New Security Architectures Based on Emerging Disk Functionality

Advances in hard disk technologies can help manage the complexity of operating system security and enforce security policies. The SwitchBlade architecture provides isolation for multiple OSs running on a machine by confining them into segments that users can only access using a physical token.

KEVIN BUTLER
University of Oregon

STEPHEN MCLAUGHLIN, THOMAS MOYER, AND PATRICK MCDANIEL
Pennsylvania State University

Computers are more complex than ever before, and their increased capabilities mean that previously unimagined functionality is now available to the average user. This power and functionality means that operating systems have also become more complex to control these systems. Modern OSs have increased their available features, the complexity of their graphical interfaces, and their compatibility with a vast array of external peripherals. The net result has been an explosion in size. For example, Windows NT 1.0 comprised approximately 4 to 5 million lines of code and had a 200-person development team; by contrast, Windows XP consisted of approximately 40 million lines of code, with an 1,800-member development team.[1]

In addition to using these larger OSs, users are interacting with large-scale programs such as Web browsers and office suites, with code bases comprising millions of lines of code. Furthermore, users can now communicate with potentially anyone in the world, thanks to the ubiquity and accessibility of the Internet. Combining the size and complexity of applications and OSs with the almost limitless interactions possible between users and remote parties means that fully understanding every facet of their operations is virtually impossible. In particular, securing these systems is an extraordinarily difficult proposition.

As the demands on computers have grown, storage has risen to the challenge and has undergone significant architectural changes. Hard disks are still the primary mechanism for storing system data, but the past few years have seen these disks capable of broad new functionality. These disks have witnessed sharp rises in onboard processing capacity. In addition, recently introduced hybrid hard disks (HHDs) have included large amounts of nonvolatile RAM to be used as an extended cache, whereas full-disk encryption (FDE) disks, capable of performing cryptographic operations such as encrypting data at the speed of the disk interface, have resulted in the introduction of cryptographic processing chips within the drive enclosure. Essentially, these disks have become full-fledged embedded computing systems. Although their tasks have increased in size and complexity, these disks represent a trusted computing base that is far smaller than that of the modern desktop or server OS. Because they completely mediate access to secondary storage in these computers, they are well-positioned to act as enforcement points for security policies.

These advances in hard disk technologies provide a means to manage the complexity involved in securing operating systems. The new disk-level functionality allows them to be used as security policy enforcement sites that are autonomous from the rest of the system. In this article, we explain how to construct and use these disks and focus on how they can provide isolation for multiple OSs running on one machine. We introduce the SwitchBlade architecture, which confines OSs stored on the same disk into segments that users can only access using physical tokens. We describe our implementation, evaluate its security and performance, and show that the isolation guarantees SwitchBlade provides are equivalent to physically separate systems without the traditional usability burdens.

Autonomously Secure Disks

Prior works in this area have proposed using disks to provide services such as detecting violations by using an audit log to compare versioned data.[2] This approach has also been used for intrusion detection.[3] Our work, however, focuses on how to combine functionality currently available in disks to provide a general security platform. Figure 1 shows an architecture for these autonomously secure disks.

Our earlier work in the area posited on the potential for the new forms of storage security that these disks could provide.[4] We focused on three sample applications to provide new functionality:

- authenticated encryption, where integrity and encryption could be simultaneously performed and integrity metadata stored in the drive's nonvolatile storage;
- capability-based access control for finer-grained access control to the level of blocks on the disk; and
- supporting information-flow preserving systems by labeling blocks on the disk and mediating access based on the labeled request received from the operating system.

