SwitchBlade: Policy-Driven Disk Segmentation

Kevin Butler, Stephen McLaughlin, Thomas Moyer, 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

Abstract

As it becomes more common for multiple operating systems to run machines to run on a single machine, strict isolation mechanisms have become increasingly critical. While OS-level isolation protects against some attacks, all OS protections can be subverted by directly accessing a shared disk. In this paper, we introduce SwitchBlade, a disk protection model that confines operating systems stored on the same disk into *segments*. Users physically insert policy-carrying tokens into the drive to access specific segments. Without the capability to access an OS segment, it is fully isolated from the user. We implemented SwitchBlade on a real prototype. We describe our architecture and implementation, and comment on the user experience and tools we have developed to work with the system. We provide a security analysis showing SwitchBlade's resistance to attack and evaluate its performance on real systems – the disk can realistically boot in about 8 seconds with our current prototype. We thus can provide the isolation guarantees equivalent to physically separate systems without the enormous usability burdens such systems entail.

1 Introduction

The rise of networks and distributed systems has allowed for new and innovative computing models. However, any opportunity for different operating systems to interact with each other raises the possibility of attack, as has been demonstrated repeatedly from the earliest worms onward [43]. In many cases, the only method of ensuring isolation from attack is through *air gap security*, or physically isolating systems from each other, such as with red/black networks [52].

The increasingly complex manners in which users operate their systems complicate these goals. Users have the ability to run multiple operating systems, either one at a time or simultaneously, on their desktop machines. These *multiboot* and *virtualized* systems, respectively, provide great benefits for users who require access to multiple OSes (e.g., a web site designer ensuring consistent page rendering in different environments), without incurring the energy and space requirements of dedicating an entire machine to an OS. In these systems, the ability to *isolate* operating systems from each other effect is critical to preventing information leakage, and to mitigate the effects of a compromised OS from attacking other OSes on the system. Numerous approaches, including separation kernels [34] and virtual machines [22, 27, 55] have been employed to maintain isolation.

Regardless of how strong the isolation mechanisms between applications or operating systems may be, however, an opportunity exists to tamper with other OSes if they share a common storage medium. Mechanisms such as partitioning the disk are constructions enforced by operating systems, and an attacker that compromises the OS will be able to trivially subvert any protections offered by it. Additionally, booting from a live CD such as BackTrack [33] is often sufficient to gain a full view of on-disk contents, free from access control mechanisms enforced by the OS. Encrypted volume solutions such as TrueCrypt [50] and LUKS [13] are reliant on the OS kernel not being compromised; additionally, these programs themselves may be open to vulnerability [10]. Full disk encryption [37] provides no protections when the disk is unlocked for regular operation.

To address these problems, we propose a novel method of providing isolation. In this paper, we present SwitchBlade, a disk protection model that effectively confines operating systems from each other's effects. An OS is able to run only in its own *disk segment*, a defined set of blocks on the disk that are accessible to the OS as a physically separate disk.

A user is able to access segments by plugging a physical token into the drive. These tokens possess capabilities that define read and write access to one or more segments. Even if the adversary ran BackTrack, the only segments visible to them would be those made available through their policy, affording no opportunity to read, write, or interact in any way with other OSes on the disk. The user forms a *trusted path* to the disk by plugging a token directly into it, ensuring that isolation is enforced independently from the rest of the system.

Many modes of operation are possible with SwitchBlade, and we have explored two of these. In *multiboot* mode, the user will access a different view of the disk and thus, available operating systems, depending on which token is plugged into the disk. In *virtualized* mode, a virtual machine monitor (VMM) and guest operating systems are exposed to the user, where the set of available OSes is defined by the token policy. We thus can enforce isolation between sets of virtual machines that may be differentiated by security class.

The contributions of this paper are as follows:

- We define the SwitchBlade architecture for achieving storage level isolation, and show how it may be used for protecting the integrity of system components and the confidentiality of user data.
- We implement a fully functioning prototype to validate our concept and provide a detailed description of how each component in the architecture is implemented using commonly available hardware and software.
- We relate our experiences with configuring the disk policy and building systems that use SwitchBlade. This includes descriptions of token management, segment creation and the installation and use of a hypervisor and guest operating systems.

2 System Goals

SwitchBlade is designed to protect operating systems on disk by isolating access to only segments on the disk specified by capabilities retrieved from a token, physically inserted into the disk by the user. Below, we consider the goals of our system and the threats that it is and is not capable of addressing.

2.1 Design Criteria

Our approach is motivated by providing a protection mechanism that is simple to administer and easy to use, and that enforces a model of isolation ensuring that no operating system will be accessible to another OS specified to be isolated from it. We explain these goals in further detail:

Management Users insert a physical token into the disk that specifies policy through capabilities. As discussed in Section 7, a token manager that operates independently of the system with SwitchBlade installed is where policy is specified. From the standpoint of the OS or VMM, they have access only to the on-disk segments that are specified on the token, which provides mandatory enforcement. No further policy decisions need to be made by SwitchBlade.

Usability The user does not need to make any changes to the way they use the system beyond plugging the correct token into the disk, providing them access to the desired operating systems. No further modifications are necessary. SwitchBlade also operates transparently to the operating systems, which require little to no modification to be run.

Isolation SwitchBlade isolates disk segments from each other. Each segment is presented to its OS as a separate disk, each of which contains a master boot record (MBR). While MBR rootkits [12, 25] may infect an individual segment, they will be unable to spread beyond their segment to infect other parts of the disk that the token-based policy does not provide access to. A by-product of this isolation is a natural confidentiality and integrity based on the inability to read or write other segments unless authorized by the token. In addition, having the user physically insert a token into the

drive creates a trusted path to the disk that is independent of the potentially compromised OS. The problem of confining multiple segments on the disk reduces to one of physical security through token management, reducing the TCB for segment protection to the disk enclosure.

2.2 Threat Model

We consider an adversary capable of arbitrarily and completely subverting a host operating system. The OS itself is 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 [32] and rootkit-resistant disks [5].

Many security architectures that consider virtualized environments make the assumption that the hypervisor and associated administrative VM is within the trusted computing base (e.g., [7, 31]). 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 machine based rootkit (VMBR) such as SubVirt [23] 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 [30] and AMD SVM [1] processors makes the threat of VMBRs running under a VMM increasingly possible. We also assume no trusted path through the operating system without more trust being placed in the VMM, such as with the trusted VMM in Terra [14].

