

Firma: Disk-Based Foundations for Trusted Operating Systems

Kevin Butler, Stephen McLaughlin, Thomas Moyer, Joshua Schiffman,
Patrick McDaniel, and Trent Jaeger

Systems and Internet Infrastructure Security (SIIS) Laboratory
The Pennsylvania State University, University Park PA 16802

{butler, smclaugh, tmmoyer, mcdaniel, tjaeger}@cse.psu.edu

20 April 2009

Abstract

Secure boot mechanisms aim to provide guarantees of integrity of a system as it loads. It ensures that if a system is running, all of its process will satisfy integrity verification requirements. While secure boot has been available for a long time, it is not available in commodity systems due to the high cost of secure hardware. In this paper, we describe Firma, an architecture that provides secure boot functionality based on a storage root of trust. Unlike previous secure boot mechanisms, use of the disk can protect data secrecy by only releasing data to systems trusted not to leak data, while also providing data integrity through release to high integrity systems. We implement a prototype of Firma and show how it may be used to provide a trusted virtual machine monitor (TVMM) capable of supporting strong security guarantees for running VMs. Only minimal administration is required, and we detail the tasks necessary to support the architecture, showing new systems can be configured with a small number of automated steps. Our evaluation shows that Firma requires additional overhead of just over 1 second for the boot process.

1 Introduction

An important security goal is to only allow processes running code that is known to be high integrity to access security-sensitive data. For example, secrets should only be made available to processes trusted not to leak them. Also, high integrity data should only be made available to processes trusted to preserve their integrity and perform updates correctly. As loading a single process depends on the correct functioning of several software layers (e.g., BIOS, bootloaders, VMMs, etc.) in modern systems, a secure system requires a mechanism that verifies each of these layers is satisfactory and that protects access to security-sensitive data based on the results of this verification.

A *secure boot* mechanism aims to provide the guarantees described above. Secure boot was first introduced in the AEGIS system design [3]. Assuming that the machine's hardware is trusted, the integrity of a software layer is valid if and only if: (1) the integrity of each of its lower layers is verified to be valid and (2) transitions to a higher layer occur only after verification of that layer is complete. Secure boot ensures that if a system is running, then all its processes satisfy the verification requirement for integrity (i.e., at least up to the extent of secure boot). Further, certain components, such as operating systems and virtual machine monitors (VMM) are verified to enforce protection of security-sensitive data access. The need for these protections is particularly acute in VMMs, given that they must provide isolation assurances between the multiple simultaneously-executing virtual machines that they manage.

While the notion of secure boot has been around for a long time and there are several commercial systems that provide a form of secure boot (e.g., [7]), these secure boot approaches either require specialized hardware or depend on the careful management of cryptographic data. In the first case, special hardware is required to store and/or protect secrets that define the root of trust. AEGIS required a special PROM board [3], and current products still require hardware extensions [7]. In the second case, controlling the release of encryption keys serves as a form of secure boot, as the secret keys are only provided to systems that can be verified to meet specified integrity criteria [9, 29, 43], but such systems require significant key management. In addition, once the keys are released, all data may be accessed, so if low integrity software is subsequently run data protection may be circumvented.

Our aim is to enable secure boot on commodity systems with minimal management overhead. In this paper, we propose to leverage the processing capabilities found in modern disks to enforce secure boot. The idea is that the host must prove its integrity to the disk prior to receiving security-sensitive data. This enables the disk to protect the

data secrecy by only releasing data to systems trusted not to leak the data. Further, the disk can also protect data integrity by only releasing it to high integrity systems. Unlike previous secure boot mechanisms, using the disk as the mediation device also enables the protection of disk data updates. Only authorized systems can update particular disk volumes. We demonstrate this approach by constructing *Firma*, a novel storage architecture for securely booting a trusted virtual machine monitor (TVMM).

A major goal in our design of the Firma system is to enable secure boot with minimal administration. A key insight is that a Firma disk can act as a remote platform in an attestation protocol. The host can use its TPM to generate attestations that the Firma disk can verify prior to granting access to the next layer. Thus, Firma administration requires that enable a Firma disk to become an attestation client (e.g., obtain the host's TPM key and high integrity file information), and we show how the root-of-trust-installation method [5] can be used to supply this information easily. Finally, we note that a Firma disk verifier makes a good client for remote attestation as both the TPM and disk will be administered by the same party, so concerns about physical attacks on the TPM are not relevant, as any attackers with physical access would also have access to the disk.

In this work, we make the following contributions:

- We present the Firma architecture, which demonstrates how to perform secure boot of a VMM using storage as a root of trust. We show that intelligent disks may be used to measure the state of a host before allowing the hypervisor and privileged VMs to be loaded, providing guarantees of the VMM's integrity state.
- We detail the management tasks necessary to support the Firma architecture, showing that new systems can be installed and configured for secure boot in a **small number** of automated steps.
- We develop a prototype of Firma using commodity hardware and software. We evaluate the costs of attesting a hypervisor and show that the time to get a VM loaded from a securely-booted VMM requires an overhead of slightly over one second.

2 Background

Firma is reliant on the use of trusting computing functionality to provide information about a system's integrity state. It also makes use of storage augmented with autonomous security enforcement capabilities. We describe how these technologies are used in conjunction with Firma in the next section, but provide a short background describing the concepts of each below.

2.1 Secure Boot

One common threat to system security is the boot process. The presence of malicious code during the early boot phase (such as in the BIOS or bootloader) can compromise the security of the full system by disabling security mechanisms or loading additional malicious programs. Two categories of approaches, authenticated boot and secure boot, have been used to combat this threat. In both secure and authenticated boot, trust is axiomatically placed in an initial component that starts a code measuring processes.