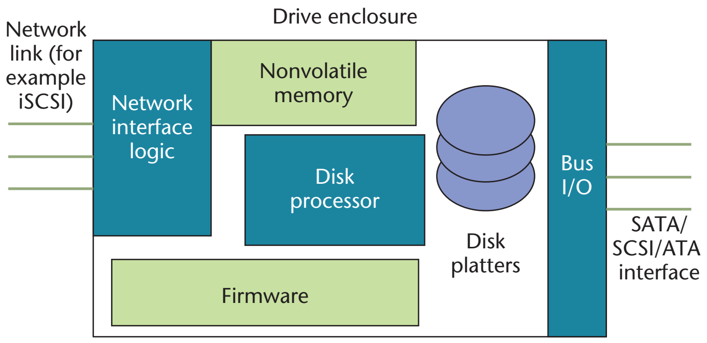


Figure 1. Proposed architecture for autonomously secure disks. This disk would be able to communicate over traditional disk interfaces or potentially over a network for supporting network-attached storage (NAS) services such as iSCSI. We use nonvolatile memory for metadata storage while an augmented disk processor performs computation and enforces policies.

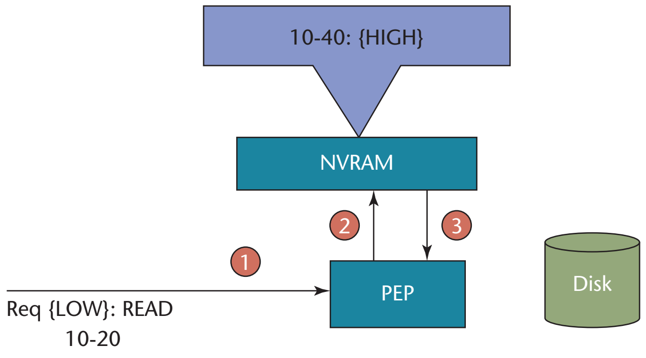


Figure 2. An example of enforcing information flow with disk metadata. The client marked by the OS with a low security level has requested to read blocks 10 to 20 (step 1). The policy enforcement point (PEP), which is usually the processor, consults the nonvolatile memory and finds that blocks 10 to 40 are labeled with a high security level (step 2). When this information is returned to the PEP (step 3), it denies the request, and no disk access is made.

Figure 2 provides an example of how such an information-flow preservation system would operate, with the disk capable of mediating over ranges of blocks by comparing request labels to those stored in nonvolatile memory. In this example, the policy enforcement point (PEP) can be instantiated as the disk's processor, with the actual policy retrieved from the firmware, as well as the labels of the stored data. The model in Figure 2 shows a multilevel security (MLS) policy to preserve data confidentiality.[5] A user marked by the OS with a low security level is attempting to read a set of blocks with a high security level, causing the disk to deny the request.

We came to realize, however, that many of our assumptions about what new functionality we could support rested on our ability to work in conjunction with the OS, which could prove problematic in the face of complex OSs that are subject to compromise through many vectors. Accordingly, we took a step back to consider what services we could provide to enhance host security in the face of this pervasive threat. We realized that the best way to allow a user or administrator to communicate with the disk without operations being subverted by a potentially compromised OS was to bypass it altogether.

We devised a prototype for a disk that accepts policies directly from tokens inserted into the disk. We used this architecture to show how we could protect the OS from persistent *rootkits*, which compromise critical binaries and configuration files in order to insert themselves into the boot process, such that users can't remove them. Our rootkit-resistant disks support the concept of block immutability.[6] That is, when the OS is installed on the disk, the installation process is staged such that files that shouldn't be modified during regular OS operation (such as system configurations or files in the `usr/bin` directory on a Unix platform) are written to the disk with an immutable token inserted into the drive. The token is removed for the remainder of the installation. Through this operation, we ensure that these critical files can't be overwritten during normal system operation. Even if the OS kernel's in-memory image is compromised, any attempts to overwrite the files without the token inserted will be denied. Importantly, when the system is rebooted, because the malware can't take root, it