Our enforcement mechanism takes place within the disk. 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. Physical security measures such as those used in secure coprocessors [42] may be possible if the cost-benefit ratio makes this design point appropriate. In addition, 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 [54] are possible where such defenses are deemed necessary.

3 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 [15,16] 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 [44], which, along with the NASD work, considered an object-based storage paradigm, rather than the block-based approach that we use. Pennington et al. [32] considered the disk as an enforcement point for intrusion detection, requiring semantically-aware disks [40] for deployment at the disk level. Sundararaman et al. [47] 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 [39] capable of distinguishing between inodes and data. These solutions require tighter coupling between the disk and the filesystem and for the operating system to work in tandem with the disk.

We use a prototyping architecture similar to the one described in the work on rootkit-resistant disks [5]. This work differs considerably from that project, however, in that we are enforcing inter-OS isolation across a common storage media, with a more complex policy that can allow selective access to operating systems based on token capabilities. In addition, there is no block labeling required with our system, meaning effectively no performance overhead. Similarly, the Firma project [4] provides a secure boot mechanism for commodity disks, but does not address the issue of OS isolation. Our use of a physical token to provide capabilities to the disk is an effective method of providing a trusted path between the user and disk that circumvents reliance on any part of the system apart from the disk; Griffin et al. [17] considered a secure channel from an administrative console to the disk that would also be potentially usable.

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 [20]. These mechanisms have

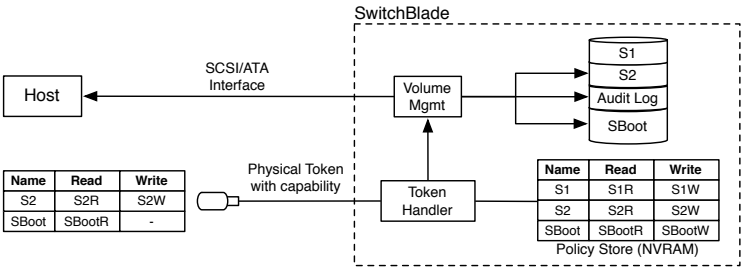

Figure 1: SwitchBlade architecture. A physical token contains the set of labels for segments to be exported, in this case, S2 and SBoot, the latter of which is exported as read only. The audit log is read only by default and writable by the disk controller. In this case, the Token Handler constitutes the policy decision point which configures volume management to export the segments allowed by the current physical token according to the policy, which is stored in NVRAM.

been shown to be vulnerable to insider attack [19]. Sion [38] has explored methods of ensuring policy enforcement at the storage level while considering the limitations of trusted hardware.

4 Architecture

SwitchBlade is a protection system consisting of a disk augmented with a processor for providing cryptographic services (such as those available in an FDE drive [37]) and policy evaluation, and non-volatile memory (as found in a hybrid hard disk [36]) for storing policy metadata. The goal of SwitchBlade is to provide storage level isolation for integrity, confidentiality and availability of storage resources. This is accomplished through the use of disk *segments*, which are analogous to the memory segments used in microprocessor architectures to provide hardware level memory protection. A disk segment appears to the host system as a single device on the storage bus, and is addressed by the host as if it is an actual disk. The host should not be able to tell the difference between a set of segments exported by a virtual disk and a chain of devices on a storage bus. The set of disk segments accessible by the host is determined by the capability on the current token in the disk, allowing the set of devices currently available on the bus by the host to change dynamically through the use of different tokens. The SwitchBlade architecture is shown in Figure 1. We now examine in detail how policy is expressed, how SwitchBlade is managed, and how it is used during system boot.

4.1 SwitchBlade Policy

The SwitchBlade policy has two main objectives. The first is to define the controlled access to disk segments (hereafter referred to solely as segments). This includes the label state of all segments on the disk as well as the semantics of capabilities stored on the physical tokens. The second is to control access to the various administrative functions needed for disk management, including segment creation and deletion. In the SwitchBlade policy, segments represent the objects in the system, and the set of sensitive operations includes the creation, deletion, reading and writing of segments. Subjects are authenticated by the current token in the disk. Each segment has several attributes:

- **name** - The unique identifying name for this disk. In the case of a SCSI or ATA bus, this could be the device identifier of the disk.
- **size** - The amount of disk space allocated for this segment
- **labels** - The set of labels defining read and write access to the segment, as well as the ability to delete the segment.

The attributes for each segment are stored in the disk's non-volatile memory to allow for concurrent access to the policy metadata and data on the disk. Access control decisions are made for each segment based on the segment's

```
Capabilities:
{D1:r,w}
{D2:r,w,d}
{DBoot:r}
{can_create}

Commands:
[delete_disk:D1
 create_disk:D3; 10000; r,w,d]
```

Figure 2: An example of the contents of a SwitchBlade token.

attributes, and the set of capabilities on the current token in the disk. Along with the specific labels needed for each segment, there is a global capability needed to create new segments. A token with this capability may issue one or more `create_disk` commands, which result in the creation of new segments (covered in Section 4.2).

Each physical token contains capabilities, where each capability maps a segment name to a set of labels specifying the operations that may be performed on that segment. Also present in a capability is the optional label to allow for the creation of new segments from the available free disk space. Along with these capabilities is a queue of `create_disk` and `delete_disk` commands to be executed. These details are described in the next section.

An example of token contents is shown in Figure 2. When this token is in the disk, segments `D1` and `D2` can both be read and written, segment `D2` can be deleted, and the boot segment is read-only. Upon insertion into the disk, the command set from the token will be parsed and executed. In this case, `D1` will be deleted and a new segment, `D3`, will be created with 10GB of reserved space and the set of labels `r,w,d` (capabilities for segment read, write, and delete).

Because no assumptions are made about the available disk space or the permissions stored in the capability before `create_disk` or `delete_disk` commands are issued, they may fail. An administrator would like to know about the success or failure of each command, and in the case of a failure, the cause. For this reason, the disk includes a tamper-proof audit log. This log is exported to the host as read-only as long as the disk is powered on. Only the disk controller may append messages to the audit log segment. Common interfaces to the log include the OS file system on the host and the segment menu as described in Section 5.2.2.

