New Security Architectures Based on Emerging Disk Functionality

Kevin Butler, Stephen McLaughlin, Thomas Moyer, and Patrick McDaniel

Computers are more complex than ever before. Their increased capabilities mean that previously-unimagined functionality is now available to the average user. This increase in power and functionality has come at the cost of increased complexity of the operating systems that control these systems. Modern OSes have increased their available features, the complexity of their graphical interfaces, and their compatability with a vast array of external peripherals. The net result of this has been an explosion in their size. For example, Windows NT 1.0 comprised approximately 4-5 million lines of code and had a development team of 200; by contrast, Windows XP consisted of approximately 40 million lines of code, with an 1,800-member development team [9]. In addition to using these larger operating systems, users are interacting with large-scale programs such as web broswers and office suites, with codebases comprising millions of lines of code. Furthermore, the scope of who users communicate with has expanded to include potentially anyone in the world, thanks to the ubiquity and accessibility of the Internet. Combining the size and complexity of applications and operating sytems with the almost limitless interactions possible between users and remote parties means that fully understading 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 last few years have seen these disks capable of broad new functionality. These disks have witnessed sharp rises in on-board processing capacity. In addition, recently-introduced hybrid hard disks (HHDs) have included large amounts of non-volatile RAM to be used as an extended cache, while full-disk encryption disks (FDE), 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. While 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 operating system. Because they completely mediate access to secondary storage in these computers, they are well-positioned to act as enforcement points for security policy.

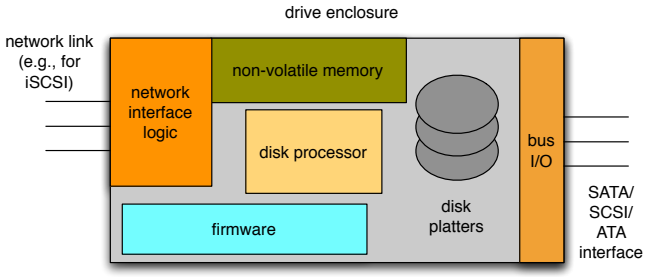

Figure 1: Proposed architecture for *autonomously secure disks*. This disk would be able to communiate over traditional disk interfaces, or potentially over a network for supporting NAS services such as iSCSI. Non-volatile memory is used for metadata storage while policies and computation can be performed by an augmented disk processor.

Autonomously Secure Disks

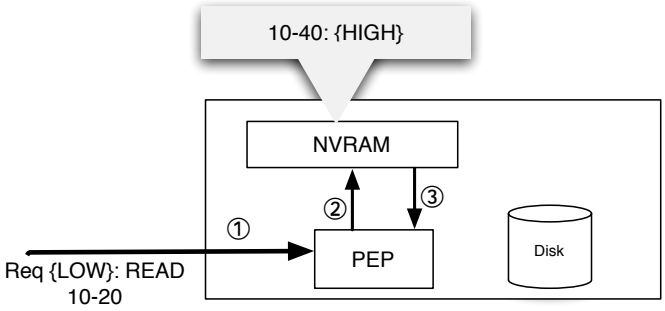

Figure 2: An example of enforcing information flow with disk metadata. A request from a client labeled LOW to read blocks 10-20 is made (step 1). The policy enforcement point, or PEP (usually the processor), consults the non-volatile memory and nds that blocks 10-40 are labeled HIGH (step 2). When this information is returned to the PEP (step 3), it denies the request and no disk access is made.

Prior works in the area have considered how disks can be used to provide services such as detecting violations through use of an audit log [13] for comparing versioned data; this approach was also used to provide for intrusion detection [11]. Our work, however, has focused on how functionality currently available in disks can be combined to provide a general platform for security. An architecture for these *autonomously secure disks* is shown in Figure 1. Our early work in the area [3] posited on the potential for new forms of storage security that these disks could provide. We focused on three sample applications to provide new functionality: 1) authenticated encryption, where integrity and encryption could be simultaneously performed and integrity metadata stored in the drive's non-volatile storage; 2) capability-based access control for finer-grained access control to the level of blocks on the disk; and 3) 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 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 non-volatile memory. In this example, the policy enforcemement 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 shown here is for a multi-level security (MLS) policy to preserve data confidentiality, as proposed by Bell and La Padula [2]. A user marked by the OS as having a security level of LOW is attempting to read a set of blocks labeled as HIGH, causing the read to be denied by the disk.