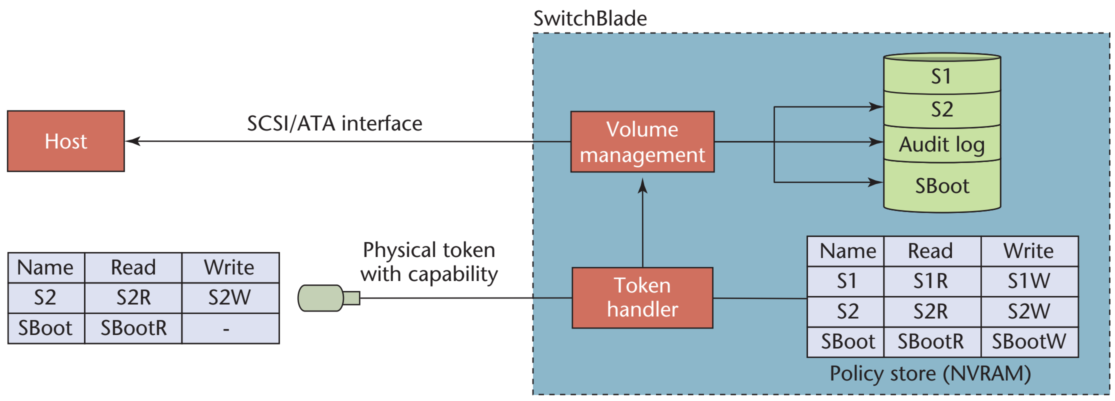


Figure 3. SwitchBlade architecture. A physical token contains the set of labels for segments to be exported—in this case, S2 and the boot segment (SBoot), the latter of which is exported as read only. The audit log is read only by default and writable by the disk controller. In this case, the token handler constitutes the policy decision point that configures volume management to export the segments allowed by the current physical token according to the policy, which is stored in nonvolatile random access memory (NVRAM).

will be eradicated from the in-memory image.

We demonstrated this to be the case by compromising one of our computers with the Mood-NT rootkit, which attacks Linux systems. The rootkit makes itself persistent on a system by replacing the file `/sbin/init` with its own initialization program, and it subsequently installs hooks into the system call table when the victim host computer is rebooted, running a backed-up version of `init` to start the system as usual. As a result, its operation is hidden from users. However, because our compromised system was using a rootkit-resistant disk, the Mood-NT rootkit was unable to override `init`. Consequently, when we rebooted the system, the rootkit wasn't installed as part of the boot process and wasn't active in memory.

Protecting Multiple OSs with Disk Segmentation

This solution gives us a way of protecting the OS by forming a trusted path directly between the storage and the user. We realized, however, that many users are running more than one OS from the same disk. Multiboot systems let users natively boot into various OSs—a common example is Apple's Boot Camp, which lets Mac users switch between Mac OS X and Windows. Virtualization has become increasingly attractive on desktops as well because it lets users simultaneously run multiple OSs. Being able to isolate these OSs from each other, however, is critical. For OSs that share a common storage medium, it's possible to tamper with other on-disk OSs. Although OSs run within their own partitions, these mechanisms are constructions enforced at the OS level; nothing prevents the OS from maliciously reading or overwriting arbitrary blocks in any other partition.

Building on our experience with rootkit-resistant disks and the methodology of using a token for supplying intent to the disk independently of the OS, we developed our disk-protection model SwitchBlade to isolate OSs from each other. (The name SwitchBlade comes from its ability to let users switch between virtual "blades" running their own OS.) An OS is only able to run within its own disk segment, a defined set of disk blocks accessible to the OS as a physically separate disk. Disk users possess a physical token containing read and write capabilities to one or more disk segments, and they plug these tokens into the disk, ensuring enforcement of these properties by the disk itself. Unlike with rootkit-resistant disks, the token remains plugged in during operation as it supplies context to the system, and when it's removed, access to the disk segments is also removed.

We've explored a multiboot mode of operation, where the user accesses a different view of the disk and its available OSs, depending on the token plugged into the disk. We also examined a virtualized mode, where a virtual machine monitor (VMM) and guest OSs are exposed to the user and the available set of OSs is defined by token policy. This lets us enforce isolation between sets of VMs that might be differentiated by security class. These features allow for a new range of functionalities, including enforcing of a separation of duties to unlock a disk segment and enforcing auditing requirements.