The enforcement mechanism for the SwitchBlade policy is a part of the disk's firmware and runs in the disk processor alongside the normal processing code. The enforcement mechanism is completely independent of the rest of the disk controller code, and is for all purposes verifiable. It consists of a single hook that intercepts all I/O requests at the disk interface and inspects their command fields to see if they are requesting a sensitive operation. If so, the current set of labels available for the specified segment on the current token are checked to make sure the request can be fulfilled. If so, the command is passed unmodified to the regular controller code. The independence of the enforcement mechanism and the disk controller code allows for a performance optimization: a pipeline of disk requests may be formed, allowing current requests to be authorized while previously authorized requests are fulfilled.

4.2 SwitchBlade Management

Up to this point, we discussed disk segments in terms of their attributes and the policies controlling their access and manipulation. We now examine disk segments in terms of their contents and the administrative procedures for managing them. We also describe token management and how SwitchBlade is used by the host system.

The key idiom that we are assuming for general use of SwitchBlade is that each segment contains at most one operating system (although, as we show in Section 5.2.2, it is possible to run more than one OS inside a segment). This may be a guest OS in a VM environment, or a standalone OS available as part of a multiboot configuration. Each OS will view its segment as a completely separate disk, and will keep both its file system and swap partition on this segment. A policy may also be defined for segments that only contain data, which can act as shared storage between multiple running OSes. These segments will most often be readable and writable.

In the case of a multiboot system, the usage of a SwitchBlade is simple - isolation is maintained, since the OS has no knowledge that other storage even exists on the disk. In the case of virtualized operation, there is an opportunity for more powerful isolation mechanisms between running VMs by trusting the VMM to multiplex access to virtual disks over VMs. In this case, the VMM is aware of each segment exported by the disk, and allows each guest OS access only to the segment from which its image was retrieved, using storage virtualization methods discussed further in Section 5.2.2.

We believe that the VMM can provide some weak guarantees for several reasons. First, as we discuss in the next section, SwitchBlade possesses the necessary mechanisms to protect the integrity of the VMM stored in the boot segment, ensuring that the VMM is always started in a good state. Second, while it has been shown that a malicious processes or OS can detect that it is running in a VM [11], VMMs have yet to be compromised, although it may be only a matter of time until this occurs.[1] Finally and most importantly, in the event that the VMM is compromised, the malicious VM is limited to affecting the integrity and confidentiality of the files in segments exposed by the current token in the disk.

Tokens used with SwitchBlade are created and modified by a trusted *token manager*, discussed further in Section 7. The token manager is used to construct capabilities and instruction queues and push them to tokens.

4.3 Boot Process

The process of booting from a SwitchBlade is similar to that of booting from a regular disk. The equivalent of a disk's MBR in SwitchBlade is the boot segment, which has several additional features beyond those found in a typical MBR. First, because segments may be of arbitrary size, the boot segment contains a stage-1 bootloader with enhanced functionality, as it needs to inspect the disk to discover which segments contain OSes. Additionally, the boot segment is almost always kept in a read-only state to protect the integrity of the code involved in the boot process, thus ensuring that the system can be booted into a safe state.

Exporting the boot segment allows SwitchBlade to provide some guarantees of integrity. Namely, the disk can guarantee the integrity of the contents of the boot segment. Once executed on the host, the boot segment can further verify the integrity of higher levels of software. This is achievable by maintaining a *measurement* of a good boot system on the token, and comparing this value to a measurement of the boot segment. In the case that the measurement of the boot segment does not match the value stored on the token, SwitchBlade will refuse to satisfy requests for blocks in the boot segment, and will append a message to the above described tamper-proof audit log.

There are two times at which the boot segment should be measured: when the disk is powered on, and when it has previously been exported as writable. In the former case, the boot segment contents may have been tampered with, and in the latter case, they may have been altered by a compromised host OS. In the case of a regular upgrade of the software on the boot segment, which we assume is relatively rare compared to OS upgrades and more akin to a firmware update, the measurement stored on the token will need to be updated to match that of the new boot segment.

Note that this approach provides guarantees about the state of the boot segment but as per our threat model, we do not guarantee against physical tampering of the disk's hardware and as such, we do not provide attestation of it. This may be possible if a device such as a trusted platform module [51] is installed in the disk for attestation to the host system or an active token; alternately, a secure boot that measures its own integrity in a manner similar to AEGIS [2] is possible. While we defer further discussion of this issue, it is notable that if we consider the physical componentry of the disk to be trusted, the disk acts as a *root of trust*.

4.4 System Management with SwitchBlade

We now describe system management techniques for multiboot and virtualized operation, and examine how they may be implemented using SwitchBlade and the policies, disk management mechanisms and boot process described above. In the case of secure multiboot, a workstation has several operating systems that may not access the data on each other's segments. The bootloader resides in the boot segment, from where the BIOS retrieves it for execution. The bootloader

[1] Note that while a technique such as SubVirt installs itself below an operating system, it is itself a VMM and there are currently no effective techniques to dislodge it.

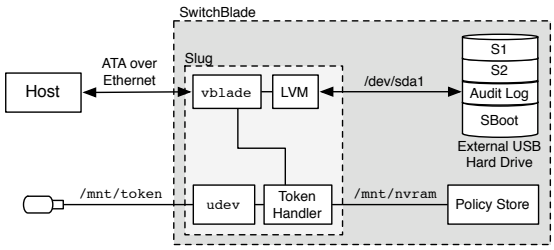

Figure 3: Details of the SwitchBlade prototype. The token handler parses the contents of the USB token, mounted by `udev`. It then invokes `vblade` to export the logical volumes allowed by the token. The audit log is also managed as a logical volume that is exported read-only by `vblade`, but directly writable by the token handler.

then probes for available segments and loads the OS from the OS segment. A policy may be specified to allow a shared segment to be accessed by multiple operating systems. In this case, the token will export two segments, one containing the OS that will be run, and one containing the shared data. Once the OS segments are loaded, the shared data segment may be mounted, as well as the segment from which it was loaded as a regular block device by its name.

