Protecting Portable Storage with Host Validation

Kevin Butler
Department of Computer and Information Science
University of Oregon, Eugene, OR
butler@cs.uoregon.edu

Stephen McLaughlin & Patrick McDaniel
Systems and Internet Infrastructure Security Laboratory (SIIS)
Pennsylvania State University,
University Park, PA
{smclaugh,mcdaniel}@cse.psu.edu

ABSTRACT

Portable storage devices, such as key-chain USB devices, are ubiquitous and used everywhere; users repeatedly use the same storage device in open computer laboratories, Internet cafes, and on office and home computers. Consequently, they are the target of malware that exploit the data present or use them as a means to propagate malicious software.We present the Kells mobile storage system, which limits untrusted or unknown systems from accessing sensitive data by continuously validating the accessing host's integrity state. We explore the design and operation of Kells, and implement a proof-of-concept USB 2.0 storage device on experimental hardware. Our experiments indicate nominal overheads associated with host validation, with a worst-case throughput overhead of 1.22% for reads and 2.78% for writes.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection

General Terms: Security

Keywords: storage, security, validation

1. INTRODUCTION

Recent advances in materials and memory systems have irreversibly changed the storage landscape. Small form factor portable storage devices housing previously unimaginable capacities are now commonplace today–supporting sizes up to 256 GB[1]. Such devices change how we store our data; single keychain devices can simultaneously hold decades of personal email, millions of documents, and thousands of songs. These devices are convenient, allowing us to carry artifacts of our digital lives wherever we go.

The casual use of mobile storage has a darker implication. Users plugging their storage devices into untrusted hosts are subject to data loss or corruption. Compromised hosts have unfettered access to the storage plugged into their interfaces, and therefore have free rein to extract or modify its contents. Users face this risk when accessing a friend's computer, using a hotel's business office, in university computer laboratories, or in Internet cafes. The risks here are real. Much like the floppy disk-borne viruses in the 1980's and 90's, malware like Conficker [7] exploit mobile storage to propagate malicious code. The compromise of hosts throughout military networks, due to malware propagated by rogue portable storage devices, has already led to a ban of their use by the US Department of Defense [11]. The underlying security problem is age-old: users cannot ascertain how secure the computer they are using to access their data is. Thus, all of their information is potentially at risk if the system is compromised. The challenge is this: *How can we verify that the computer we are attaching our portable storage to is safe to use?*

Storage security has recently become an active area of investigation. Solutions such as full-disk encryption [9] and Microsoft's BitLocker to Go [5] require that the user supply a secret to access stored data. This addresses the problem of device loss or theft, but does not aid the user when the host to which it is to be attached is itself untrustworthy. Conversely, BitLocker (for fixed disks) uses a trusted platform module (TPM) [13] to seal a disk partition to the integrity state of the host, thereby ensuring that the data is safeguarded from compromised hosts. This is not viable for mobile storage, as data is bound to the single physical host. Additionally, BitLocker does not assess the integrity of the host at any time after the data is mounted. In another effort, the Trusted Computing Group (TCG) has considered methods of authenticating storage to the host through the Opal protocol [14] such as pre-boot authentication and range encryption and locking for access control. These services may act in a complementary manner to our solution for protecting mobile storage from potentially compromised hosts.

This abstract introduces *Kells*[2], an intelligent USB storage device that validates host integrity prior to allowing read/write access to its contents, and thereafter only if the host can provide ongoing evidence of its integrity state. When initially plugged into an untrusted device, Kells performs a series of *attestations* with trusted hardware on the host. These exchanges are repeated periodically to ensure the integrity state of the system remains stable. Kells uses integrity measurement to ascertain the state of the system and the software running on it at boot time in order to determine whether it presents a safe platform for exposing data. If the platform is deemed to be trustworthy then a trusted storage partition will be exposed to the user; otherwise, depending on a configurable policy, the device will either mount only a "public" partition with untrusted files exposed or will not mount at all. If at any time the device cannot determine the host's integrity state or the state becomes undesirable, the protected partition becomes inaccessible. Kells can thus ensure the integrity of data on a trusted storage partition by ensuring that data can only be written to it from high-integrity, uncompromised systems. Our design uses the commodity Trusted Platform Module (TPM) found in the majority of modern computers as our source for trusted hardware, and our implementation and analysis use it as a component. *We note, however, that it is not in-*

[1]http://www.kingston.com/ukroot/flash/dt300.asp

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.
ACM 978-1-4503-0244-9/10/10.

