U Can't Touch This: Block-Level Protection for Portable Storage

Kevin R.B. Butler

Systems and Internet Infrastructure Security Lab
Pennsylvania State University, University Park, PA, USA
butler@cse.psu.edu

Petros Efstathopoulos

Symantec Research Labs
Symantec Corporation, Culver City, CA, USA
petros_efstathopoulos@symantec.com

ABSTRACT

Advancements in portable storage have made it increasingly likely for users to carry large amounts of data with them, attaching their devices to multiple computers and transferring data between systems. This model poses new challenges for preserving data isolation policies, particularly when traditional information flow preserving operating systems no longer have control over the portable storage medium.

Using secure disks and principles of label persistence from the Asbestos operating system, we propose mechanisms to address these concerns, by making the drive responsible for enforcing data isolation at the block level, and preventing block sharing between hosts that are not considered equally trusted. We describe models of partial protection and full disk labeling, and corresponding operations and preconditions necessary for the OS-disk interaction to occur. This model poses many new system design challenges and can lead to interesting new security mechanisms.

1 INTRODUCTION

Portable storage media, such as external hard drives and USB "flash drives", have become very popular due to their convenient size and weight, good performance, and decreasing cost per GB. However, the popularity of portable storage has raised many security and privacy issues, as evidenced by recent incidents [6, 15] where sensitive data (e.g., classified military or trade secrets) were stored on misplaced or stolen portable media. Current mechanisms for protecting information leakage, due to storage falling into the wrong hands, include full-disk encryption (FDE) for on-disk data protection and authentication mechanisms such as biometrics or a password supplied can be used for access control.

These protections, however, are based on two suppositions: the host being attached to can be trusted not to compromise or leak data from the disk, and a binary "all-or-nothing" approach to releasing access to disk information is adequate. Consider another case, where portable drives are used in a secure environment enforcing an information flow control (IFC) policy such as multilevel security (MLS). While a secure workstation may protect data by ensuring security primitives (e.g., *labels* [3]) are used to implement the desired policy, we are in a conundrum when it comes to guaranteeing the security of that portable data. As portable devices continue to increase their storage capacities, the probabilities of users having just one device with them for all of their storage needs increase commensurately. This will lead to situations where users will want to access their data on host platforms that may not be as well-protected as their secure workstations. For example, a user might connect her portable storage device to a lower-integrity host such as her laptop, or even share it with a user whose system is of unknown integrity.

These latter cases demonstrate how rigid binary access breaks down. Either the files are completely accessible and open to potentially malicious hosts, or all files—including "unclassified", low-secrecy files—become unavailable, hampering usability of the portable drive. If, for example, the user's laptop was trusted to handle certain types of low-secrecy data, it should be able to have access to them. We wish to protect, however, against systems that could be using a flawed or compromised storage stack, which misinterprets or intentionally disregards file system policy primitives.

This paper argues that the best way to protect the secrecy and integrity of data stored on portable media is to re-examine the relationship between the host and storage, and shift from a model of the host being wholly responsible for security policy enforcement to one where the disk can perform policy checks at the block level. To achieve this goal, we first leverage some of the unique characteristics of *Asbestos labels* [5], presented in Section 2. The decentralized nature of Asbestos labels, and the ability to serialize and store them on disk, makes the file system label-aware and allows for self-contained policies to be stored on stable storage. Combining these serialized, self-contained policy descriptions with *autonomously secure disks*, presented in Section 3, we can improve the security of data access on portable disks. In particular, we describe how we can use secure bootstrapping to identify the host, determine its level of integrity and trust, and recover the privileges associated with it—stored on disk during the initial "pairing" process. In Section 4, we describe an architecture for enforcing data isolation at the disk block level. We consider mechanisms for protecting policy-critical files, as well as mechanisms for full disk block labeling that would allow us to enforce data protections regardless of the host integrity level. Our primary contribution is thus *a demonstration of how secure disks can work in concert with operating system primitives to support policy mechanisms and improve the security of portable stor-*

age. We discuss challenges related to our design, and in Section 5 consider related work, before concluding in Section 6.

2 ASBESTOS LABEL PERSISTENCE

The Asbestos operating system [5] aims to improve security by containing the effects of application bugs. An Asbestos label is a function mapping *tags* and *levels*: every tag in the system (represented by a unique, opaque identifier), is associated with one of five possible Asbestos label level values—each corresponding to a different privilege, integrity and secrecy level.