In the virtualized mode of operation, we wish to have multiple OSes running concurrently, each with access to their own segment and no other, thus preventing any information leakage between OSes through storage.[2] In this case, a VMM resides in the boot sector that is trusted to multiplex access to the multiple segments exported by the token. The VMM uses a policy that a guest OS can only access the segment from which it was loaded. In order to protect the integrity of the VMM image, the token should only contain the read capability for the boot segment. In both of the above-described cases, the `create_disk` and `delete_disk` capabilities will not be needed as a part of normal system management.

5 Prototype Implementation

We describe in this section our implementation of the SwitchBlade prototype, including the command protocol, the token interface, and segments. We then discuss the deployment of SwitchBlade in virtualized mode, using Xen as a VMM.

5.1 SwitchBlade implementation

We implemented a prototype SwitchBlade with the capacity for creating, deleting and exporting segments based on the capabilities present on the physical token. We then used the prototype to build the two types of systems described in Section 4.4, secure multiboot and virtualized operation. Because of the prohibitive difficulty of obtaining and modifying the firmware of a commodity disk, we implemented our prototype using a modified network attached storage device. The main difference between the prototype's interface and that of a commodity disk drive is the use of Ethernet for the physical and MAC layers between the disk and host, as opposed to an ATA or SCSI bus.

In order to minimize differences between our prototype's command interface and that of a real disk, we use the ATA over Ethernet (AoE) [6] command level protocol between the disk and host. As the name implies, AoE sends ATA commands directly over the Ethernet interface between a client and host. Unlike iSCSI [35], AoE does not use higher level network or transport layer protocols, reducing performance overhead in the network stack normally incurred by protocols such as iSCSI; while this does not allow commands to propagate beyond an Ethernet segment, our assumption is that the disk will be co-located with the firmware in a SwitchBlade system, making this access mode more appropriate. In a typical AoE installation, a host machine, or *initiator*, issues ATA commands to a *target* storage device. In the case of the prototype SwitchBlade, the initiator consists of the host machine using one or more `aoe` block devices, which issue

[2] We do not claim to provide protection against copy-paste or other forms of information leakage above the storage layer; solutions such as labeled desktops in Trusted Solaris [46] may provide for this level of mediation.

commands to AoE targets on the LAN. The targets consist of the modified NAS devices running `vblade`[3], a server program that listens for ATA commands via raw sockets. Individual AoE devices are addressed by a major and minor number used by the target to demultiplex commands to each device.

The SwitchBlade prototype as shown in Figure 3, consists of two discrete physical components, an external USB hard disk which provides the actual storage and a NAS device which sits between the host and disk that acts as the policy store and enforcement mechanism. For the latter, we use a Linksys NSLU2 storage link [26], commonly known as a "slug," which is used in academic and industrial research venues for approximating the processing capabilities of modern disks. In order to program the slug to act as a policy enforcement point, we replaced its default firmware with the SlugOS Linux distribution [41], allowing us to leverage a large collection of existing software, as well as write our own. Token handling in the slug is performed using the `udev` [24] device management framework for the Linux 2.6 kernel, which detects when a token is inserted or removed. Upon token insertion, `udev` invokes a token handler program we wrote that extracts the capability and commands from the token. A new instance of `vblade` is started for each segment specified in the capability.

The basic unit of storage over which access control is performed in our prototype is the segment. Segments are implemented in our prototype using the Linux logical volume manager [48]. The segments are specified as logical volumes (LVs) over the single physical volume (PV) that is the external USB storage device. We then use a single instance of `vblade` to export each LV to initiators. If multiple segments are to be exported under a given token, we launch as many instances of `vblade` as there are segments to be exported. Each segment is addressed by the host using its unique major and minor number as exported by AoE, and is visible to processes on the host as a block device containing the major and minor number of the segment, e.g. `/dev/e1.1`, `/dev/e1.2` etc.

5.1.1 Audit log

We were able to easily use the implementation of segments to create the tamper-proof audit log. An instance of `vblade` is started to export the audit log read-only at disk power-on. When the token handler executes any commands or detects boot segment tampering, it appends detailed messages to the audit log by writing directly to the LV. Any guest OS may obtain the audit log contents by mounting the AoE device with minor number 0 and reading that location, e.g., `/mnt/audit_log`. Such functionality can be used as the basis for the implementation of an S4 disk [44] which requires the existence of an append-only audit log provided by the disk. We have not investigated other requirements for S4, such as journaled metadata and log-structured writes, as those are orthogonal to our goals with SwitchBlade.

5.1.2 Policy management

For each segment in the prototype SwitchBlade, an entry is stored in the disk's NVRAM, containing the segment name, read and write labels and the minor number used to identify that segment when exported. When a token is inserted into the disk, the capability is extracted, and from it are parsed a set of segment names and the corresponding read and write labels and minor numbers. The pair of labels for each name is compared against that stored in NVRAM and used to determine whether the segment is exported and if it is writable. In the case where only a read label is present, the segment is exported as read-only, and in the case where only a write label or no labels are present, the segment is not exported. An instance of `vblade` is pointed at the corresponding LV for each segment to be exported, each of which is flagged as read-only. At this point, the initiator will detect the new block device.

5.1.3 Boot sector integrity requirements

SwitchBlade verifies the integrity of the boot sector against a measurement stored on the token. As discussed in Section 4, the boot segment must be measured after the disk is powered on, and after the boot segment has been exported as writable

[3]The genesis of the name "SwitchBlade" is because of `vblade`, which exports virtual storage blades to the user. We provide the ability to "switch" between these blades while keeping them isolated from each other.

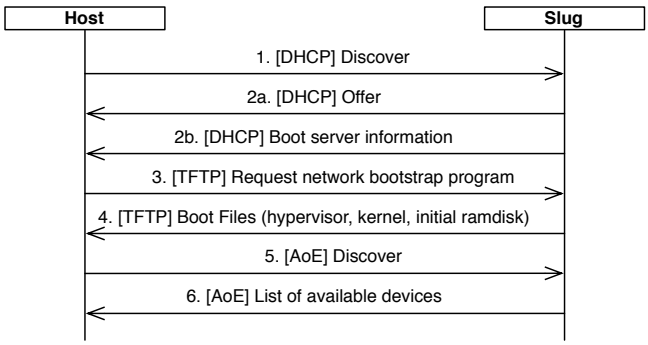

Figure 4: The process of booting the host system from the Slug disk. The initial phases are the DHCP and TFTP protocols in order to retrieve the kernel, hypervisor and initial ramdisk (`initrd`) images. Once the host has started booting the system, it discovers the available AoE devices, which will include the root file system for the platform.