Architecture

Figure 3 shows the SwitchBlade architecture. The disk is divided into segments, analogous to memory segments used in microprocessor architectures for providing hardware-level memory protection. To the host, the segment appears to be a single device on the storage bus, and it's addressed as though it's an actual disk. The accessible segments are determined by policy encoded on the token. In this case, the token contains the ability to read and write the disk segment S2; it can read the boot segment (SBoot) but can't write to it. Multiple segments accessible to the user are on the disk, along with others that aren't, including the audit log and SBoot. The disk's nonvolatile memory stores attributes for each disk segment, allowing concurrent access to policy metadata and disk data.

If the token user is an administrator, the audit log is exported, allowing viewing of past access decisions. Enforcement of access policy takes place within the disk's firmware and is independent of the rest of the disk controller code. It mediates all I/O requests at the disk level, inspecting them to see if sensitive operations requiring a policy decision are necessary. The token plugged into the disk determines which segments are available for access. Decoupling the enforcement and controller code allows disk requests to be pipelined, improving performance.

Management

For general use of SwitchBlade, we assume that each segment contains one OS. This might be a guest OS under a VMM's control or a stand-alone OS available as part of a multiboot configuration. Each OS views its segment as a separate disk, with separate file systems and swap partitions within these segments. If desired, the disk policy can also define segments that only contain data, which can act as shared storage between multiple running OSs.

For a multiboot system, using SwitchBlade is simple. We can maintain isolation because the OS doesn't know about any other storage existing on the disk beyond itself. For virtualized operation, we provide better isolation guarantees between running guest VMs by trusting the VMM to multiplex access to virtual disks. 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. This means that if the VMM is compromised, only the segments exposed to it, which might be only a subset of available segments on the disk, will be exposed.

Booting from the disk is similar to a regular boot process. The equivalent of a disk's master boot record (MBR) in SwitchBlade is the SBoot, which has several additional features beyond those found in a typical MBR. First, because segments can be of arbitrary size, the SBoot contains an enhanced boot loader since it needs to inspect the disk to discover which segments contain OSs. Second, the SBoot is almost always kept in a read-only state to protect the boot code's integrity, ensuring that the system can be booted into a safe state. Of course, trusting the code on the SBoot is highly important; advances in proving that a system is based on a trustworthy installation can help strengthen these guarantees.[7]

Exporting the SBoot lets SwitchBlade provide integrity guarantees for its content. Once executed on the host, the SBoot can further verify the integrity of higher levels of software by maintaining a measurement of a good SBoot on the token and comparing this value to a measurement of the SBoot. If the SBoot's measurement doesn't match the value stored on the token, SwitchBlade will refuse to satisfy requests for blocks in the SBoot and will append a message to the tamper-proof audit log.

The SBoot should be measured when the disk is powered on and when it has previously been exported as writable. In the former case, the SBoot contents might have been tampered with, and, in the latter case, they might have been altered by a compromised host OS. In the case of a regular upgrade of the SBoot's software, 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 SBoot.

This approach provides guarantees about the SBoot's state, but we don't guarantee against physical tampering of the disk's hardware, and, as such, we don't provide attestation of it. This might be possible if a device such as a trusted platform module (TPM) is installed in the disk for attestation to the host system or an active token. If we consider the physical components of the disk to be trusted, the disk acts as a root of trust.

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 operate in multiboot and virtualized mode. Unfortunately, the technical challenges of obtaining and modifying the firmware of a commodity disk are considerable, and the legal difficulties make the process infeasible. Therefore, we made our prototype using a modified network-attached storage (NAS) device, the Linksys NSLU2 Network Storage Link, which is commonly known as a *slug*. The main difference between the prototype's interface and that of a commodity disk drive is that it uses Ethernet for the physical and MAC layers between the disk and host, as opposed to an Advanced Technology Attachment (ATA) or Small Computer System Interface (SCSI) bus. Our prototype would work just as well with ATA or SCSI, however, if the interfaces on the disk were accessible to us.