The first approach, authenticated boot, enables a remote verifier to inspect the integrity of the host's boot process. This is done by performing measurements of every phase of the boot process before executing it and storing those measurements in a secure log. The Trusted Computing Group's (TCG) Trusted Platform Module (TPM) is an example of a device that supports authenticated boot [39]. It does this by securely storing measurements in hardware and using cryptographic keys to link the measurements to the physical machine.

The major drawback to authenticated boot is that it offers no security to the host. Even if a malicious program is measured, it is still running on the system. An alternative to authenticated boot is secure boot. A secure boot approach also measures code before loading it, but denies its execution if it does not meet some requirements. The end result is a system running only trusted code because only such code can be loaded.

An example of a secure boot device is IBM's 4758 secure coprocessor [7]. In addition to being built with tampering countermeasures, the 4758 uses code signing techniques to ensure only authorized code is loaded into the coprocessor's memory. Unfortunately, the device is too expensive for commodity use. AEGIS [3] is a secure bootstrap system that checks each boot phase against a special PROM containing trusted measurement definitions. However, AEGIS assumes the firmware in the physical hardware and parts of the BIOS are trusted.

2.2 Autonomous Storage

2.2.1 Trusted Storage

The TCG Storage Work Group has described specifications for interfacing with storage devices, considering the storage device to be a Trusted Peripheral (TPer). The primary specification in support of this effort is the Storage Architecture Core Specification [12], which primarily describes how to negotiate and establish a secure channel between the trusted disk and the host system, largely predicated on the existence of trusted send and receive commands specified in the SCSI and ATA specifications. In particular, the Storage Interface Interactions Specifica-

tion (SIIS) [41] describes the mapping of secure events, such as sending and receiving secure information to their equivalents in the SCSI command set [38] (i.e., SECURITY PROTOCOL IN/OUT) and ATA command set [2] (i.e., TRUSTED SEND/RECEIVE). The core specification provides a `StartSession` method for the disk and host to establish communication with each other and provides the basis for a portion of our key establishment with Firma. There is also support for pre-boot authentication, defined in the Opal SSC standard [42]. However, the specification does not consider the staged secure boot mechanism that we provide, nor a methodology for providing attestations of the system state to the disk from the host. The implications are that once the system is considered to be in a “good” state prior to boot, no further checks of system state are considered. Note that there are no current disks that support the trusted extensions to SCSI or ATA; many of the standards are still in the draft/discussion stage without any reference implementation.

2.2.2 Augmented Disks

Previous research into storage that independently enforces security properties has helped to guide our design approaches. Strunk et al. first independently enforced secure storage systems with the Self-Securing Storage project [35]. The basis of this work was isolating storage and providing extra data that would be inaccessible to the rest of the system, notably, an audit log that could be checked to ensure that violations against specific files were not occurring, or could be recovered from. As will be discussed in Section 5, our implementation of Firma also incorporates an audit log that is read-accessible to the user if system boot fails. Our approach differs in that we are not using the log as a general-purpose storage intrusion detection system, but rather as a means of diagnosing the cause of a boot failure (e.g., a failed attestation). Notably, this and other solutions that more explicitly use the disk as an intrusion detection system [23] do not prevent access to the disk if the host system is in a bad state.

From an architectural and implementation viewpoint, the Firma system resembles the rootkit-resistant disk architecture [4]. With rootkit-resistant disks, the system is designed to provide immutability of blocks based on insertion of a token by the user, who in effect forms a trusted path to the disk. This approach places minimal trust in the operating system. We differ, however, by showing that the system connected to *can* be booted to a trusted state, and proof of the system’s good state can be further provided through attestation directly to the disk, which can compare these measurements to recognized good states. These approaches are potentially complementary but are orthogonal in purpose from one another.

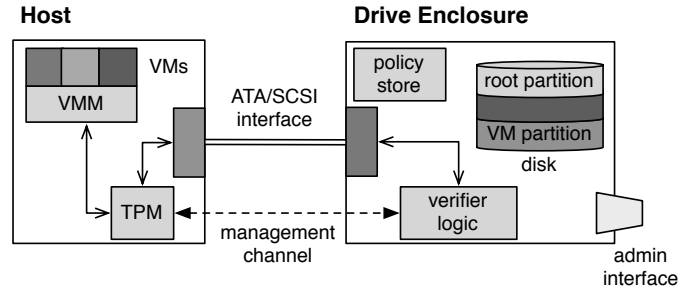


Figure 1: Overview of the Firma architecture. The disk is augmented with verification logic for receiving attestations from the host system, and stores secure policy information and cryptographic keys. Secure communication between the host and the disk can occur through enhancements to the disk protocol or out-of-band, such as with a hardware token delivering information between the devices.

3 Design of Firma

Firma is designed to protect on-disk storage by determining whether a host system is trusted to run an operating system, a process performed by *measuring* the state of the host from before boot time until the OS kernel is loaded from the disk. Firma can be used to protect a single system, by loading the OS kernel into the root partition of the disk. We focus on a use case of providing a trusted VMM, given the increased demands for system integrity necessitated by managing multiple simultaneously-executing OSes. The high-integrity system resulting from Firma facilitates trust that a VMM’s policies (e.g., full isolation vs sharing) are initialized to a good state. Figure 1 provides an overview of the components that comprise Firma. These are used to address the following design goals:

1. **Secure boot from storage:** Firma provides a staged secure boot mechanism that allows the disk to assess the state of the host system at set points of the boot process, such that if the BIOS or boot loader is compromised, it will be detected and no further boot processes will be allowed. This allows the host to provide trust guarantees and support services such as a trusted hypervisor, whose policy can be guaranteed by the system. Unlike AEGIS or use of secure co-processors such as the IBM 4758/4764, we do not require secure hardware beyond a disk augmented with functionality that is currently feasible.
2. **Continuous enforcement of system state:** Beyond load-time guarantees, Firma provides a means for the system to attest its state to the disk in a periodic manner. This architecture supports measurement of binaries loaded from the disk’s root partition, but is gen-

eral in scope and has the ability to support runtime integrity attestations.