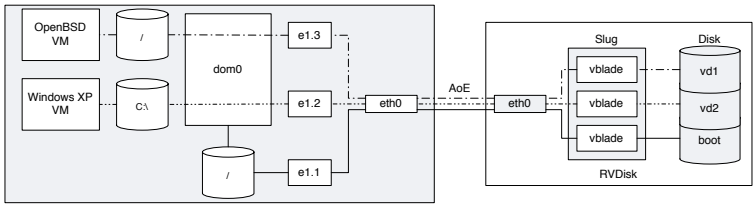

Figure 5: Operation in a virtualized environment. Each ATA-over-Ethernet (AoE) device is a separate block device that either dom0 or the domUs can access. The domUs are unaware that the block device runs AoE.

as the disk is vulnerable to tampering in either circumstance. To ensure that measurements are always done at these times, the prototype uses a UNIX temporary file that is cleared either by `tmpfs` at disk power down, or by the token handler when the boot segment is exported as writable. Any time the file is not present, the SHA-1 hash of the boot segment is computed with OpenSSL [49] and compared against the hash stored on the slug. If the contents of the boot segment have been intentionally altered by an administrator, e.g. for a bootloader upgrade, it is up to the administrator to ensure that the proper new measurement is stored on the token.

5.2 Client System

In this section, we explain how SwitchBlade is implemented to operate in either multiboot or virtualized mode. We begin with the multiboot scenario and then cover the installation procedure for the virtualized operation.

5.2.1 Multiboot Operation

In order to provide secure multiboot, we use the Preboot Execution Environment (PXE) [21] network-based boot, supported by many common network cards. The slug acts as a server that provides the kernel image for the host to boot from. In order to boot using the AoE targets made available by the slug, some modifications to the OS are typically needed. We experimented with the Ubuntu and Debian Linux distributions, and modified the `initrd` image to load the AoE driver and mount the AoE segment for the root file system. For Windows XP, the additional steps involved installing the AoE driver, and then using the open-source `gPXE` software [9] to boot over AoE. The slug's PXE boot process is illustrated in Figure 4. When the system is first powered on, it sends a broadcast request looking for a DHCP server, which sends a response indicating the location of the TFTP server. In this case, both servers are on the slug. The host then sends a request to the TFTP server for the network bootstrap program (NBP), responsible for retrieving the files necessary to boot. For multiboot operation, these are the kernel and `initrd` images, or some next stage bootloader.

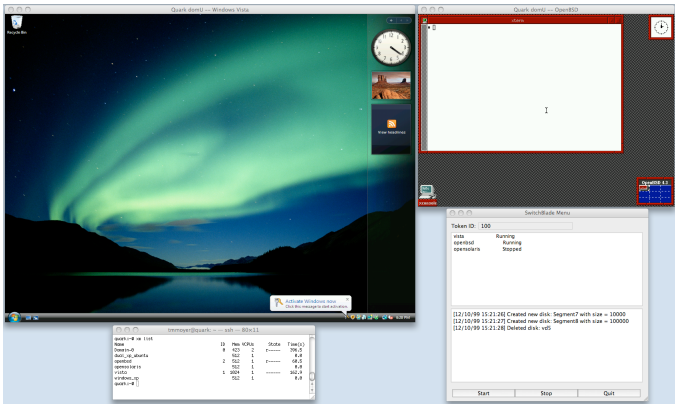

Figure 6: The above screen capture depicts Windows Vista and OpenBSD 4.3 running in virtual machines. The terminal window is showing the VMs that Xen is configured to run, which in this case is five different VMs. The menu interface is shown in the lower right. This interface only shows VMs that can be correctly started based on the available devices presented to dom0, which includes the two running VMs and another VM containing OpenSolaris currently in a stopped state.

For virtualized operation, the hypervisor image is also sent to the host. The host begins its boot using these images, and once the boot has progressed far enough to load the AoE drivers, the root file system can be mounted and the boot can proceed as normal.

Note that the need for PXE boot and OS boot modifications is necessary only because we deployed the SwitchBlade logic on the slug. On a production device, the kernel image would be booted directly off the boot segment over an ATA or SCSI interface and the need for AoE would be obviated.

5.2.2 Virtualized Operation

Figure 5 provides an overview of how SwitchBlade operates in virtualized mode. We use the Xen [3] VMM to support running virtual machines. The host system contains a processor with AMD virtualization extensions, allowing Xen to run unmodified guest operating systems. The initial setup involves installing the hypervisor and administrative domain (collectively called dom0), which is achieved in two phases. The first phase is installation of a base Linux environment on the root segment (using a Debian installer) and manually loading the AoE driver, while modifying `initrd` to boot an AoE root file system. The second phase involves copying the kernel and `initrd` images to the slug in order to allow PXE boot.

Once the system has been rebooted, Xen is installed and the kernel and `initrd` are again copied to the slug along with the hypervisor. A final reboot allows the slug to serve the VMM image, and creation of guest domains (domUs) can occur at this point. We installed four different guest OSes on SwitchBlade, including Windows XP, Windows Vista, OpenBSD 4.3, and OpenSolaris. To demonstrate that each segment considers itself to be a physical disk, we created a fifth segment and installed a dual-boot configuration of Windows XP and Ubuntu Linux 8.04.1 inside of it.

To provide a usable interface to the user, we have developed a menu interface that shows the user which virtual machines are currently able to boot given the inserted token. The user is also presented with an audit log that shows token activity from the disk. The interface is shown in Figure 6. This figure contains four different windows. The top two windows are VNC connections to the VMs that are running, namely Windows Vista and OpenBSD 4.3. The lower-left window is a terminal showing the VMs that the hypervisor is currently configured to run. The list includes five different VMs plus a listing for dom0. The window in the lower right is the user interface which only shows VMs that are able to be started based on the token that is inserted. Currently the window shows three available VMs, two of which are running and the third being stopped. The menu interface also shows the audit log and current token ID.

5.3 Experience

To illustrate the typical use of SwitchBlade in a production environment, we describe our experiences performing several common administrative tasks. We cover regular operation, which consists of segment and token management and segment creation. We also describe the response of different OSes to unexpected token removal during regular operation, when given no warning from the SwitchBlade and when gracefully shutdown by the hypervisor.