Asbestos uses a “split label” design for processes: the *tracking label* keeps track of all contamination and privilege acquired by the process, while the *clearance label* tracks the level of contamination that the process is cleared to receive with respect to each tag.[1] Using this split label model, Asbestos labels can implement *decentralized information flow control* (DIFC). By modulating these labels directly, or through a high-level policy description language [4], one can implement a wide variety of application-defined, kernel-enforced security policies.

Asbestos labels can be stored persistently on the file system. Similar to processes, files carry two labels: a *file tracking label* and a *file clearance label*, representing the contamination level acquired when reading the file and the privilege necessary to modify it, respectively. All file system labels are immutable and are set at file creation time.[2]

The *pickle* primitive [22] allows applications to serialize and store privilege on the file system. Any runtime Asbestos tag can be “pickled” and stored on a special *pickle file*, along with a privilege level and a key. A special “unpickle” operation allows privilege to be recovered for that tag (e.g., after application restart) if various constraints are satisfied. All pickle files can be unpickled at the least privileged (i.e. most restrictive) level by anyone. Unpickling at more privileged levels is controlled through labels on pickle files themselves, as well as access control checks (i.e., keys associated with each pickle file). With the pickle mechanism in place, all file system labels can be described in terms of pickled tags. Therefore, controlling access to pickle files is essential for the security and integrity of the file system, since pickles are the key to controlling file system labels and serialized, persistently stored privilege.

3 SECURE DISKS AND LABELS

In order to address some of the security challenges related to portable storage, we need a means of ensuring that certain policy decisions can be made securely and independent of the operating system. By making data partially or fully securable below the OS layer, we attempt to address cases where the portable disk is connected to multiple systems with different levels of trust.

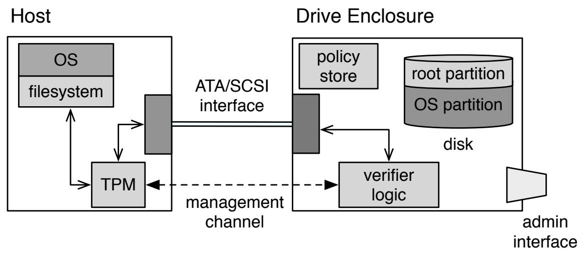

Figure 1: Overview of the Firma architecture.

We have proposed *autonomously secure disks* (ASDs) [2] that are based on the increasing capabilities found in modern disks, such as non-volatile memory and cryptographic processors. These disks form a basis for our investigations and, in conjunction with labeling, provide a means for protecting information flow primitives on disk, and potentially providing the basis for (partial or full) disk-level information flow control.

3.1 Assumptions and Security Goals

Using the advanced capabilities of ASDs, we seek to address certain security concerns related to portable disks. Specifically, we are guided by the following goals and assumptions:

- We assume that the portable drive is equipped with a tamper-proof administrative interface, that can be accessed securely so as to perform certain privileged administrative tasks (e.g. firmware/software upgrade). It could be reliant on physical security, such as requiring the use of a hardware key to access the interface, or policy on what machines are considered administrative could be set within the ASD and upon verification of the administrative machine’s identity, it would have the capability to perform privileged operations. A full discussion of the operational details of this mechanism is beyond the scope of this study.
- We assume that data stored on the portable drive are labeled. In particular, we assume that the file system is using the Asbestos label persistence mechanism.
- Portable disks can be connected to many different hosts. We want to provide data isolation between hosts, by ensuring that information generated from one host is by default not accessible to any other host the drive may be connected to, *unless* it was explicitly stored as “unprotected” (or “unclassified”) data.
- We first assume that all hosts the drive is connected to are attempting to access data through the labeled file system. Attempting to protect against malicious hosts who try to access the raw disk blocks poses additional challenges and is discussed separately in Section 4.2.2.

3.2 Bootstrapping and Integrity Checks

The *Firma* architecture [2] provides a disk-based secure boot mechanism that assures the integrity state of the host system

[1]Alternatively, one can think of the clearance label as the lowest acceptable integrity level incoming information must be at.