3. **Ease of Management and Usability:** Once Firma is set up, there is nothing that needs to be done by the user in order to boot the system. Management of the disk may be performed by a user communicating directly with the device, so as to bypass the host in the event that it is in a bad state. We cannot necessarily rely on a secure channel that has a path through a potentially malicious host, as even if the host was unable to affect the confidentiality or integrity of data emanating from the disk, it could still arbitrarily drop the data. As a result, we consider that communication with the disk can involve plugging the disk directly into an administrative interface, or through the use of an administrative hardware token that can be physically carried by the administrator.

Firma is a protection system consisting of a disk augmented with a processor for providing cryptographic services (such as those available in an FDE drive [27]) and non-volatile memory (as found in a hybrid hard disk [28]) for security-critical keys and policy metadata, combined with a host with a mechanism for providing attestations (e.g., a TPM) to the disk. The disk and host communicate over a management channel out of band to regular requests to the disk. The disk contains a *root partition* that is static, and only released to the host once its good integrity state has been established. This partition may contain the kernel and binaries for a single operating system, and Firma is fully usable in this configuration. We consider a primary use case, however, of installing a virtual machine monitor onto the disk's root partition, allowing it to provide a trusted base for other virtual machines to be executed from the disk.

3.1 Threat Model

We consider an adversary capable of arbitrarily and completely subverting a host operating system. The OSes running on the virtual machines from the disk are outside of the purview of what can be protected at the storage level in our design, as we do not consider protections inside the OS. Other solutions are possible for protecting the OS at the storage level, including storage-based intrusion detection [23] and rootkit-resistant disks [4].

Many security architectures that consider virtualized environments make the assumption that the hypervisor and associated administrative VM are within the trusted computing base (e.g., [6, 22]). We do not make the assumption that these components are automatically trustworthy, given the ability of an adversary to attack storage by installing malicious software such as rootkits affecting a disk's MBR. In addition, while a virtual ma-

chine based rootkit (VMBR) such as SubVirt [17] may currently be unable to operate under a virtual machine monitor due to its lack of handling nested virtualization, self-virtualization in the x86 architecture through the Intel VT [21] and AMD SVM [1] processors makes the threat of VMBRs running under a VMM increasingly possible.

Our enforcement mechanism takes place within the disk. For our purposes, we assume that the attacker does not have the ability to mount physical attacks against the disk, such as opening the drive enclosure to attack the media or X-raying the disk. Depending on where the disks are deployed, these concerns may be valid; a design parameter for these drives can be physical security measures such as those used in secure coprocessors [33], and may be possible if the cost-benefit ratio makes this design point appropriate. Additionally, we consider physical attacks against the host system's TPM to be outside of our protection domain. One method of delivering policy that we describe in our implementation of Firma (see Section 5) is the use of a physical token. We do not consider physical attacks against the tokens; however, solutions for secure, tamper-resistant tokens such as those found in the IBM Zurich Trusted Information Channel [44] are possible where such defenses are deemed necessary.

4 Management and Operation

The basis for trusting Firma's VMM is predicated on its installation in a trusted manner and the ability to validate the integrity state of the host system. This section describes how a trusted installation process, pairing with the host, runtime operation may be used to provide these guarantees.

4.1 Installation

The installation of software necessary for Firma comprises installation of the VMM and kernel to the root partition of the disk. This process may be performed by the drive manufacturer at the time of fabrication, or by the user installing or updating the base software. In either case, the process requires use of a trusted installer. We use a root of trust installer (ROTI) [34], which establishes a system whose integrity can be traced back to the installation media.

The system performing the installation must contain a trusted platform module (TPM). In short, the TPM is a tamper-evident chip capable of performing some cryptographic operations, as well as providing non-volatile storage for root keys that it generates. Every TPM contains an *endorsement key* (EK), a 2048-bit RSA public/private key pair created when the chip is manufactured. As we describe below, this provides us with a basis for establishing

the unique identity of the TPM, which is essential to being able to verify the installation. The stages of this initial installation are as follows:

1. The installation media is loaded into the installer system, which contains a TPM. This system needs to be trusted, i.e., the hardware and system BIOS cannot have been subverted at this time.¹ As described in more detail below, the system’s *core root of trust for measurement* (CRTM), which contains the boot block code for the BIOS, provides a self-measurement to attest this good state.
2. An administrator authorizes the disk to accept writes to the root partition, and the VMM (and if necessary, supporting OS for privileged domains, i.e., *dom0* for the Xen hypervisor) is installed to the root partition off the custom installer. Every file in the root partition is hashed, and the list of hashes is kept in a file that is *sealed* (i.e., encrypted) by the installing system’s TPM. This process links the installing TPM with the installed code and the filesystem hashes.

The interface to the disk, and authentication mechanisms for the user, are defined by the device manufacturer; our prototype, described in more detail in Section 5, uses USB tokens inserted into the disk to deliver these capabilities (i.e., administrator access). Other solutions, such as the use of keypads or direct interfacing with a trusted console are also possible. Note that if the installing system is different from the host system (as would be the case if installation was performed by the manufacturer, for example), the CRTM measurement and the TPM key used to seal the hash list would be needed to verify the installation. Note that the TPM’s EK is not permitted off the module, so an encryption key is generated by the TPM for this operation. This encryption key, along with the CRTM measurement, can be supplied in an out-of-band manner (e.g., including the key on a slip of paper along with the disk or including it on a token), and specified directly to the disk through its administrative interface.