To minimize differences between our prototype's command interface and that of a real disk, we used the ATA over Ethernet (AoE) protocol between the disk and host. As the name implies, AoE sends ATA com-

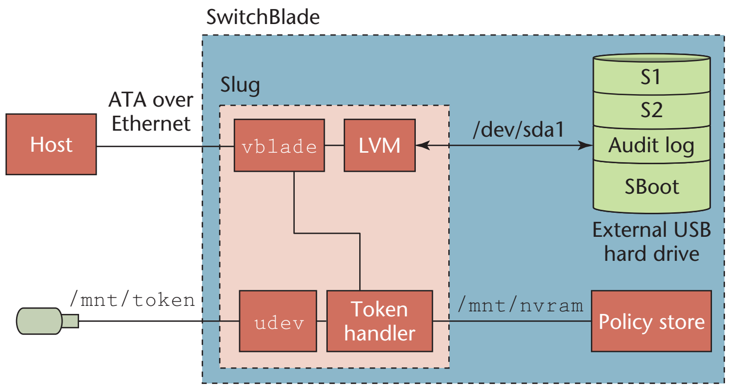


Figure 4. The SwitchBlade prototype. The token handler parses the contents of the USB token, mounted by `udev`. It then invokes `vblade` to export the logical volumes the token allows. The audit log is also managed as a logical volume that `vblade` exports as read only, but it's directly writable by the token handler.

mands directly over the Ethernet interface between a client and host. In a typical AoE installation, a host machine, or initiator, issues ATA commands to a target storage device. For our SwitchBlade prototype, the initiator consists of the host machine using one or more AoE block devices, which issue commands to AoE targets on the LAN. The targets consist of the modified NAS devices running `vblade`, a server program that listens for ATA commands via raw sockets. (It also provided us with inspiration for naming the architecture.) Individual AoE devices are addressed by a major and minor number the target uses to demultiplex commands to each device.

Figure 4 shows the SwitchBlade prototype, which consists of two discrete physical components: an external USB hard disk that provides the actual storage, and the slug, which sits between the host and disk that acts as the policy store and enforcement mechanism. Our prototype uses the segment as the basic unit of storage over which access control is performed. Segments are implemented in our prototype using the Linux logical volume manager (LVM). We specified them as logical volumes (LVs) over the single physical volume (PV) comprising the external USB storage device. We then used 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 it's visible to processes on the host as a block device containing the major and minor number of the segment—for example, `/dev/e1.1`, `/dev/e1.2`, and so forth.

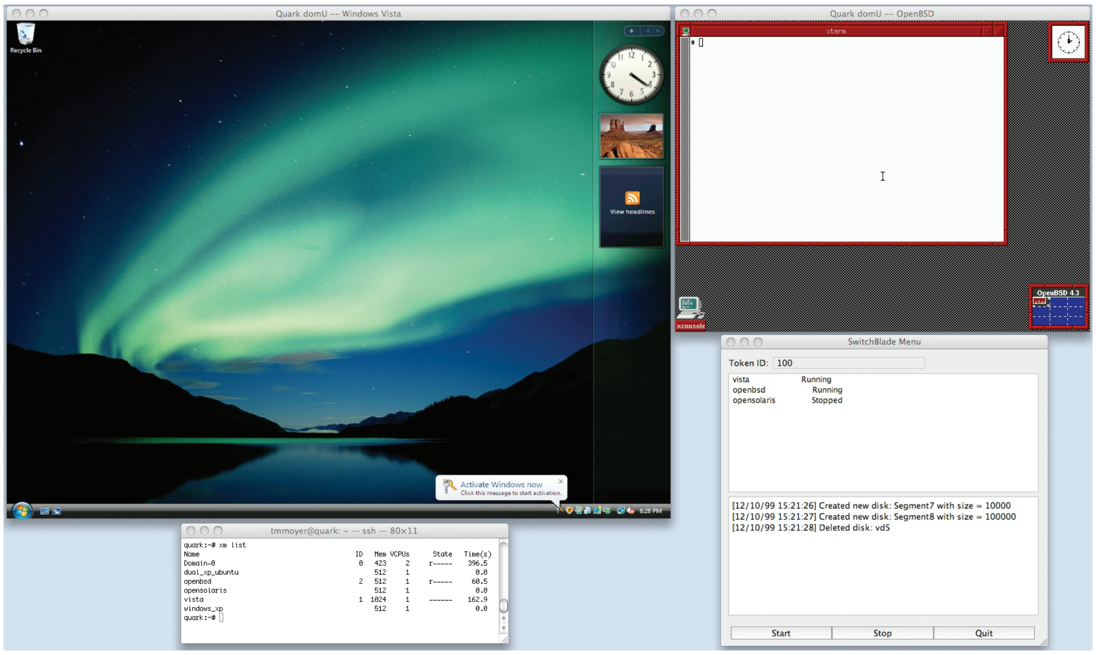
We easily used the implementation of segments to create the tamper-proof audit log. We started an instance of `vblade` to export the audit log read-only at disk power-on. When the token handler executes any commands or detects SBoot tampering, it appends detailed messages to the audit log by writing directly to the LV. Any guest OS can obtain the audit log contents by locally mounting the AoE device with minor number 0 and reading the content of, for example, `/mnt/audit_log`.