[2]Consequently, declassifying data out of a file requires a privileged process to copy the data to a new, “uncontaminated” file.

attached to the disk. In a secure boot model, every stage of the boot process must be validated in order for the boot process to continue. Because with Firma our mechanism emanates from the storage itself, there are increased benefits as no information on the disk will be available to the rest of the system unless the stages are correct. An overview of the architecture is shown in Figure 1.

Validating the integrity state of the host system that the disk connects to requires the ability of the host to provide proofs of its integrity. We rely on the host system containing a Trusted Platform Module (TPM) [20], a commodity tamper-evident chip that provides some non-volatile storage and cryptographic functionality, found in virtually all modern computers. The TPM can be used to generate a *measurement list* (*ML*), to be compared against a list stored on the disk. The stages of the boot process that are measured are the hardware (through a core root of trust measurement), the system BIOS, the bootloader, and the operating system and associated drivers. Firma makes use of the Linux Integrity Measurement Architecture (IMA) [16] for attestations, though other alternatives are also possible. If every measurement stage is validated by the disk, the disk will be in a usable state to the system.

There are numerous methods for the disk to obtain measurements from a remote system. The easiest method would be if the host supported a *dynamic root of trust* for measurement, which does not rely on knowledge of the underlying hardware configuration of the system. The OSLO bootloader [8] provides this functionality, which is contingent on the host's processor supporting special virtualization extensions (e.g., SVM extensions on AMD processors). Because the code for loading the bootloader and operating system resides on the disk, the disk can independently compute the necessary list of measurements and will then be able to compare this list to the measurements received from the host in a hardware-agnostic fashion. Another method uses a trusted third party that divulges the list to the disk prior to its interaction with the system. This could be a trusted web server or a similar infrastructure. Alternately, if the portable disk is a boot drive, it can measure the system itself by being placed in *measurement mode*. In this mode, the boot process is run on the host system with the measurements recorded at each stage, and the final list given to the disk, which uses these as the basis for a subsequent secure boot. This solution is best used when the host system has not been previously used, to mitigate the potential for malware on the system.

Once the system is operational, we can check the system's integrity state through a variety of runtime integrity mechanisms, which are unspecified by Firma. With Firma, we did not specify a particular mechanism for runtime integrity, but many exist; for example, if the disk runs a virtual machine monitor then a dynamic detector such as Patagonix [10] could be used; the use of the Linux Kernel Integrity Monitor (LKIM) [12] is also possible. Periodic attestations of the integrity state can be requested from the host, which can deliver these over a secure disk channel, such as the trusted send and receive commands proposed in the SCSI and ATA specifications.

3.3 Secure Pairing

Using the methods for secure bootstrapping and integrity checking described above, we can identify a host and verify its integrity state. By leveraging the non-volatile memory and increased computing ability found in secure disks, we can design a secure portable disk able to keep track of all systems it has successfully paired with.

After it has been paired with a host, the disk stores a unique identifier for it. A means of guaranteeing the uniqueness of the host system being paired is to use the *endorsement key* (*EK*) of the host's TPM, which is a public-private RSA key pair installed in the TPM at the time of manufacture. The *EK* does not change during the lifetime of the TPM. Normally the *EK* is not revealed to the outside world because knowledge of it can compromise the privacy of the TPM, since it is often used in the context of remote attestation (i.e., identify that a TPM is in use, but not which one specifically). However, in this scenario, it is exactly this property of globally unique identification that allows us to ensure that we are pairing the correct policy with the correct machine. Because we store the public key associated with the *EK*, there are no issues of compromising the secrets on the TPM.

The public *EK* and a corresponding measurement list can be stored in the ASD's non-volatile storage, and this list—comprised of all machines the disk has paired with in the past—is consulted every time the disk is connected to a host. If the host is not recognized, it is considered new, and, after security measurements have been performed, its *EK* and *ML* are inserted to the list of known hosts.

4 PROTECTION MECHANISMS

4.1 Pickle File Protection

Information stored on disk by a system using Asbestos labels may or may not carry file labels protecting access to it. In our model, all unlabeled information on the disk is considered "unclassified" and is accessible to any system the portable medium is connected to.

As described in Section 2, access to all labeled data depends on the ability to unpickle the necessary privilege, or, in other words, on the ability to access the relevant pickle files. Therefore, by controlling access to pickle files, one could control the amount of privilege that could be recovered from disk and, consequently, the disk data that may become available. Our proposed mechanism uses the ASD to control access to pickle files, based on the identity of the host the portable disk is connected to.