Each physical token contains a capability and a list of one or more create and delete disk commands.

```
create_disk:boot:1000:boot_r:boot_l:1
create_disk:vd1:10000:vd_r:vd_l:2
```

Instructing the disk to create a 1GB segment called "boot" with the specified read and write labels, and minor number 1 and a 10GB segment called "vd1" with read and write labels `vd_r` and `vd_w` and minor number 2. Upon inserting the token, the disk removes the commands from the token, creates the segments, and pushes the names of each segment with the provided labels back to the token. The names of each segment along with labels and minor numbers are then stored in the disk's NVRAM. Because the boot segment should normally be read-only, we edit the token contents using a trusted workstation by removing the write label from the entry for the boot segment. Upon reinserting the token, the boot sector was exported read-only.

In order to verify that the disk successfully verifies the integrity of the boot sector, we modified replaced one block of the boot segment with random bytes and rebooted the SwitchBlade. At power on, the disk would no longer export the boot segment and recorded a message to the audit log containing the measurement from the disk and the expected value from the token.

Once the token containing a `create_disk` command for the boot segment is inserted, installation of the boot segment OS can proceed. The installation requires very few modifications from the standard installer. During the installation, the kernel module for the ATA over Ethernet driver is loaded and the boot segment block device is found. The installation procedure is able to correctly install the base system to the boot segment. Once the base installation finishes, the bootloader installation is skipped as the slug boots the host system using PXE.

Once the boot segment is installed, virtual machine creation can commence. Installation of the virtual machines requires no special modifications to the installation procedures. In order to interface with the virtual machines, Xen provides a VNC interface. These interfaces present the same view the user expects if installing the OS to a physical platform and not in a virtual machine. Once the virtual machine OS is installed, the OS sees a traditional physical platform, with no indication that it is actually running within a domU on the host system.

Along with the VNC interface for the virtual machines, the menu interface is provided for interaction with the boot segment, or dom0. The menu application interfaces with dom0 to determine what virtual machines are available, and presents the user with only valid virtual machine choices. The interface is show in the lower right corner of Figure 6. The interface is simple, only allowing the user to start and stop the listed VMs shown in the upper half of the window. The lower half of the window contains the audit log kept by the disk.

6 Evaluation

6.1 Security Analysis

The purpose of SwitchBlade is to provide storage level isolation to protect certain portions of storage from compromised programs running on the host. We provide a non-exhaustive list of attacks that are indicative of how systems may be attacked and the defenses that we provide. In addition, we discuss how SwitchBlade provides complete mediation of block requests in a manner consistent with reference monitor guarantees.

Live CD attacks One attack against which traditional storage devices are defenseless, is a takeover of the host through the use of a live CD. By booting a system from a live CD, an attacker can bypass file system-level security policies, accessing all data on the disk through the block interfaces. In the case of SwitchBlade, a live CD would in the best case not be able to access any segments due to the lack of physical token. Assuming a more powerful adversary that may posses some tokens, the accessible disk blocks are limited to those exported by the tokens.

Compromised OS A second example of an attack in which access to "bit-bucket" style storage devices is unrestricted, is the case of a compromised OS on the host system. Control of the OS allows an adversary to linearly scan the entire logical block address space, accessing all data on disk. Assuming a secure multiboot system, in the case of OS compromise, only the data in the segment of the currently running OS is accessible. Similarly, in a red-black isolation system, if the VMM is compromised, access to data on disk is limited to the set of segments visible to the VMM.

Availability attacks Another attack class not previously addressed by storage devices in multi-user systems is that of attacks on resource availability. In this case, the resource is free space. SwitchBlade may protect against attacks in which disk space is intentionally exhausted, or disk quotas surpassed. Because each segment has an associated size, quotas can easily be enforced by assigning a different segment to each user, where that segment's size is that user's quota. Similarly, because the permission to create new segments is limited to physical tokens containing `create_disk` commands, the global amount of free space on the disk may also be controlled.

Complete Mediation guarantees The above-described security semantics of SwitchBlade are only useful if we can guarantee that it provides complete and correct mediation of block requests. To do this, we reason about the verifiability and complete mediation properties of the SwitchBlade enforcement mechanism.

We can reason about complete mediation in terms of authorization hook placement. In the case of the SwitchBlade reference monitor, hook placement is simple enough to be verified in two dimensions, vertical and horizontal. We define horizontal hook placement as the number of hooks needed to mediate all resources, i.e., all of the execution points requiring authorization checks, and vertical hook placement as the overall complexity of each authorization check. Horizontal hook placement in SwitchBlade is nearly trivial. A single hook intercepts the serial queue of requests from the host. Because the semantics of each command are independent of other commands, the enforcement mechanism does not need to check for conditions that lead to common TOCTTOU attacks [28], in which dependent file system operations can lead to undetected policy violations. The vertical complexity of hooks is also minimal. Requests to SwitchBlade arrive over an ATA interface from the host. Each hook need only extract the ATA protocol's command code from the command descriptor block and check the target segment. The hook then checks for the label associated with that command for the target segment in the token plugged into the disk.

6.2 Performance

In order to better understand the bottlenecks of our prototype and how they may be improved upon, we measured the performance aspects of the prototype. We examined the overhead created by boot segment measurements and the time needed to load a VMM and guest OS from the SwitchBlade in light of the amount of data transferred, and then project these results onto typical SCSI storage devices to predict load times from a SwitchBlade on a typical storage bus.

We use `openssl` to perform cryptographic operations on the slug. To get an initial estimate of the throughput of computing sha1 hashes on the slug, we ran the `openssl speed sha1` command on the slug, which approximates the processing capability of what may be found in a disk. Measuring a 580 MB boot segment, the size needed for the Xen hypervisor and Debian dom0 kernel, took approximately 2 minutes and 45 seconds.

Obviously, a more efficient method of measurement is needed. This can be achieved either by reducing the size of the boot segment, or by improving the cryptographic hardware available to the SwitchBlade. The former case may be achieved by using a more minimal boot system, such as VMWare Server ESXi [53], which requires only 32 MB of

storage for the base system. We performed the same measurement on a 32 MB segment in approximately 8.15 seconds, which in the context of a system reboot is very low. In addition to reducing the size of the boot segment, performance gains may be made by offloading cryptographic operations to a special-purpose co-processor. High-performance ASICs can compute SHA-1 hashes at rates greater than 1.5Gbps [18].