We came to realize, however, that many of our assumptions of what new functionality could be supported were on the ability to work in conjunction with the operating system, which could prove to be problematic in the face of these complex OSes 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 the pervasive threat to operating systems. 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 policy directly from a token inserted into the disk. Ws used this architecture to show how the OS could be protected from *persistent rootkits*, which compromise critical binaries and configuration files in order to insert themselves into the boot process, such that they cannot be removed by the user. Our *rootkit-resistant disks* [4] support the concept of block *immutability*: when the OS is installed to the disk, the installaton process is staged such that files that should not be modified during regular OS operation (e.g., system configurations, 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 cannot be overwritten during normal operation of the system. Even if the in-memory image of the OS kernel is compromised, any attempts to overwrite the files without the token inserted will be denied. Importantly, when the system is rebooted, because the malware cannot take root, it 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 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 will be hidden from the users. However, because our compromised system was using a rootkit-resistant disk, the Mood-NT rootkit was not able to override `init`. Consequently, when the system was rebooted, the rootkit was not installed as part of the boot process, and it was not active in memory.

Protecting Multiple Operating Systems with Disk Segmentation

This solution provides us with a way of protecting the operating system by forming a trusted path directly between the storage and the user. We realized, though, that many users are running more than one operating system from the same disk. Multiboot systems allow users to natively boot into a variety of operating systems; a common example nowadays is the user of Apple's Boot Camp to allow Mac users to switch between Mac OS X and Windows. Virtualization has become increasingly attractive on the desktop as well, with the ability for users to simultaneously run multiple operating systems. Being able to isolate these OSes from each other, however, is critical. For OSes that share the common medium of storage, an opportunity to tamper with other on-disk OSes is possible. While OSes run within their own partitions, these mechanisms are constructions enforced at the OS level; there is nothing preventing the OS from maliciously reading or overwriting arbitrary blocks in any other partition.

The remainder of this article describes our solution to this problem. 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 present a disk protection model called SwitchBlade, which isolates operating systems from each other. (As described later, the name comes from the ability for a user to 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. Users of the disk possess a physical token containing read and write capabilities to one or more disk segments, and plug these tokens into the disk, ensuring enforcement of these properties by the disk itself. We have explored a multiboot mode of operation, where the user accesses a different view of the disk and its available operating systems, depending on the token plugged into the disk. We also examine virtualized mode, where a virtual machine monitor (VMM) and guest operating systems are exposed to the user and the available set of OSes is defined by token policy. This allows us to enforce isolation between sets of virtual machines that may be differentiated by security class.

Architecture

Figure 3 shows the architecture of SwitchBlade. The disk is divided into *segments*, analagous 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 is addressed as though it is 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* and it can read *SBoot*, but cannot write to it. Multiple segments accessible to the user are on the disk, along with others that are not, including the audit log and boot segment

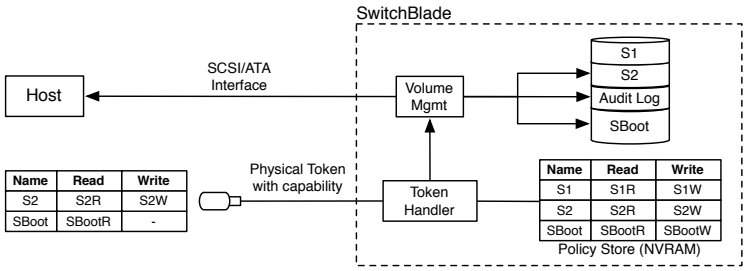

Figure 3: SwitchBlade architecture. A physical token contains the set of labels for segments to be exported, in this case, S2 and SBoot, the latter of which is exported as read only. The audit log is read only by default and writable by the disk controller. In this case, the Token Handler constitutes 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.