During the process of pairing with a new host X, the ASD stores along with the host's fingerprint (i.e., $EK(X)$ and $ML(X)$) a pickle access token, PA_X, unique to that host. Access to all pickle files stored on disk by X is controlled at

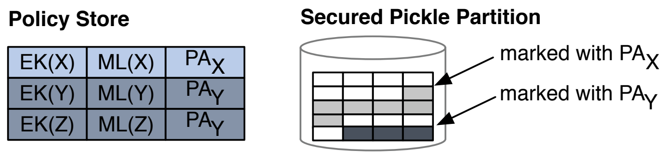

Figure 2: Policy storage and labeling inside the ASD, which stores the endorsement key, measurement list and pickle access token for each paired host. Blocks within the secure pickle partition that correspond to pickle files are marked by the *PA* of the paired host. Note in this example the *equivalence* between paired hosts Y and Z; they share the same access token PA_Y.

the disk block level: when a new pickle file is created by X, the ASD will mark the relevant disk blocks with PA_X. All accesses to pickle file blocks are controlled by *PA* token checks: when attempting to access disk blocks corresponding to pickle files, the ASD checks the current "active" *PA* (i.e. the *PA* of the host the ASD is currently paired with) against the *PA* associated with the disk blocks in question. If the *PA*s do not match, the pickle file blocks access to the pickle file blocks is denied. Notice that this mechanism is independent from higher-level, file system pickle file access control mechanisms.

By using *PA*s, the ASD prevents access at the disk block level, making the pickle file inaccessible from other hosts—even if the user is able to present the necessary credentials for an unpickle operation. Therefore, by controlling access to pickle files created by each host, we are able to control the amount of privilege that can be gained through unpickle operations and limit each host to its own separate, isolated view of the file system.

The pickle access mechanism assumes that the ASD will be able to identify pickle blocks and mark then accordingly. This can be achieved by storing all pickle files in a secure area on the disk, an operation feasible for disks supporting the Opal trusted storage specification [21]. Alternately, the pickle operation can issue an $ioctl()$ to notify the disk. Both solutions rely on the correct implementation of the pickle operation, which is one of the things that can be measured during the integrity measurements performed using the TPM.

We introduce the idea of *equivalence* between two hosts, which can be specified as a policy parameter through the disk's administrative interface. By making two hosts equivalent, we instruct the ASD to operate in exactly the same way (i.e., apply the same policy) when either of two hosts' fingerprint is detected. Host equivalence is clearly marked on the portable disk's list of known hosts. Note that equivalent hosts share the same *PA* and, therefore, have the identical access rights to pickle files. This would allow file sharing between equally trusted machines, e.g., two secure workstations in the same environment.

4.2 A Step Further: Full Labeling

Using the *PA* mechanism, the ASD can implement complete pickle file isolation, and the notion of host equivalence can enable full file sharing between hosts: *all* pickle files created by X are inaccessible to hosts not equivalent to X. This level of protection is a significant improvement over the current situation, and may be adequate for most cases.

However, the mechanism may be too coarse-grained for certain scenarios: we may want two hosts to share access to some, but not all, disk blocks. Moreover, the sets of shared disk blocks between different pairs of hosts may differ: host X may be sharing one set of its disk blocks with host Y and another (overlapping or not) set with host Z.

Implementing such shared sets of disk blocks could be achieved through the use of multiple *PA*s per host, each representing a different *category* of trust between hosts, as shown in Figure 2. The notion of implementing fine-grained policies using such dynamic categories, is very analogous to Asbestos labels themselves: In essence, this block sharing behavior would require the equivalent of Asbestos labels at the block level, which would allow for the definition of such sharing, as well as many other, disk block access policies.

Moving to a full block labeling mechanism enforced at the disk level would also result in a significant change of the security model: disk block protection would no longer be transparent and mandatory, but user-visible, user-controlled and discretionary. Users would be able to instruct the drive to create new categories by generating the disk equivalent of Asbestos tags—which we call *dtags*—used to label the disk blocks belonging to that category.