For each segment in the prototype, an entry is stored in the disk's nonvolatile random access memory (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's writable. If only a read label is present, the segment is exported as read only, and if only a write label or no labels are present, the segment isn't 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.

SwitchBlade verifies the SBoot's integrity against a measurement stored on the token. The SBoot must be measured after the disk is powered on and after the SBoot has been exported as writable because 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 SBoot is exported as writable. Any time the file isn't present, the SBoot's SHA-1 hash is computed with OpenSSL and compared against the hash stored on the slug. If an administrator has intentionally altered the SBoot's content—for example, for a boot loader upgrade—he or she must then ensure that the proper new measurement is stored on the token.

Multiboot Operation

To provide a multiboot environment, we use the Preboot Execution Environment (PXE) 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. To boot using the AoE targets made available by the slug typically requires some modifications to the OS. 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 to boot over AoE.



```
quark:~# xm list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0   423     2     r-----     396.5
dual_xp_ubuntu                                   512     1                  0.0
openbsd                                      2   512     1     r-----      60.5
opensolaris                                      512     1                  0.0
vista                                        1  1024     1     ------     162.9
windows_xp                                       512     1                  0.0
quark:~# 
```

```
Token ID: 100

vista          Running
openbsd          Running
opensolaris      Stopped

[12/10/99 15:21:26] Created new disk: Segment7 with size = 10000
[12/10/99 15:21:27] Created new disk: Segment8 with size = 100000
[12/10/99 15:21:28] Deleted disk: vd5
```

Figure 5. Windows Vista and OpenBSD 4.3 running in virtual machines (VMs). 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 shows only 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.

When the system is first powered on, it sends a broadcast request looking for a DHCP server, which sends a response indicating the Trivial File Transfer Protocol (TFTP) server's location. 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. 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.

The PXE and OS boot modifications are necessary only because we deployed the SwitchBlade logic on the slug. On a production device, the kernel image would be booted directly off the SBoot over an ATA or SCSI interface, which would obviate the need for AoE.

Virtualized Operation

To implement virtualized mode, we used the Xen VMM to support running VMs.[8] The host system contained a processor with AMD virtualization extensions, letting Xen run unmodified guest OSs. The initial setup involved installing the hypervisor and administrative domain (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 to allow PXE to boot.

Once the system was rebooted, Xen was installed and the kernel and `initrd` were again copied to the slug along with the hypervisor. A final reboot let the slug serve the VMM image, and creation of guest domains (domUs) could occur at this point. We installed four different guest OSs on SwitchBlade, including Windows XP, Windows Vista, OpenBSD 4.3, and OpenSolaris. To demonstrate that each segment considers itself 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 user interface, we developed a menu interface that shows the user which VMs are currently able to boot given the inserted token (see Figure 5). The user is also presented with an audit log that shows token activity from the disk. Figure 5 shows four different windows. The top two windows are VNC connections to the VMs that are running 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 what VMs allow by the currently inserted token. The window in Figure 5 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.

Security Analysis

To illustrate SwitchBlade's strong segment isolation guarantees, we evaluated its protections in light of a nonexhaustive, yet representative list of attacks against storage. Our results show that SwitchBlade can limit data access in the event that host-level protection mechanisms are circumvented.