4.2 Deployment to the Host

In order to assure that we can attest to the integrity state of the host system, we need to *pair* it with the disk; in other words, we need to establish a method of establishing a unique binding between the host and the disk. We assume that the host system contains a TPM.

When a host system is first installed, it can be considered to be in a “greenfield” state, and we make the

¹This restriction is not necessary after the installation has occurred, as malicious changes to the system state will be measured by the CRTM.

assumption that there is no pre-existing malware or subverted hardware on the system at this stage. In a similar fashion to the installed system, we assume the ability to retrieve the CRTM measurement in an offline manner if the integrity of the hardware is a concern. Our first step is to retrieve information to identify the TPM in the host machine. While the EK is unique to the TPM, there are privacy concerns with exposing it, and it cannot be used to perform signatures [40]. Instead, an *attestation identity key* (AIK) public/private key pair is generated as an alias for the EK, and strictly used for signatures. However, while the EK is kept in the TPM’s non-volatile memory, the AIK is stored in volatile memory. Therefore, for it to be persistent across host system boots, we must store both the public and private AIK on the disk. We cannot reveal the private AIK; fortunately, the TPM provides another persistent key pair, the *storage root key* (SRK), used for encrypting keys stored outside the TPM. Thus, the SRK encrypts the private AIK before it is sent to the disk. Formally, the set of operations occurs as follows. Given a host’s TPM H and a disk D , the following protocol flow describes the initial pairing of the host to the disk and the initial boot:

Pairing

- (1) H : generate AIK = (AIK^+, AIK^-)
- (2) $H \rightarrow D$: $AIK^+, \{AIK^-\}_{SRK^-}$

Boot

- (3) $D \rightarrow H$: $\{AIK^-\}_{SRK^-}$
- (4) D : n = Generate nonce
- (5) $D \rightarrow H$: $Challenge(n)$
- (6) $H \rightarrow D$: $Attestation = Quote + ML$
- (7) D : $Validate(Quote, ML)_{AIK^+}$

Steps 1 and 2 occur prior to the boot process, while the subsequent measurements occur after the system has booted. Through the administrative interface to the disk, an administrator sends a command to allow the disk to operate without secure boot constraints for the initial system boot. This is done in order to provide a list of *measurements* that the disk can use to verify the system’s integrity state. The following states are measured in order: (a) the core root of trust for measurement (CRTM), (b) the system BIOS, (c) the bootloader (e.g., GRUB) and its configuration, (d) the VMM, and (e) the OS running on the VMM. We use the Linux Integrity Measurement Architecture (IMA) [25] as a framework for supporting attestation. Measurements are made by with the TPM’s *extend* operation, which hashes code and/or data, concatenates the result with the previous operation, and stores the result in the TPM’s *Platform Configuration Registers* (PCRs). The *quote* operation takes the challenger’s nonce n and returns a signature of the form $Sign(PCR, N)_{AIK^-}$, when the PCRs and n are signed by the private AIK. The measurement list (ML), which contains a log of all measurements

sent to the TPM, is also included in the attestation. At this point, the host must be restarted; this is a one-time operation.

The above process could also involve the setup of a session key associated with a secure channel between the disk and host, as described in Section 2. For example, the IKEv2-SCSI protocol [16] provides methods of establishing an IKE security association between the disk and the host. As there are many methods of performing such actions, we do not further elaborate on them here.

4.3 Regular Operation

On subsequent boots, the disk will revert to secure-boot mode. It will request an attestation prior to the VMM being loaded and will not allow the load to occur unless the attested integrity state matches the one obtained during the pairing. Similarly, after the VMM is loaded, a system measurement and subsequent quote is required before a new VM may be loaded from the VMM. If the check fails, further access to the disk will not be permitted.

When the system has booted to the kernel, it can continue to receive attestations of the system integrity state. While our current prototype enforces integrity measurement with IMA, it is possible to use any mechanism, including runtime execution monitors such as Patagonix [19]. Our solution defines a framework for any monitoring system to be able to enforce access to the disk based on system state.

We have two choices for placing monitors in the system. The first is having a runtime monitor within the VMM itself, where it checks for violations of system integrity. The IMA approach that we currently use is an example of this. This approach is simple to implement but it has the disadvantage of being potentially vulnerable to attack if the VMM was ever to be compromised. The other location for a monitor could be within a separate privilege-separated virtual machine that is highly restricted by the VMM policy in terms of its read and write access. A thin OS could be running in this VM, in a manner similar to the management VM in Terra.

4.4 Error Handling

Fundamentally, our approach of measuring data before releasing access allowing access, while necessary for a secure boot mechanism, makes recovering from errors problematic. AEGIS addressed this issue by specifying that upon a failure, either a secondary ROM (which mirrors the address space of all internal components) is consulted, or a recovery kernel from ROM kept within the AEGIS board is booted from, which contacts a “trusted” host over a network. Establishing this procedure for the disk presents many challenges. Disks and file systems

provide a number of solutions to this issue. Journaling file systems such as ext3 and JFS provide a means for recovering updates which may have resided in the operating system buffer cache at the time of access to storage was revoked. Similarly, in the event of a compromise, versioning file systems [26] and storage devices [37] provide a means for rolling back changes to a pre-compromise state. If disk access must be maintained, copy on write schemes [24, 36] may be used instead of simply denying write access altogether. This will preserve the integrity of the data in storage while preventing a host compromise from denying disk access to critical applications.

Random environmental errors are well-protected by robust error correcting codes such as Reed Solomon preventing bit errors on hard disks. Concern about larger-scale individual disk failures can be mitigated by using RAID storage systems. In a RAID configuration verification logic may be performed by the RAID controller, and the root partition may be striped amongst multiple disks but present a single interface to outside the disk.

Administrative access may be appropriate under a failure condition, particularly one that results from a monitoring or attestation error. A log of events that is accessible to the administrator can provide information on the nature of the error that occurred, and can be further investigated to determine whether it was the result of natural phenomena, component failure, or malicious tampering.