[2]Named for the Book of Kells, which is traceable to having resided at the Abbey of Kells, Ireland, in the 12th century thanks to information on land charters written into it.

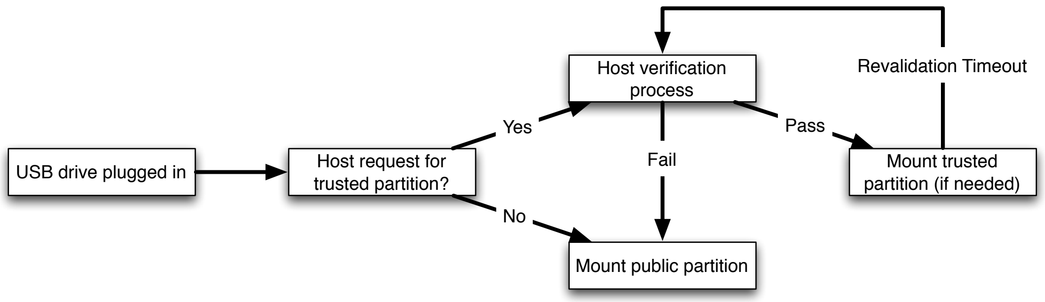
USB drive plugged in
Host request for trusted partition?
Yes
No
Host verification process
Fail
Pass
Mount public partition
Mount trusted partition (if needed)
Revalidation Timeout

Figure 1: Overview of the Kells system operation. Attestations of system state are required to be received successfully by the device in order for the integrity of the host to be proved, a necessary precondition for allowing data to be available to the host.

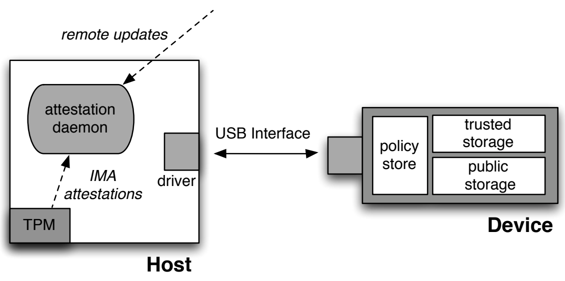

Figure 2: Overview of the Kells architecture.

tegral to the design: any host integrity measurement solution (e.g., Pioneer [10]) can be used.

Kells diverges substantially from past attempts at securing fixed and mobile storage. In using the mobile storage device as an autonomous trusted computing base (TCB), we extend the notion of self-protecting storage [2, 6] to encompass a system that actively vets the devices that make use of it. In so doing, we provide a path to enjoying the convenience of now-ubiquitous portable storage in a safe manner. Our contributions are as follows:

- We identify system designs and protocols that support portable storage device validation of an untrusted host's initial and ongoing integrity state. To our knowledge, this is the first use of such a system by a dedicated portable storage device.
- We describe and benchmark our proof of concept Kells system built on a DevKit 8000 board running embedded Linux and connected to a modified Linux host. We empirically evaluate the performance of the Kells device. These experiments indicate that the overheads associated with host validation are minimal, showing a worst case throughput overhead of 1.22% for read operations and 2.78% for writes.

In our full technical report [1], we also reason about the security properties of Kells using the LS^2 logic [3] and prove that the storage device can only be accessed by hosts whose integrity state is valid (within a security parameter Δ_t).

