (2005-2009), `http://www.opentc.net/`
33. Pfitzmann, B., Riordan, J., Stueble, C., Waidner, M., Weber, A., Saarlandes, U.D.: The perseus system architecture (2001)
34. Pirker, M., Toegl, R., Winkler, T., Vejda, T.: Trusted computing for the Java^{TM}platform (2009), `http://trustedjava.sourceforge.net/`

35. Pirker, M., Toegl, R.: Towards a virtual trusted platform. Journal of Universal Computer Science 16(4), 531–542 (2010), http://www.jucs.org/jucs_16_4/towards_a_virtual_trusted
36. Pirker, M., Toegl, R., Gissing, M.: Dynamic enforcement of platform integrity (a short paper). In: Acquisti, A., Smith, S.W., Sadeghi, A.R. (eds.) Trust '10: Proceedings of the 3rd International Conference on Trust and Trustworthy Computing. LNCS, vol. 6101. Springer Berlin / Heidelberg (2010)
37. Popek, G.J., Goldberg, R.P.: Formal requirements for virtualizable third generation architectures. Commun. ACM 17(7), 412–421 (1974)
38. Qumranet: KVM - Kernel-based Virtualization Machine (2006), http://www.qumranet.com/files/white_papers/KVM_Whitepaper.pdf
39. Ravi Sahita, U.W., Dewan, P.: Dynamic software application protection. Tech. rep., Intel Corporation (2009), http://blogs.intel.com/research/trusted%20dynamic%20launch-flyer-rls_pss001.pdf
40. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: Hempelmann, C., Raskin, V. (eds.) NSPW. pp. 67–77. ACM (2004)
41. Safford, D., Kravitz, J., Doorn, L.v.: Take control of tcpa. Linux Journal 2003(112), 2 (2003), http://domino.research.ibm.com/comm/research_projects.nsf/pages/gsal.TCG.html
42. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX Security Symposium. USENIX Association, San Diego, CA (2004)
43. Scarlata, V., Rozas, C., Wiseman, M., Grawrock, D., Vishik, C.: Tpm virtualization: Building a general framework. In: Pohlmann, N., Reimer, H. (eds.) Trusted Computing, pp. 43–56. Vieweg (2007)
44. Schiffman, J., Moyer, T., Shal, C., Jaeger, T., McDaniel, P.: Justifying integrity using a virtual machine verifier. In: ACSAC '09: Proceedings of the 2009 Annual Computer Security Applications Conference. pp. 83–92. IEEE Computer Society, Washington, DC, USA (2009)
45. Shi, E., Perrig, A., Van Doorn, L.: Bind: a fine-grained attestation service for secure distributed systems. In: 2005 IEEE Symposium on Security and Privacy. pp. 154–168 (2005)
46. Singaravelu, L., Pu, C., Härtig, H., Helmuth, C.: Reducing TCB complexity for security-sensitive applications: three case studies. In: EuroSys '06: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006. pp. 161–174. ACM, New York, NY, USA (2006)
47. Strachey, C.: Time sharing in large, fast computers. In: IFIP Congress (1959)
48. Trusted Computing Group: TCG infrastructure specifications, https://www.trustedcomputinggroup.org/specs/IWG/
49. Trusted Computing Group: TCG TPM specification version 1.2 revision 103 (2007)
50. Tygar, J., Yee, B.: Dyad: A system for using physically secure coprocessors. In: Technological Strategies for the Protection of Intellectual Property in the Networked Multimedia Environment. pp. 121–152. Interactive Multimedia Association (1994)
51. Vasudevan, A., McCune, J.M., Qu, N., van Doorn, L., Perrig, A.: Requirements for an Integrity-Protected Hypervisor on the x86 Hardware Virtualized Architecture. In: Proceedings of the 3rd International Conference on Trust and Trustworthy Computing (Trust 2010) (Jun 2010)