5 Prototype Implementation

The Firma prototype consists of a host side on which the secure VMM and VMs run and the storage root of trust that enforces secure boot via access control, and provides continuous enforcement. We built the storage root of trust using the Linksys NSLU2 [18] network attached storage device, which was running the OpenSlug Linux distribution. All communication between the host and storage occurs via the ATA over Ethernet (AoE) protocol, which extends the ATA disk command set to storage devices sitting on local area networks. AoE is implemented on the storage side using `vblade`, a daemon process that exports one or more block devices. On the host side, the Linux AoE driver is used to mount the storage root of trust as a block device. The boot process is shown in Figure 2. We now detail how the prototype provides a secure boot, continuous enforcement.

5.1 Secure Boot

To provide the necessary mechanism for secure boot, the storage root of trust needs two things, a means of authenticating the host and a criteria for trusted host configurations. For management purposes, a secure channel to the

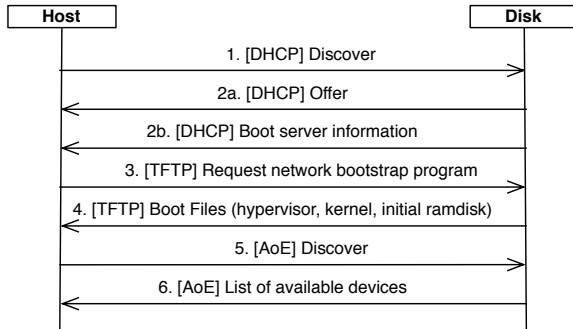


Figure 2: The boot process we implemented for our prototype. The slug requires a network boot to start from disk, and the initial phases allow the retrieval of the kernel, hypervisor, and initial ramdisk (`initrd`) images. Once the boot process begins, the list of AoE devices, including the root filesystem for the platform, becomes visible.

disk is also needed to deliver these to the disk. For this purpose, we use a USB token which carries the EKs associated with all TPMs on hosts that might access the drive and the set of approved binary measurements for software that may run on those drives. The EKs serve to authenticate hosts and verify that they are using authentic TPMs. The measurement lists may be used both for verifying the integrity of the trusted VMM at boot time and of applications loaded in VMs during continuous monitoring.

As several stages of the secure boot sequence require binary measurements to be delivered to the disk, we made a simple extension to the AoE protocol to allow for binary strings to be delivered to the disk either as part of another block request or as an independent message.

The secure boot process is carried out by the prototype as follows. When the system begins booting, the BIOS is hashed and this hash is extended into a PCR on the TPM. The BIOS then measures and loads the next stage which in our prototype is the `PXELINUX` bootloader (part of the `SYSLINUX` project)². The PXE bootloader loads each binary needed by the system, including the hypervisor, kernel, and initial RAM disk image. These are measured and the hashes extended into a PCR on the TPM. When the system components are being loaded, AoE is used to communicate with the disk, which sends the signed measurement list in its current form to the disk. If at any stage in the boot process, incorrect binaries are detected, the `vblade` instance is stopped, thus preventing access to the disk by malicious binaries.

²The use of `PXELINUX` is needed only by our implementation, in an actual implementation a traditional disk based bootloader like `GRUB` or `LILO` would be used.

5.2 Continuous Enforcement

The above described disk protocol extensions may be used to extend the secure boot mechanism in storage to a continuous monitor that allows re-attestations of up to date system state by the VMM. It is then the job of the disk to verify these measurements against its own integrity criteria as delivered by the physical USB token. The AoE driver on the host take a parameter specifying the number of seconds between re-attestations. It then sends an up to date set of measurements from the host TPM to the disk at the specified time interval.

All modifications needed for re-attestations were made to the linux AoE driver. TPM measurements are cached in the Xen dom0 kernel to alleviate the overhead of obtaining a TPM quote on the host. If time-based re-attestations are made, we use a kernel timer to schedule the construction and transmission of the attestation command.

In the event that the disk receives an untrusted measurement either during secure boot or continuous monitoring, a mechanism is needed to deny the hosts access to storage and, if desired, provide a secure environment from which the system may be recovered. All attestations arriving at the storage root of trust are checked against the disk’s integrity criteria by our utility, `VerifyQuote`. This utility reads the public key matched to the private key used to sign the TPM quote, along with the TPM quote itself. `VerifyQuote` then validates the signed quote using standard RSA signature validation. If the verification process fails, the instance of `vblade` exporting the disk’s root partition is killed, preventing any further access.

If a trusted environment is needed for recovery of either the system or data, a failed attestation may also lead to the launching of a new `vblade` instance, which exports a disk partition containing a minimal trusted environment that may be used to inspect the contents of storage. We also modified `vblade` to maintain an audit log of failed attestations, which could be potentially useful in determining when the system entered an untrusted state.

6 Evaluation

In this section, we evaluate the performance of Firma as described in preceding sections. We examine performance of the host and of the disk during the various operations each performs under the Firma architecture. All experiments were performed on a Dell Optiplex 745 with a dual-core Intel Core 2 Duo 6300 at 1.86GHz, 4GB of RAM, and an internal 160GB SATA drive. The prototype also includes a Linksys NSLU2 (“Slug”) connected to the host over a 100Mbps Ethernet interface. Attached to the Slug is a 160GB Seagate FreeAgent Go external USB hard disk. Also attached to the Slug was the token, a 2GB Lexar Firefly USB drive. The base operating sys-

Operation	Time
<i>Host System</i>	
Measure boot binaries	24.9388 (24.9113, 24.9662)
TPM Extend	39.9934 (39.9887, 39.998)
TPM Quote	880.037 (880.03, 880.045)
<i>Disk Firmware</i>	
Verify quote	70.0msec.