(*SBoot*). The disk's non-volatile 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 necesary. The token plugged into the disk provides context in terms of what 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 operating system. This may be a guest OS under the control of a virtual machine monitor, or a standalone OS available as part of a multiboot configuration. Eash OS views its segment as a separate disk, with separate filesystems 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 OSes.

For a multiboot system, using SwitchBlade is simple. Isolation is maintained as the the OS does not know about any other storage existing on the disk beyond itself. For virtualized operation, we provide better guarantees of isolation between running guest virtual machines 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 itself compromised, only the segments exposed to the VMM, which may be only a subset of available segments on the disk, will be exposed to compromise.

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 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 an enhanced bootloader, as it needs to inspect the disk to discover which segments contain OSes. Second, the boot segment is almost always kept in a read-only state to protect the integrity of the boot code, ensuring that the system can be booted into a safe state. Of course, trusting the code on the boot segment is highly important; advances in proving that a system is rooted in a trustworthy installation can be used here to strengthen these guarantees [12].

Exporting the boot segment allows SwitchBlade to provide integrity guarantees for its contents. Once

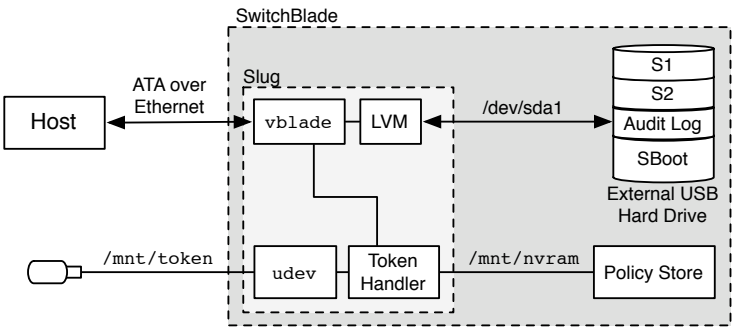

Figure 4: Details of the SwitchBlade prototype. The token handler parses the contents of the USB token, mounted by `udev`. It then invokes `vblade` to export the logical volumes allowed by the token. The audit log is also managed as a logical volume that is exported read-only by `vblade`, but directly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

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 tamperproof 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 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 (TPM) is installed in the disk for attestation to the host system or an active token. Note that if we consider the physical componentry 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, obtaining and modifying the firmware of a commodity disk is infeasible. The technical challenges would be considerable, but the legal difficulties make the process infeasible. Therefore, we made our prototype using a modified network attached storage device, the Linksys NSLU2 Network Storage Link, or "slug", as it's commonly known. 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 ATA or SCSI bus. It's important to note, however, that our prototype would work just as well with ATA or SCSI if the interfaces on the disk were accessible to us.

In order to minimize differences between our prototype's command interface and that of a real disk, we use the ATA over Ethernet (AoE) protocol between the disk and host. As the name implies, AoE sends ATA commands 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. In the case of the prototype SwitchBlade, 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 used by the target to demultiplex commands to each device.

The SwitchBlade prototype, shown in Figure 4, consists of two discrete physical components: an external USB hard disk which provides the actual storage, and the slug, which sits between the host and disk that acts as the policy store and enforcement mechanism. 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 (LVM). The segments are specified as logical volumes (LVs) over the single physical volume (PV) comprising 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.

We were able to easily use the implementation of segments to create the tamperproof 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 tampering to the boot segment, 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 to its local reading the contents of, for example `/mnt/audit_log`.

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.

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

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. 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 filesystem. For Windows XP, the additional steps involved installing the AoE driver, and then using the open-source