Although this mechanism may seem redundant in the presence of a trusted labeled file system, its merits become apparent when one considers how the portable drive would operate in a less trusted or friendly environment: disk block access policies are defined and enforced within the portable drive itself. This self-contained system is able to protect data even in when the host is not co-operating, and could be used to protect from malicious users, or even lost or stolen drives. Essentially, the disk can act as a defense-in-depth mechanism by using the policy received from the host as a type of anomaly detection at the block layer if the request received from the host appears to be inconsistent with what has been laid out by the policy received from it. This could trigger a runtime attestation to ensure that the host is in a good state before fulfilling the request.

4.2.1 *Labeling Pickle Blocks*

By replacing *PA*s with labels, we could implement more complex policies regarding how the ASD would restrict access to pickle disk blocks. Once could envision a mechanism operating in the following manner: When an ASD is paired with a new host X, it generates a new dtag x_0 for that host. Unless the user requests the creation of a new dtag for X, x_0 will be used to provide the same security that the *PA* mechanism would: all pickle blocks are labeled with x_0 and accessing them would require holding x_0 privilege. If the user requested the creation of a new category (by using the relevant $ioctl()$ call), the ASD would create a new dtag x_1, associate it with $EK(X)$, and make x_1 the "active" dtag—i.e. the one used to label the disk blocks of all pickle files created by X from that point on.

Along with each host's EK the ASD would store a list of dtags the host holds privilege over. When pairing with a new host Y, a user with secure access to the ASD administrative interface could grant Y privilege with respect to dtags other than y_0, and grant y_0 privilege to already existing hosts.

4.2.2 Full Block Labeling

By labeling pickle blocks we still use block labels only to determine access control rights over pickle disk blocks, essentially implementing a capability system. Consequently, we do not need to use the Asbestos split-label design. Using the full features of Asbestos labels (multiple dtag levels and split-label design) would enable block-level information flow tracking for this mechanism. However, implementing dtag management logic and block labeling capabilities within the ASD brings us a step closer to a fully labeled disk able to support disk-enforced, application-level policies. Apart from labeling pickle file disk blocks for host access control, the disk also needs to support labeling of all disk data blocks, based on check-pointed application-level Asbestos label policies. Either periodically or on-demand, file system labels would be translated to block access restrictions, and pushed to the ASD's secure storage using trusted SCSI or ATA commands.

While this disk-wide block level policy enforcement entails some complexity within the ASD (requiring label bookkeeping logic, label checking functions etc.), proposals such as the rootkit-resistant disk prototype [1] show that the administrative overhead of such operations can be very low, on the order of 1%. Additionally, this enforcement model provides some unique benefits. By downloading application level policies to the ASD, in combination with secure pairing and pickle disk block protection, the ASD would be able to provide a complete, self-contained data security solution. A self-contained, policy-enforcing portable ASD would be able to provide protection against loss or theft: the drive would refuse to grant access to pickle files when paired with unknown hosts. If the pickle mechanism is bypassed and a malicious user attempts to access raw disk blocks, the ASD would still refuse access, due to failed disk block label checks. Additionally, full disk block labeling would be able to protect against compromised or low-integrity systems—as detected by the security measurement capabilities of the ASD—by performing label checks within the disk based on the last high-integrity label policy downloaded to it (and not having to rely on the integrity of the operating system).

Discussion Labeling individual blocks using application-defined policies would also allow us to implement interesting file sharing and locking semantics. By applying different labels to blocks belonging to the same file, one could create potentially usefully locking mechanisms, as well as implement file access control policies at a fine granularity. Applications that manipulate large files internally to implement their own storage logic (e.g. databases storing row data inside a file) could benefit from such a mechanism, but block labels are expected to be immutable and applications would need take this into consideration. Additionally, making certain disk blocks inaccessible might cause anomalies: making file system metadata blocks accessible to a process would reveal a lot of information about the file, even if some of the actual data blocks are not accessible by that process.

5 RELATED WORK

Self-securing storage [19] is an effort to collocate security metadata with the disk, but relies on an object-based paradigm. We are uncertain when object-based storage will appear and consider strictly a block-based approach that does not require semantic awareness [18] or type differentiation [17] from the disk. BitLocker [13] performs volume encryption based on a TPM but does not protect against post-boot OS compromise. There is also no support for multiple pairing as our solution provides. Protection mechanisms for storage are numerous; one of the more intriguing schemes is Plutus [7], which provides lockbox-based encryption; SNAD [14] provides a similar mechanism. In these systems, a symmetric key is guarded using public-key encryption; with Plutus, for example, there are reader and writer keys. A problem with this approach is that while users can be authorized through giving out public keys, there is no method of ensuring the security of the underlying platform being attached to.