Table 1: Microbenchmarks of the host operations needed to support Firma. For each measurement, 95% confidence intervals are shown in parentheses. All numbers are shown in milliseconds unless otherwise noted.

tem was Debian 5.0 (“Lenny”) running a modified 2.6.24 Linux kernel and a stock Xen 3.2.1 hypervisor. Table 1 shows the time for each operation in the Firma architecture.

When the host system begins the boot process, it begins by loading a bootloader which in the prototype is the PXE bootloader. In order to gain access to the next stage boot files like the hypervisor, kernel, and initial RAM disk, the host sends a TPM quote to the disk. This TPM quote takes an average of 880ms to generate on the host, which is not atypical for current TPMs. This quote is then transmitted to the firmware, where it is validated. This validation takes an average of 70 milliseconds to complete. After the quote from the host is validated, the boot files are loaded. Before each file is loaded, it is measured by the bootloader, and these measurements extended into the TPM. The TPM extend operation takes an average of 40 milliseconds to complete. This extend operation will happen three times during the boot process, once each for the kernel, initial RAM disk image, and the hypervisor. The overall additional boot delay is $t_{over} = 880ms + 70ms + (40ms * 3) = 1070ms$, or just over 1 second, most of which is attributed to the TPM quote operation. This additional overhead is not unreasonable even on systems with sub-20 second boot times. This can be further reduced by prefetching the boot files while the quote generation and validation is occurring, thus eliminating the additional overhead.

During normal system boot, the disk is accessed in order to load the boot time binaries mentioned above. A standard SATA disk in our setup has a measured throughput of 70MB/s. A standard bootloader, in this case GRUB, has a total size of about 130K, while the boot files loaded initially have a total size of about 8MB. The hypervisor is 370KB, the kernel is 1.7MB and the initial RAM disk image is 6.2MB. All of the boot files are compressed to increase transfer speed. In addition to these boot files, the system must read several other binaries and their configuration files from the disk during the boot process. For

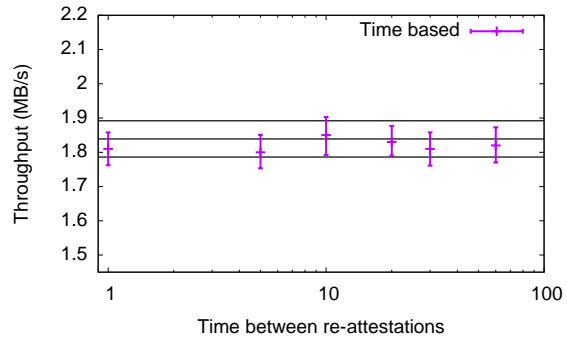


Figure 3: Throughput of Firma prototype when using time-based reattestations

a stock Debian Xen system, these transfers total 46MB with a boot time of 93 seconds. This is higher than some other hypervisor systems, such as VMware ESXi which has a disk footprint of only 32MB. This footprint includes everything needed to boot the system. In our prototype system, measuring the entire VMware ESXi disk footprint takes an average of 9 seconds.

In order to understand the impact of Firma’s continuous enforcement on I/O intensive workloads, we benchmark our prototype under a range of re-attestation time intervals. Figure 3 shows the throughput of our Firma prototype as the frequency of re-attestation is varied from 1 second to 60 seconds. The lower bound for time frequency based attestations is limited by the TPM, which can offer a new quote every 880 ms in our system. The horizontal lines in Figure 3 show the mean throughput of a base Firma system (i.e. without re-attestation) as well as the 95% confidence intervals. In all cases, the system that is re-attesting shows little to no throughput degradation. The base Firma system showed an average throughput of 1.839MB/s. The system with re-attestation showed at most a 2% overhead, well within range of experimental error.

7 Discussion

7.1 Dynamic Root of Trust for Measurement

Our design has centered around the use what the TCG refers to as a Static Root of Trust for Measurement (SRTM). The SRTM refers to the chain of trust rooted in the hardware (the motherboard’s CRTM), which measures each phase of the boot process up to the OS. Since the measurement gathering code is rooted in hardware, the correctness of subsequent measurements not based in hardware are subject to the integrity of this SRTM. Due to the variability of BIOS code and underlying hardware

on the average compute, no single well-known good measurement value can be known by a Firma disk a priori to pairing. As a result, the disk must gather measurements of this early part of the boot process when pairing.

This has several drawbacks. When the owner of the host wants to make changes to the host's BIOS or changes hardware components, the disk must be re-paired to obtain the new set of early boot measurements. Another issue is the initial trustworthiness of the BIOS and hardware. While we assume the greenfield state of the host during pairing to be trusted, the potential for undetected errors or exploits still exists.

To obviate these issues, a *dynamic* root of trust for measurement (DRTM) can be used. A DRTM is instead rooted in software and does not depend on the system's underlying hardware configuration. The Open Secure Loader (OSLO) [15] is a bootloader that implements such a DRTM. For AMD processors supporting the Secure Virtual Machine (SVM) extensions, and in particular the instruction SKINIT (which disables direct memory access to physical memory within a secure loader block), OSLO can use SKINIT to measure and execute a bootloader after clearing the processor pipeline and memory. This effectively allows the system to start from a clean slate without depending on the code loaded previously. Additionally, the DRTM bootloader measurement is stored in a PCR specifically designed to be used only via the SKINIT command.

What this means for us is that a Firma disk can pair with a host without having to gather SRTM measurements such as the BIOS. As a result, we can pre-compute the correct measurement lists directly on the disk, as we can hash them to determine what their values should be. Additionally, we can avoid making the assumption that the host is not running malicious firmware during pairing.