The remainder of this abstract describes the operation of Kells as well as its architecture, implementation, and evalation.

2. DESIGN AND IMPLEMENTATION

The Kells architecture is shown in Figure 2. We modify three major components of the system in order to use Kells: the interface between the host and the device, the storage device itself, and the host's operating system.

2.1 USB Interface

We designed Kells to use USB, as it is the most common external computer interface. We built Kells by modifying the USB mass storage device stack. Within operating systems that support USB, such as Linux, the number and functionality of supported devices is large and diverse. To support devices that do not conform to the USB specificaton, Linux contains a repository for these devices and sets flags when they to modify the host behavior. A host that recognizes a Kells device will set these flags and send special *command transfer* operations to it. When the device receives these commands, it can begin the attestation protocol. If these commands sare not received, the host can access the public partition as a standard mass storage device, oblivious to the trusted protocols and storage.

2.2 Designing the Storage Device

Kells requires the ability to perform policy decisions independent of the host. As a result, logic must execute on these devices, as well as a way of receiving command transfers from the host and to use these for making the correct access decisions.

The basic architecture for the storage device is an extension to the Linux USB gadget stack, along with a user-space daemon in charge of policy decisions and accessing other important information. Within the kernel, we added new functionality allowing Kells to receive control transfers from the host. These are exported to user space through the *sysfs* interface, where they are read as strings by the daemon tasked with marshaling this data. When plugged in, the daemon on the device sets a timer and waits to determine whether the host presents the proper credentials.

If the command transfer containing authenticating information from the host is not received within a defined time period, operation on the device defaults to only exposing the public operation. If the device is configured such that the policy does not allow any partitions to be mounted, the device will not present any further information to the host. If the protocol fails, the failure is logged in the storage device's audit log, which is unexposed to the host. If the protocol is successful and the host attests its state to the device, the daemon presents the trusted partition to be mounted.

Within the Kells device is a policy store containing information about every known host, a *measurement database* to compare attestations against, and policy details, such as whether the host is authenticated as an administrative console and whether the host should expose a public partition if the attestation check fails. Optionally, the device can also store information on user credentials supplied directly to the device through methods such as biometrics. Policy can then be configured to allow or disallow the device to be plugged into specific machines.

2.3 Modifications to Host

A host must be capable of recognizing that the Kells device is trusted and sending information to it differs from a standard USB mass storage transaction. We made some small changes to the USB driver, defining a flag in the USB device repository allowing the Kells device to be recognized. Because the host must interact with its trusted hardware and perform some logic, we designed an *attestation daemon* that runs in the host's user space. While it may be desirable to reduce the number of additional programs that need to run, a major design consideration in the kernel is separating policy from mechanism. The attestation daemon both retrieves boot-time attestations using the Linux Integrity Measurement Architecture (IMA) [8] and can act as an interface to any runtime monitoring systems on the host.

3. ATTESTING HOST INTEGRITY

In order for a host connecting to the Kells device to be trustworthy, it must be installed and maintained in a manner that protects its integrity. A way to ensure this is by provisioning a secure kernel and supporting operating system, from which measurements of system integrity can be made and transferred to Kells. The host system maintainer must thus re-measure the system when it is installed or when measurable components are updated. Solutions for ensuring a trusted base installation include the use of a root of trust installer (ROTI) [12], which establishes a system whose integrity can be traced back to the installation media.

Our full technical report describes the attestation protocol in detail. Briefly, we measure the hardware and software using IMA, based on the Kells device being placed in *measurement mode*. Subsequent attestations use this measurement list for validating the system state; this list can also be remotely disseminated. A portion of non-volatile memory within Kells records this information, which includes a unique identity for the host, the list of measurements that are associated with the host (for attestation verification), and policy-specific information, such as whether the host being attached to, should allow administrative access.