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, thus accessing all data on the disk through the block interface. In a SwitchBlade system, a live CD wouldn't be able to access any segments because it lacks a physical token.

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. By controlling the OS, an adversary can linearly scan the entire logical block address space, accessing any data on the disk. Assuming a secure multiboot system, an OS compromise will only affect data in the segment corresponding to the running OS. 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 according to the current token policy.

This example illustrates the limitations of using TPM-based integrity measurement alone. In the event that a host OS or VMM is subverted, it can simply stop providing measurements of newly loaded code to the TPM, effectively stopping integrity measurement. Even when recorded correctly, integrity measurements provide minimal forensic value compared to the audit log SwitchBlade maintains. Although a measurement list can reveal that some untrusted or unknown program was loaded, it says nothing about what disk accesses it might have attempted or if they were allowed.

As a final example, attacks on available storage space haven't been previously addressed by storage devices in multiuser systems. SwitchBlade implicitly protects against unauthorized storage exhaustion. The protection in this case comes from the implicit quota placed on segment size by the token policy. Because the permission to allocate new segments is limited to users with privilege to execute `create_disk`, control over the total amount of free space on the disk is limited.

Performance

We used `openssl` to perform cryptographic operations on the slug to determine performance bottlenecks. Our SBoot, which contained a Xen hypervisor and Debian dom0 kernel, was 580 Mbytes in size and took approximately 2 minutes and 45 seconds to hash over.

Clearly, we need a more efficient measurement method. We can achieve this by either reducing the SBoot's size or improving the cryptographic hardware available to the SwitchBlade. The former case is possible by using a more minimal SBoot, such as VMWare Server ESXi (www.vmware.com/products/esxi), which requires only 32 Mbytes of storage for the base system. We performed the same measurement on a 32-Mbyte segment in approximately 8.15 seconds, a small amount of time compared to the length of a system reboot. In addition to reducing SBoot's size, performance gains might be made by offloading cryptographic operations to a special-purpose coprocessor. High-performance ASICs can compute SHA-1 hashes at rates greater than 1.5 Gbps.[9]

Applications for SwitchBlade

SwitchBlade's ability to enforce isolation through token-based policy makes it applicable for real-world applications. For example, SwitchBlade can enforce a separation of duties by requiring multiple capabilities to allow access to a segment. This might be a useful method for enforcing election procedures. For example, officials from two or more parties might need to be present to enable certain functionality in a voting machine, such as supervisor tasks or recount mechanisms. We could enforce this 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.

In addition, SwitchBlade can be used to enforce segments that are write-only segments except in the presence of a trusted auditor, who would insert a token to read the segment and examine the written log. Such enforcement mechanisms could help ensure compliance with internal controls as specified in Sarbanes-Oxley legislation. Although write-once, read-many (WORM) systems allow append-only access (also possible with SwitchBlade), a write-only policy might be even more beneficial because it doesn't allow employees to scrutinize the logs in advance of an audit. This write-once device is considered a viable method of ensuring a trusted audit.[10]

Our work on SwitchBlade and other problems related to storage security is ongoing. Readers interested in more details can read our extended technical report on the project.[11] One of the intriguing questions that we discovered through our investigation was the need for a mechanism to measure the SBoot where a VMM would reside. Some of our recent work has shown how we can root a secure boot mechanism in storage with only a TPM necessary in the host system.[12] In this manner, we can provide the trusted VMM postulated by work such as Terra.[13]

Our investigations have scratched the surface of what is possible with the new capabilities available from storage. With ever-more functionality appearing on the horizon, and increasing OS support for technology appearing within storage devices, we are poised to develop an array of new methods to help us in the ever-expanding battle to keep our computers safe and secure. ☐

Acknowledgments

This work was supported by US National Science Foundation grants CCF-0937944 and CFF-0621429.