7 Discussion

7.1 Token Management

As discussed in Section 4, there is a trusted token manager that is used in conjunction with the SwitchBlade. In a large-scale environment of deployed systems, the token manager may reside on a dedicated administrative workstation, isolated from the network. The manager has interfaces that allow for programming the token with its capabilities, placing `create_disk` and `delete_disk` commands on the token for the creation of new disk segments, and generating a label for the token, which in turn may map to a globally unique physical identifier on the token. Tokens may then be duplicated by inserting a blank token along with the token to be copied into the token manager, which would transfer the information to the blank token while retrieving its identifier for accounting purposes.

The administrator would keep a copy of a token separately from users to provide an easy method of revocation: the token manager can push a `revoke` command to the token that is revoked, which in turn would push that information to a SwitchBlade when it is plugged into the drive. SwitchBlade would then keep the revoked token's label in NVRAM such that if the token were plugged in by a user, it would not be accepted.

7.2 Disk Administration

Communicating with SwitchBlade Because the operating systems running in a SwitchBlade environment are not trusted, performing any maintenance and administrative tasks are more difficult. Above, we described a "sneakernet" mode of communicating information from the token manager to SwitchBlade. Griffin et al. [17] considered the issue of how to communicate with a disk in a secure manner. Their proposed method was to install a secret key in the disk's firmware and create a cryptographic tunnel between an administrative console and the disk, with the host system interposing on these requests and routing them to the disk through extensions to the underlying SCSI or ATA protocol. This form of communication requires the operating system to be modified to accommodate these requests, which may be arbitrarily dropped if the OS is compromised. Butler et al. [5] considered out-of-band methods of communicating with a disk, including potentially sharing a common bus or a wireless signal from a master machine that disks would listen for. If the environment dictates the need for rapid revocation [45], one of these communication methods may be appropriate.

Backup Using the token manager, an administrator can encode the "backup" capability onto a token for a single use. When this token is inserted into SwitchBlade and another device is connected, the segments specified to be exposed by the administrator will be transferred between disks. Metadata such as capability mapping will also be transferred. The administrator may choose to have all segments exposed for backup or to only backup a subset of the available disks. Because an administrative token with access to all segments may expose every segment on the disk, an administrator must be judicious regarding its use. For example, in a red/black storage setup, it may be desirable to perform a backup as two operations, for transferring red and black segments separately, to prevent exposure of different-colored segments to each other.

RAID Operation Operating SwitchBlade as a RAID configuration is possible if enforcement occurs at the level of the hardware RAID controller, which enforces segmentation. If a disk fails, it must be securely erased, particularly if it contains security-sensitive information: guidelines for disposal of sensitive media should be followed. Rebuilding a disk requires an administrative token programmed with the `rebuild` capability. The writing of data from multiple segments

to the new disk media happens at this stage and is unavoidable; thus, the RAID controller must be trusted to properly map the interleaved data to its correct disk location and to subsequently enforce segmentation.

7.3 SwitchBlade Applications

In this section, we provide two examples of how SwitchBlade's ability to enforce isolation through token-based policy may be used for real applications.

Voting systems SwitchBlade can enforce a separation of duties by requiring multiple capabilities to allow access to a segment. This may be a useful method for enforcing election procedures. For example, officials from two or more parties may need to be present in order to enable certain functionality in a voting machine, such as supervisor tasks or recount mechanisms. This would be enforceable by specifying multiple capabilities necessary for the segment to become available and giving each token one capability, requiring all of them to be inserted into the system for the segment to become available and the corresponding functionality to be unlocked.

Trusted auditing SwitchBlade may be used to enforce segments that are write-only segments except in the presence of a trusted auditor, who would insert a token in order to read the segment and examine the log that is written. Such enforcement mechanisms may be used to ensure compliance with internal controls as specified in Sarbanes-Oxley legislation. While write-once, read-many (WORM) systems allow append-only access (also possible with SwitchBlade), a write-only policy may be even more beneficial as it does not allow employees the ability to scrutinize the logs in advance of an audit. This "write-once" device is considered a viable method of ensuring a trusted audit [29].

8 Conclusion

This paper has presented SwitchBlade, a novel storage device architecture to provide isolation that is both defined and enforced below the storage interface. SwitchBlade enforces an isolation policy based on capabilities stored on physical tokens presented to the disk, providing a trusted path for policy configuration and user authentication. SwitchBlade provides isolation guarantees equal to those of physical separation, removing the need to trust OSes and VMMs to enforce isolation policies. We describe our experiences in building, configuring, using and evaluating a SwitchBlade prototype, and show how it may be used to implement secure multiboot and red-black style VM based systems. Inter-OS security properties are feasible through a token-based mechanism, and can provide a basis for whole-system security.

References

[1] AMD. AMD64 Virtualization Technology: Secure Virtual Machine Architecture Reference Manual, May 2005.

[2] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A Secure and Reliable Bootstrap Architecture. In *Proc. 1997 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1997.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, Oct. 2003.

[4] K. Butler, S. McLaughlin, T. Moyer, J. Schiffman, P. McDaniel, and T. Jaeger. Firma: Disk-Based Foundations for Trusted Operating Systems. Technical Report NAS-TR-0114-2009, Network and Security Research Center, Pennsylvania State University, Apr. 2009.

[5] K. R. B. Butler, S. McLaughlin, and P. D. McDaniel. Rootkit-Resistant Disks. In *Proc. 15th ACM Conference on Computer and Communications Security (CCS'08)*, Alexandria, VA, USA, Oct. 2008.

[6] Conraid Inc. ATA over Ethernet Protocol Definition. `www.coraid.com/RESOURCES/AoE-Protocol-Definition`.

[7] 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 *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, USA, Dec. 2002.

[8] EMC. Centera Compliance Edition Plus. `http://www.emc.com/centera`, 2007.

[9] Etherboot Project. Etherboot/gPXE. `http://www.etherboot.org`, Sept. 2008.

[10] M. Featherston. Encrypted LUKS disks store passphrase plaintext in memory. `https://bugs.launchpad.net/ubuntu/+bug/196368`, Feb. 2008.