We provide a framework for supporting runtime integrity monitoring, but we do not impose constraints on what system is to be used. The runtime monitor can provide information to the storage device as to the state of the system, with responses that represent good and bad system states listed as part of the host policy. For example, if the host system uses the Patagonix system for detecting covertly-executing rootkits [4], it could provide a response to the disk on being queried as to whether the system has any hidden binaries currently executing. Solutions that use the TPM may also be appropriate for runtime monitoring. Our design considers attestations from a runtime monitor to be delivered in a consistent, periodic manner; one may think of them as representing a *security heartbeat*. The period of the heartbeat is fixed by the device and transmitted to the host as part of the device enumeration process, when other parameters are configured.

4. EVALUATION

We performed a series of experiments aimed at characterizing the performance of Kells in realistic environments. All experiments were performed on a Dell Latitude E6400 laptop running Ubuntu 8.04 with the Linux 2.6.28.15 kernel. The laptop TPM performs a single quote in 880 msec. The Kells device was implemented using a DevKit 8000 development board that is largely a clone of the popular BeagleBoard.

Our experiments sought to determine the overhead of read operations. Each test read a single 517 MB file, the size of a large video, from the Kells device. We varied the security parameter Δ_t (the periodicity of the host integrity re-validation) over subsequent experiments, and created a baseline by performing the read test with a unmodified DevKit 8000 USB device and Linux kernel. All statistics are calculated from an average of 5 runs of each test.

Read operation performance was largely unaffected by the validation process, as the host preemptively creates validation quotes and delivers them to the device at or about the time a new one is needed (just prior to a previous attestation becoming stale). Thus, the validation process is mostly hidden by normal read operations. Performance, however, does degrade slightly as the validation process occurs more frequently. At a 1 second reattestation interval, throughput is reduced by only 1.2%, and as little as 0.2% at 10 seconds. This overhead is due largely to overheads associated with receiving and validating the integrity proofs (as large as 100KB).

Writes were performed over a 200 MB file. These are substantially slower on flash devices because of the underlying memory materials and structure. Here again, the write operations were largely unaffected by the presence of host validation, leading to a little less than 3% overhead at a 1 second reattestation inverval and just under 1% at a 10 second interval.

5. REFERENCES

[1] K. Butler, S. McLaughlin, and P. McDaniel. Kells: A Protection Framework for Portable Data. Technical Report NAS-TR-0134-2010, Network and Security Research Center, Pennsylvania State University, June 2010.

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

[3] A. Datta, J. Franklin, D. Garg, and D. Kaynar. A Logic of Secure Systems and its Application to Trusted Computing. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2009.

[4] L. Litty, H. A. Lagar-Cavilla, and D. Lie. Hypervisor Support for Identifying Covertly Executing Binaries. In *Proceedings of the 17th USENIX Security Symposium*, pages 243–258, San Jose, CA, USA, Aug. 2008.

[5] Microsoft. BitLocker and BitLocker to Go. `http://technet.microsoft.com/en-us/windows/dd408739.aspx`, Jan. 2009.

[6] A. G. Pennington, J. D. Strunk, J. L. Griffin, et al. Storage-based Intrusion Detection: Watching storage activity for suspicious behavior. In *Proceedings of the 12th USENIX Security Symposium*, Washington, DC, USA, Aug. 2003.

[7] P. Porras, H. Saidi, and V. Yegneswaran. An Analysis of Conficker's Logic and Rendezvous Points. Technical report, SRI Computer Science Laboratory, Mar. 2009.

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

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

[10] A. Seshadri, M. Luk, E. Shi, et al. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *Proceedings of ACM SOSP*, Brighton, UK, 2005.

[11] N. Shachtman. Under Worm Assault, Military Bans Disks, USB Drives. *Wired*, Nov. 2008.

[12] L. St. Clair, J. Schiffman, T. Jaeger, and P. McDaniel. Establishing and Sustaining System Integrity via Root of Trust Installation. In *ACSAC*, Miami, FL, USA, Dec. 2007.

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

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