References

1. V. Marala, *The Build Master: Microsoft's Software Configuration Management Best Practices*, Addison-Wesley, 2005.
2. J.D. Strunk et al., "Self-Securing Storage: Protecting Data in Compromised Systems," *Proc. 4th Symp. Operating Systems Design and Implementation* (OSDI 00), vol. 4, Usenix Assoc., 2000, pp. 165–180.
3. A.G. Pennington et al., "Storage-Based Intrusion Detection: Watching Storage Activity for Suspicious Behavior," *Proc. 12th Usenix Security Symp.*, Usenix Assoc., 2003, pp. 137–152.
4. K. Butler, S. McLaughlin, and P. McDaniel, "Non-Volatile Memory and Disks: Avenues for Policy Architectures," *Proc. 1st ACM Computer Security Architectures Workshop* (CSAW 07), ACM Press, 2007, pp. 77–84.
5. D. Bell and L. LaPadula, *Secure Computer Systems: Mathematical Foundations and Model*, tech. report M74-244, Mitre, 1973.
6. K. Butler, S. McLaughlin, and P.D. McDaniel, "Rootkit-Resistant Disks," *Proc. 15th ACM Conf. Computer and Communications Security* (CCS 08), ACM Press, 2008, pp. 403–416.
7. L. St. Clair et al., "Establishing and Sustaining System Integrity via Root of Trust Installation," *Proc. 23rd Ann. Computer Security Applications Conf.* (ACSAC 07), 2007, pp. 19–29.
8. P. Barham et al., "Xen and the Art of Virtualization," *Proc. 19th ACM Symp. Operating Systems Principles* (SOSP 03), ACM Press, 2003.
9. "Fast Dual SHA-1 and SHA-256 Hash Core for ASIC," Helion Technology, 2005; www.heliontech.com/multihash.htm.
10. *A Guide to Understanding Audit in Trusted Systems*, tech. report NCSC-TG-001, Tan Book, ed., Nat'l Computer Security Center, 1987.
11. K. Butler et al., *SwitchBlade: Policy-Driven Disk Segmentation*, tech. report NAS-TR-0098-2008, Network and Security Research Center, Dept. of Computer Science and Eng., Pennsylvania State Univ., 2008.
12. K. Butler et al., *Firma: Disk-Based Foundations for Trusted Operating Systems*, tech. report NAS-TR-0114-2009, Network and Security Research Center, Dept. of Computer Science and Eng., Pennsylvania State Univ., 2009.
13. T. Garfinkel et al., "Terra: A Virtual Machine-Based Platform for Trusted Computing," *Proc. 19th ACM Symp. Operating Systems Principles* (SOSP 03), ACM Press, 2003, pp. 193–206.

***Kevin Butler** is an assistant professor in the Department of Computer and Information Science at the University of Oregon. His research interests include systems, storage, and network security. Butler has a PhD from Pennsylvania State University. He's a member of IEEE, the ACM, and Usenix. Contact him at butler@cs.uoregon.edu.*

***Stephen McLaughlin** is a doctoral candidate in the Systems and Internet Infrastructure Security Lab in the Department of Computer Science and Engineering at Pennsylvania State University. His research interests include autonomously secure storage systems, Scada security, and the privacy implications of smart metering infrastructure. McLaughlin has a BS in computer science from Penn State. Contact him at smclaugh@cse.psu.edu.*

***Thomas Moyer** is a doctoral candidate researching network and systems security in the Systems and Internet Infrastructure Laboratory in the Computer Science and Engineering Department at Pennsylvania State University. His research interests include Web security, systems security, distributed system security, and large-scale network configuration. Moyer has an MS in computer science and engineering from Penn State. Contact him at tmmoyer@cse.psu.edu.*

***Patrick McDaniel** is an associate professor in the Computer Science and Engineering Department at Pennsylvania State University and is codirector of the Systems and Internet Infrastructure Security Lab. His research efforts focus on network, telecommunications, systems, and language-based security and technical public policy. He's the editor in chief of* ACM Transactions on Internet Technology *and is an associate editor of* ACM Transactions on Information and System Security, IEEE Transactions on Software Engineering, *and* IEEE Transactions on Computers. *McDaniel has a PhD in computer science and engineering from the University of Michigan, Ann Arbor. Contact him at mcdaniel@cse.psu.edu.*