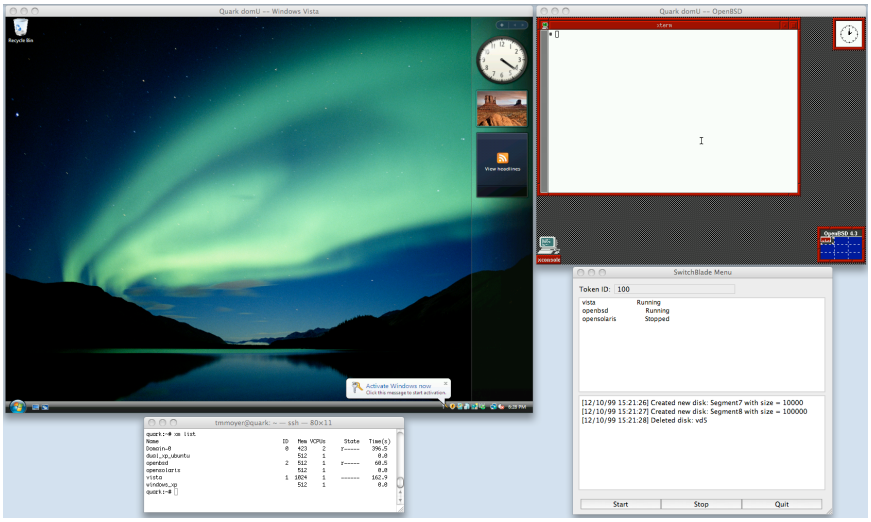

Figure 5: The above screen capture depicts Windows Vista and OpenBSD 4.3 running in virtual machines. The terminal window is showing the VMs that Xen is configured to run, which in this case is five different VMs. The menu interface is shown in the lower right. This interface only shows VMs that can 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.

`gPXE` software to boot over AoE. 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. 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 filesystem 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.

Virtualized Operation

To implement virtualized mode, we use the Xen [1] 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 filesystem. 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 5. 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.

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 consider a non-exhaustive list of attacks that are indicative of how systems may be attacked and the defenses that we provide.

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 FS 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.

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.

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.

Performance

We used `openssl` to perform cryptographic operations on the slug in order to determine performance bottlenecks. Our boot segment, which contained a Xen hypervisor and Debian dom0 kernel, was 580 MB 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 size of the boot segment, or improving the cryptographic hardware available to the SwitchBlade. The former case is possible by using a more minimal boot system, such as VMWare Server ESXi [14], 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, a small amount of time compared to the length of a system reboot. 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 [8].

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 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.

In addition, 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 [10].

Conclusion

Our work on SwitchBlade, and on other problems related to storage security, are ongoing. Readers interested in more details about SwitchBlade can read our extended technical report on the project [5]. One of the intriguing questions that we discovered through our investigation was the need for a mechanism to measure the boot segment, where a virtual machine monitor would reside. Some of our recent work [6] has shown how a secure boot mechanism can be rooted in storage with only a TPM necessary in the host system. In this manner, we can provide the trusted VMM postulated by work such as Terra [7].

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

References

[1] 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 *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, Oct. 2003.

[2] D. Bell and L. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. Technical Report M74-244, MITRE Corporation, Bedford, MA, 1973.

[3] K. Butler, S. McLaughlin, and P. McDaniel. Non-Volatile Memory and Disks: Avenues for Policy Architectures. In *Proceedings of the 1st ACM Computer Security Architectures Workshop (CSAW'07)*, Fairfax, VA, USA, Nov. 2007.

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

[5] K. Butler, S. McLaughlin, T. Moyer, P. McDaniel, and T. Jaeger. SwitchBlade: Policy-Driven Disk Segmentation. Technical Report NAS-TR-0098-2008, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, Nov. 2008.

[6] 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, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, Apr. 2009.

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

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

[9] V. Marala. *The Build Master: Microsoft's Software Configuration Management Best Practices*. Addison-Wesley Microsoft Technology Series. Addison-Wesley Professional, 2005.

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

[11] A. G. Pennington, J. D. Strunk, J. L. Griffin, C. A. N. Soules, G. R. Goodson, and G. R. Ganger. Storage-based Intrusion Detection: Watching storage activity for suspicious behavior. In *Proceedings of the 12th USENIX Security Symposium*, Washington, DC, USA, Aug. 2003.

[12] L. St. Clair, J. Schiffman, T. Jaeger, and P. McDaniel. Establishing and Sustaining System Integrity via Root of Trust Installation. In *Proceedings of the 23rd Annual Comptuer Security Applications Conference (ACSAC 2007)*, Miami, FL, USA, Dec. 2007.

[13] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-Securing Storage: Protecting Data in Compromised Systems. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI'00)*, San Diego, CA, USA, Oct. 2000.

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