[11] P. Ferrie. Attacks on Virtual Machine Emulators. In *Proceedings of the 9th Association of Anti-Virus Asisa Researchers International Conference (AVAR 2006)*, Auckland, New Zealand, Dec. 2006.

[12] E. Florio. Trojan.Mebroot. http://www.symantec.com/business/security_response/writeup.jsp?docid=2008-010718-3448-99&tabid=2, Jan. 2008.

[13] C. Fruwirth. LUKS - Linux Unified Key Setup. http://luks.endorphin.org, 2008.

[14] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, Oct. 2003.

[15] 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 *Proc. 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, USA, Oct. 1998.

[16] 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.

[17] J. L. Griffin, A. Pennington, J. S. Bucy, D. Choundappan, N. Muralidharan, and G. R. Ganger. On the Feasibility of Intrusion Detection Inside Workstation Disks. Technical Report CMU-PDL-03-106, Parallel Data Lab, Carnegie Mellon University, Dec. 2003.

[18] Helion Technology Ltd. Fast Dual SHA-1 and SHA-256 Hash Core for ASIC, 2005. http://www.heliontech.com/multihash.htm.

[19] W. W. Hsu and S. Ong. WORM storage is not enough. *IBM Systems Journal*, 46(2), Apr. 2007.

[20] IBM. IBM TotalStorage Enterprise. http://www-03.ibm.com/servers/storage, 2007.

[21] Intel Corporation. Preboot Execution Environment (PXE) Specification. http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf, Sept. 1999. Version 2.1.

[22] P. A. Karger, M. Zurko, A. Mason, and C. Kahn. A Retrospective on the VAX VMM Security Kernel. *IEEE Transactions on Software Engineering*, 17(11), Nov. 1991.

[23] 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 *Proc. 2006 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006.

[24] G. Kroah-Hartman. udev – A Userspace Implementation of devfs. In *Proc. Ottawa Linux Symp. (OLS)*, Ottawa, ON, Canada, July 2002.

[25] V. Kumar and N. Kumar. BOOT KIT: Custom boot sector based Windows 2000/XP/2003 Subversion. http://www.nvlabs.in/archives/5-BOOT-KIT-Custom-boot-sector%-based-Windows-2000XP2003-Subversion.html, Feb. 2007.

[26] Linksys. NSLU2 Product Information. http://www.linksysbycisco.com/US/en/products/NSLU2, Apr. 2008.

[27] S. E. Madnick and J. J. Donovan. Application and Analysis of the Virtual Machine Approach. In *Proceedings of the Workshop on Virtual Computer systems*, Cambridge, MA, USA, Mar. 1973.

[28] W. McPhee. Operating system integrity in OS/VS2. *IBM Systems Journal*, 13(3), 1974.

[29] National Computer Security Center. *A Guide to Understanding Audit in Trusted Systems*, NCSC-TG-001 (Tan Book) edition, July 1987.

[30] 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.

[31] 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.

[32] 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 *Proc. 12th USENIX Security Symposium*, Washington, DC, USA, Aug. 2003.

[33] remote-exploit.org. BackTrack. http://www.remote-exploit.org/backtrack.html, 2007.

[34] J. Rushby. Design and Verification of Secure Systems. *ACM SIGOPS Operating Systems Review*, 15(5), Dec. 1981.

[35] J. Satran, K. Meth, C. Sapuntzakis, and M. Chadalapaka. Internet Small Computer Systems Interface (iSCSI). RFC 3720, Apr. 2004.

[36] Seagate. Seagate Techonology - Momentus 5400 PSD Hybrid Hard Drives, Sept. 2007. http://www.seagate.com/www/en-us/products/laptops/momentus/momentus_5400_psd_hybrid/.

[37] Seagate Technology LLC. Self-Encrypting Hard Disk Drives in the Data Center. Technology Paper TP583.1-0711US, Nov. 2007.

[38] R. Sion. Strong WORM. In *Proc. 28th Intl. Conference on Distributed Computing Systems (ICDCS 2008)*, Beijing, China, June 2008.

[39] G. Sivathanu, S. Sundararaman, and E. Zadok. Type-Safe Disks. In *Proc. 7th Symposium on Operating Systems Design and Implementaition (OSDI'06)*, Seattle, WA, USA, Nov. 2006.

[40] M. Sivathanu, V. Prabhakarn, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Semantically-Smart Disk Systems. In *Proc. 2nd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, Apr. 2003.

[41] NSLU2 - Linux. http://www.nslu2-linux.org/wiki/SlugOS/HomePage, 2008.

[42] 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.

[43] E. H. Spafford. The Internet worm program: An analysis. *ACM Computer Communication Review*, 19(1):17–57, Jan. 1989.

[44] 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 *Proc. 4th Symposium on Operating Systems Design and Implementation (OSDI'00)*, San Diego, CA, Oct. 2000.

[45] S. G. Stubblebine. Recent-Secure Authentication: Enforcing Revocation in Distributed Systems. In *Proc. 1995 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1995.

[46] Sun Microsystems. Trusted Solaris Operating System, Nov. 2005. http://www.sun.com/software/solaris/trustedsolaris/index.xml.

[47] S. Sundararaman, G. Sivathanu, and E. Zadok. Selective Versioning in a Secure Disk System. In *Proc. 17th USENIX Security Symposium*, San Jose, CA, USA, July 2008.

[48] The LVM Project. LVM2 Resource Page. `http://sourceware.org/lvm2/`.

[49] The OpenSSL Group. OpenSSL. `http://www.openssl.org/`, May 2000.

[50] TrueCrypt Foundation. Truecrypt. `http://www.truecrypt.org`, 2008.

[51] Trusted Computing Group, Mar. 2005. `http://trustedcomputinggroup.org`.

[52] United States Department of Defense. Red/Black Engineering–Installation Guidelines. Military Handbook MIL-HDBK-232A, July 1988.

[53] VMware. VMware ESXi, Hypervisor for Server Virtualization. `http://www.vmware.com/products/esxi/`.

[54] 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 *Proc. 1st International Conference on Trusted Computing and Trust in Information Technologies (TRUST 2008)*, Villach, Austria, Mar. 2008.

[55] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proc. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, USA, Dec. 2002.