7.2 Beyond Unique Pairing

One advantage the Firma approach has over an on host secure boot mechanism is that a Firma disk can be used to secure boot on multiple hosts. If implemented on a portable disk, a Firma disk could be modified to pair with multiple host machines. Then when a user later plugs the disk into a previously pairing host machine, Firma could use the host specific information obtained from the pairing to expose the associated AIK to the host and to verify its attestations.

7.3 Lazy Attestation

Firma incurs performance overheads both at the host while generating binary measurements and at the disk while verifying measurements. These overheads may be addressed using lazy attestation and verification. In lazy

attestation, a trusted VMM may cache integrity measurements until a TPM extend operation. For example, in our implementation, measurements were cached in a Xen dom0 kernel which delivered the attestations to the disk at intervals. Using lazy evaluation a disk may hide the performance cost of integrity verification behind disk seek times. This is done by the disk processor which first initiates a disk seek and then performs integrity verifications for any attestation commands from the host. As disk response times dominate the disk processor time needed to check a quote, the overhead of integrity verification is nullified.

8 Related Work

Augmenting the capabilities of storage systems to provide security has been a topic of sustained interest over the past decade. Initial research into this area included the investigation of network-attached secure disks [10,11], an infrastructure where metadata servers issue capabilities that are processed by disks augmented with additional processing abilities to use capabilities as the basis for access control, requiring trust in servers external to the disk. Further research in this vein included self-securing storage [35], which, along with the NASD work, considered an object-based storage paradigm rather than the block-based approach that we use. Pennington et al. [23] considered the disk as an enforcement point for intrusion detection, requiring semantically-aware disks [32] for deployment at the disk level. Sundararaman et al. [37] also considered disk-level policy to provide secure, selective versioning of data, requiring additional capabilities from the disk in the form of type-safe disks [31] capable of distinguishing between inodes and data. These solutions require tighter coupling between the disk and the filesystem; in addition, they rely on the operating system to work in tandem with the disk.

BitLocker [20] is a disk volume encryption feature built into Windows Vista and Windows Server 2008 that optionally leverages the TPM to protect a volume encryption key (VEK). Bitlocker functions by placing the Windows system volume in an encrypted partition and using the TPM to `seal` the VEK to a known integrity state. The TPM `seal` operation encrypts the VEK along with a the set of PCRs. When unsealing the VEK, the TPM's current PCR set must match the PCRs specified with the key or the TPM refuses to decrypt the key, which effectively denies access to the system volume. The target set of good PCRs are selected by measuring the early boot phase of the Windows system before encrypting the disk. When booting, the TPM measures firmware, the boot loader, and other early boot phase files. Modifications of these files will be detected when the VEK is attempted to be decrypted.

One issue with BitLocker is its inability to protect the disk partition after releasing the VEK to the OS. Since bitlocker is designed to facilitate just secure boot of the OS, later compromise of the OS system will go undetected. In Firma, the disk is able to detect if malicious code is loaded by the VMM and deny further access to the disk. Another concern with BitLocker is the management of the VEK during firmware updates. In BitLocker, the VEK is placed unencrypted on the disk while an administrator performs changes that would effect the early boot measurements. In Firma, the disk can remain protected while an administrator updates the firmware. When the updated system is ready, the management token can be inserted into the disk to gather the new boot measurements.

Another example of enforcing on-disk policy is through write-once, read-many (WORM) mechanisms enforced through hard disks, commercially available through vendors such as EMC [8] and IBM [14]. These mechanisms have been shown to be vulnerable to insider attack [13]. Sion [30] has explored methods of ensuring policy enforcement at the storage level while considering the limitations of trusted hardware.

9 Conclusion

This paper has presented Firma, a novel storage architecture that demonstrated how to enable secure boot on commodity systems with minimal management overhead. Firma uses autonomously-enforced storage in concert with a TPM on the host and a trusted installation process to ensure a measureable chain of trust derivable from the running operating system instance to the installer. Unlike previous secure boot mechanisms, the disk can actively mediate access to data, disabling reads and writes if monitoring detects an integrity violation. We have shown the use of Firma to provide a trusted virtual machine monitor using storage as a root of trust, and showed that management of the system is minimal. The impact on system performance is minimal, with just over 1 second added to the boot process as a result of Firma's measurements.

There are opportunities to enhance the operation of Firma. Reducing the size of the OS associated with the VMM will reduce the amount of measurement required, and hence make boot times faster. We are also investigating the use of a dynamic root of trust to make the small number of management tasks even smaller. Finally, we are investigating implementation of the Firma prototype as emerging disks are produced that comply with the recently-defined enhanced disk interface specifications for secure communication.

Acknowledgements

We thank William Enck for his considerable help with idea generation and feedback. We also thank Machigar Ongtang for her input and Emily Rine for her help with editing. This work was supported in part by the NSF under award HECURA-0621429.

References

- [1] AMD. AMD64 Virtualization Technology: Secure Virtual Machine Architecture Reference Manual, May 2005.
- [2] ANSI. Information technology - AT Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS). INCITS T13/1699-D, Revision 3f, Dec. 2006.
- [3] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A Secure and Reliable Bootstrap Architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1997.
- [4] K. R. B. Butler, S. McLaughlin, and P. D. McDaniel. Rootkit-Resistant Disks. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, Alexandria, VA, USA, Oct. 2008.
- [5] L. S. Clair, J. Schiffman, T. Jaeger, and P. McDaniel. Establishing and Sustaining System Integrity via Root of Trust Installation. In *Proceedings of the 2007 Annual Computer Security Applications Conference*, Dec. 2007.
- [6] G. W. Dunlap, S. T. Ling, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, USA, Dec. 2002.
- [7] J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the ibm 4758 secure coprocessor. *Computer*, 34(10):57–66, 2001.
- [8] EMC. Centera Compliance Edition Plus. <http://www.emc.com/centera>, 2007.
- [9] C. Fruwirth. LUKS - Linux Unified Key Setup. <http://luks.endorphin.org>, 2008.
- [10] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. In *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, USA, Oct. 1998.
- [11] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, and D. Rochberg. A case for network-attached secure disks. Technical Report CMU-CS-96-142, Carnegie Mellon University, Pittsburgh, PA, USA, Sept. 1996.
- [12] T. C. Group. Stopping Rootkits at the Network Edge, January 2007.
- [13] W. W. Hsu and S. Ong. WORM storage is not enough. *IBM Systems Journal*, 46(2), Apr. 2007.
- [14] IBM. IBM TotalStorage Enterprise. <http://www-03.ibm.com/servers/storage>, 2007.
- [15] B. Kauer. OSLO: Improving the security of Trusted Computing. In *Proceedings of the 16th USENIX Security Symposium*, Boston, MA, USA, Aug. 2007.
- [16] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), Dec. 2005.