Persistent labeling has been addressed by both research operating systems, such as Asbestos, HiStar [23] and Flume [9], as well as commercial implementation such as SELinux [11]. Each of these systems addresses label policy persistence using its own mechanism, including Asbestos pickle files, HiStar's single-level store, Flume's use of its reference monitor for persistence, and SELinux's file security context labels. However, the problems arising from portable storage have not been addressed by any of these systems, whose mechanisms rely on assumptions about the correctness and integrity of the relevant software mechanisms (storage stack, file system or reference monitor implementation etc) to ensure data safety.

6 CONCLUSION

This paper has proposed an architecture for block level policy enforcement at the disk layer, to provide multiple levels of security for the potentially many systems that can attach to portable storage. This allows users to access information on their disk without exposing sensitive information to potentially malicious systems or those that may become compromised during use. We use Asbestos labels in conjunction with the Firma secure boot mechanism to provide guarantees of host integrity and describe how we can ensure that the appropriate security policies are maintained depending on the host connected to. The collaborative security between the host system and the disk provides an interesting new security model and challenging issues as we further explore this problem space and implement prototypes demonstrating these new functionalities.

REFERENCES

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

[2] 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, University Park, PA, Apr. 2009.

[3] Department of Defense. *Trusted Computer System Evaluation Criteria (Orange Book)*, Dec. 1985. DoD 5200.28-STD.

[4] P. Efstathopoulos and E. Kohler. Manageable fine-grained information flow. In *Proc. the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Galsgow, UK, Apr. 2008.

[5] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris. Labels and event processes in the Asbestos operating system. In *Proc. 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, Brighton, England, Oct. 2005.

[6] Financial Times. Sensitive RAF data stolen, May 2009. `http://www.ft.com/cms/s/0/86b1ec88-498d-11de-9e19-00144feabdc0.html`.

[7] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, Apr. 2003.

[8] B. Kauer. OSLO: Improving the Security of Trusted Computing. In *Proc. of the 16th USENIX Security Symposium*, Boston, MA, Aug. 2007.

[9] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard OS abstractions. In *Proc. the 21th ACM Symposium on Operating Systems Principles (SOSP '07)*, Stevenson, Washington, Oct. 2007.

[10] L. Litty, H. A. Lagar-Cavilla, and D. Lie. Hypervisor Support for Identifying Covertly Executing Binaries. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, Aug. 2008. USENIX Association.

[11] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. In *Proc. 2001 USENIX Annual Technical Conference—FREENIX Track*, pages 29–40, June 2001.

[12] P. A. Loscocco, P. W. Wilson, J. A. Pendergrass, and C. D. McDonell. Linux kernel integrity measurement using contextual inspection. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 21–29, 2007.

[13] Microsoft. BitLocker Drive Encryption Technical Overview. `http://technet.microsoft.com/en-us/library/cc732774.aspx`.

[14] E. L. Miller, W. E. Freeman, D. D. E. Long, and B. C. Reed. Strong Security for Network-Attached Storage. In *Proceedings of USENIX FAST'02*, Monterey, CA, USA, Jan. 2002.

[15] National Archives and Records Administration. Missing Clinton Administration Hard Drive, May 2009. `http://www.archives.gov/news/clinton-hard-drive-faq-2009-5-20.pdf`.

[16] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proc. 13th USENIX Security Symp.*, San Diego, CA, Aug. 2004.

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

[18] 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, USA, Apr. 2003.

[19] 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 Symp. on Operating Systems Design and Implementation (OSDI'00)*, San Diego, CA, Oct. 2000.

[20] TCG. *TPM Main: Part 1 - Design Principles*. Specification Version 1.2, Level 2 Revision 103. TCG, July 2007.

[21] TCG. *TCG Storage Security Subsystem Class: Opal*. Specification Version 1.0, Revision 1.0. Trusted Computing Group, Jan. 2009.

[22] S. VanDeBogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris, and D. Mazières. Labels and event processes in the Asbestos operating system. *ACM Transactions on Computer Systems*, 25(4):11:1–11:43, Nov. 2007.

[23] N. B. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in HiStar. In *Proc. 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, Seattle, WA, Nov. 2006.