- [17] S. T. King, P. M. Chen, Y.-M. Wan, C. Verbowski, H. J. Wang, and J. R. Lorch. SubVirt: Implementing malware with virtual machines. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006.
- [18] Linksys. NSLU2 Product Information. http://www.linksys.com/servlet/Satellite?childpagename=US%2FLayout&packedargs=c%3DL_Product_C2%26cid%3D1118334819312&pagename=Linksys%2FCommon%2FVisitorWrapper, Apr. 2008.
- [19] L. Litty, H. A. Lagar-Cavilla, and D. Lie. Hypervisor Support for Identifying Covertly Executing Binaries. In *Proceedings of the 17th USENIX Security Symposium*, pages 243–258, San Jose, CA, Aug. 2008. USENIX Association.
- [20] Microsoft. BitLocker Drive Encryption Technical Overview. <http://technet.microsoft.com/en-us/library/cc732774.aspx>.
- [21] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, 10(3), Aug. 2006.
- [22] B. D. Payne, M. Carbone, M. Sharif, and W. Lee. Lares: An Architecture for Secure Active Monitoring Using Virtualization. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2008.
- [23] A. G. Pennington, J. D. Strunk, J. L. Griffin, C. A. N. Soules, G. R. Goodson, and G. R. Ganger. Storage-based Intrusion Detection: Watching storage activity for suspicious behavior. In *Proceedings of the 12th USENIX Security Symposium*, Washington, DC, USA, Aug. 2003.
- [24] Z. Peterson and R. Burns. Ext3Cow: A Time-Shifting File System for Regulatory Compliance. *ACM Transactions on Storage*, 1(2):190–212, May 2005.
- [25] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, Aug. 2004.
- [26] D. J. Santry, M. J. Feeley, N. C. Hutchinson, and A. C. Veitch. Elephant: The file system that never forgets. In *Workshop on Hot Topics in Operating Systems*, pages 2–7, 1999.
- [27] Seagate. Seagate Technology - Momentus 5400 FDE.2 Hard Drives, Sept. 2007. http://www.seagate.com/www/en-us/products/laptops/momentus/momentus_5400_fde.2/.
- [28] Seagate. Seagate Technology - Momentus 5400 PSD Hybrid Hard Drives, Sept. 2007. http://www.seagate.com/www/en-us/products/laptops/momentus/momentus_5400_psd_hybrid/.
- [29] Seagate Technology LLC. Self-Encrypting Hard Disk Drives in the Data Center. Technology Paper TP583.1-0711US, Nov. 2007.
- [30] R. Sion. Strong WORM. In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS 2008)*, Beijing, China, June 2008.
- [31] G. Sivathanu, S. Sundararaman, and E. Zadok. Type-Safe Disks. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*, Seattle, WA, USA, Nov. 2006.
- [32] M. Sivathanu, V. Prabhakarn, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Semantically-Smart Disk Systems. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, Apr. 2003.
- [33] S. W. Smith and S. Weingart. Building a High-Performance, Programmable Secure Coprocessor. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 31(9), Apr. 1999.
- [34] L. St. Clair, J. Schiffman, T. Jaeger, and P. McDaniel. Establishing and Sustaining System Integrity via Root of Trust Installation. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007)*, Miami Beach, FL, Dec. 2007.
- [35] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-Securing Storage: Protecting Data in Compromised Systems. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI'00)*, San Diego, CA, USA, Oct. 2000.
- [36] Sun Microsystems, Inc. Solaris ZFS file storage solution. *Solaris 10 Data Sheets*, 2004. www.sun.com/software/solaris/ds/zfs.jsp.
- [37] S. Sundararaman, G. Sivathanu, and E. Zadok. Selective Versioning in a Secure Disk System. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, USA, July 2008.
- [38] T10. Information technology - SCSI Primary Commands - 4 (SPC-4), Feb. 2009.
- [39] TCG. Trusted Computing Group: Trusted Platform Module (TPM) Specifications. <https://www.trustedcomputinggroup.org/specs/TPM/>.
- [40] TCG. *TPM Main: Part 1 - Design Principles*. Specification Version 1.2, Level 2 Revision 103. TCG, July 2007.
- [41] TCG. *TCG Storage Interface Interactions Specification*. Specification Version 1.0, Revision 1.0. Trusted Computing Group, Jan. 2009.
- [42] TCG. *TCG Storage Security Subsystem Class: Opal*. Specification Version 1.0, Revision 1.0. Trusted Computing Group, Jan. 2009.
- [43] TrueCrypt Foundation. Truecrypt. <http://www.truecrypt.org>, 2008.
- [44] T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch. The Zurich Trusted Information Channel – An Efficient Defence against Man-in-the-Middle and Malicious Software Attacks. In *Proceedings of 1st International Conference on Trusted Computing and Trust in Information Technologies (TRUST 2008)*, Villach, Austria, Mar. 2008.