Lockdown: A Safe and Practical Environment for Security Applications

Amit Vasudevan, Bryan Parno, Ning Qu, Virgil D. Gligor, Adrian Perrig

July 14, 2009

CMU-CyLab-09-011

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Lockdown: A Safe and Practical Environment for Security Applications

Amit Vasudevan, Bryan Parno, Ning Qu, Virgil D. Gligor, Adrian Perrig
amitvasudevan@acm.org, parno@cmu.edu, quning@cmu.edu, gligor@cmu.edu, perrig@cmu.edu
CyLab, Carnegie Mellon University

Abstract

We describe, build, and evaluate Lockdown, a system that significantly increases the level of security for online transactions, even on a platform infested with malicious code. Lockdown provides the user with a highly-protected, yet also highly-constrained trusted environment for performing online transactions, as well as a high-performance, general-purpose environment for all other (non-security-sensitive) applications. A simple, user-friendly external interface allows the user to securely learn which environment is active and easily switch between them.

We focus on making Lockdown deployable and usable today. Lockdown works with both Windows and Linux, and provides immediate improvements to security-sensitive tasks while imposing, on average, only 3% memory overhead and 2–7% storage overhead on non-security-related tasks.

1 Introduction

Consumers currently use their general-purpose computers to perform many sensitive tasks; they pay bills, fill out tax forms, check account balances, trade stocks, and access medical data. Unfortunately, increasingly sophisticated and ubiquitous attacks undermine the security of these activities. Indeed, the user currently lacks the means to even verify whether a transaction's safety was preserved. Existing security solutions (Section 10) tend to be hard to use and difficult to verify. Many degrade system performance and restrict application choice.

In this work, we aim to preserve the safety of applications that perform online transactions without degrading the performance or generality of the user's non-security-sensitive applications. Obviously, protecting security-sensitive applications that do not require an Internet connection represents a subset of this goal. In this context, we define safety to mean the protection of the security-sensitive application's execution integrity, as well as the secrecy and integrity of the data it handles.

Our key insight is that users tend to perform security-sensitive transactions infrequently. Furthermore, modern hardware, combined with some of our new techniques, allows for rapid and secure switching between two completely different computing environments. In this work, we leverage these observations, along with the assumption that users will remember (and tolerate the need) to activate a "secure mode" before performing security-sensitive transactions.

We present Lockdown, a system for improving the security of sensitive applications, while preserving the flexibility, functionality, and performance of non-security-sensitive applications. Lockdown provides strong isolation between a trusted environment intended for security-sensitive applications, and an untrusted environment that runs all of the user's non-security-related applications. This strong isolation prevents untrusted code from directly attacking trusted applications. It also prevents the trusted applications from receiving maliciously crafted inputs from code in the untrusted environment. To keep the trusted environment pristine, Lockdown only permits known, trusted code to execute. Since this trusted code may still contain bugs, Lockdown ensures that trusted applications can only communicate with trusted sites. This prevents malicious sites from corrupting the applications, and ensures that even if a trusted application is corrupted, it can only leak data to sites the user already trusts with her data.

Designing Lockdown entails several technical challenges. First, we aim to securely split system execution into trusted and untrusted environments and yet preserve the full generality, functionality, and performance of applications in the untrusted environment. Second, Lockdown must securely display the current state of the system (e.g., trusted or untrusted) in a way the user can understand and trust. Finally, Lockdown should be simple to install, use, and maintain.

Lockdown meets these challenges via several novel approaches. First, to create the two environments and switch between them, Lockdown employs a light-weight hypervisor and ACPI support to *partition*, not virtualize, system resources. Our use of *hyper partitioning* and *hyper switching* leads to smaller, simpler code, since we partition existing

resources, instead of trying to add new (or abstract old) functionality. It also increases performance, since it requires less resource interpositioning. Thus, applications in the untrusted environment can run at nearly native speed with full access to system devices. Second, we use a small, external device to communicate the state of the system (i.e, trusted or untrusted) to the user. Thus, the security display is beyond the control of an adversary and cannot be spoofed or manipulated. Finally, the external USB device uses a very simple interface with essentially one bit of input and one bit of output, making it easy to understand.

We have implemented a full prototype of Lockdown for Windows. The Windows implementation has 8,471 lines of code, including the code on the external device. We have also implemented a partial prototype of Lockdown for Linux to verify that hyper-partitioning and hyper-switching, the core concepts driving Lockdown, work with ACPI-compliant operating systems. The primary limitation of Lockdown's approach is that our prototype currently requires 42–46 seconds to switch between the two environments. However, in Section 8.2.3, we describe several potential optimizations that would significantly reduce this switching time.

2 Background

2.1 ACPI Specification

The Advanced Configuration and Power Interface (ACPI) specification [7] is an open standard for unified, OS-centric device configuration and power management that is supported by most modern PCs. The specification defines an ACPI subsystem (BIOS+chipset) and an Operating-System-directed configuration and Power Management (OSPM) subsystem. With an ACPI-compatible OS, applications and device drivers interact with the OSPM code, which in turn interacts with the low-level ACPI subsystem.

The ACPI specification defines four system sleep states which an ACPI-compliant computer system can be in: S1 (power is maintained to all system components, but the CPU stops executing instructions), S2 (the CPU is powered off), S3 (standby), and S4 (hibernation: all of main memory is saved to the hard disk and the system is powered down).

2.2 Trusted Computing

Many commodity computers currently come equipped with a Trusted Platform Module (TPM) [19]. The TPM is a dedicated security chip designed to help establish trust in a computing platform. At a high-level, the TPM can be thought of as possessing a public-private keypair, with the property that the private key is only handled within a secure environment inside the TPM. The TPM also contains Platform Configuration Registers (PCRs) that can be used to record the state of software executing on the platform. The PCRs are append-only, so previous records cannot be eliminated without a reboot.

A platform containing a TPM can generate an *attestation*, which is essentially a signature with the TPM's private key over the values currently held in the PCRs. Given the TPM's corresponding public key, an external verifier can check that the signature is valid and conclude that the PCR values in the attestation represent the software state of the platform.

2.3 AMD's Secure Virtual Machine (SVM)

AMD's SVM extensions provide hardware support for virtualizing unmodified operating systems. Intel's VT and TXT extensions provide similar support. These extensions allow the CPU to execute in two modes: host (for the hypervisor) and guest (for the operating system). The host and guest modes have separate address spaces and CPU registers, and can execute in any of four privilege levels. With SVM, the host can intercept certain instructions, I/O operations, exceptions, interrupts, and CPU MSR accesses. Further, on an intercept the host can either resume the guest or re-inject an event into the guest for processing.

To simplify memory management, SVM provides **Nested Page Tables (NPT)** which separate memory addresses into guest virtual, guest physical, and host physical. The guest virtual addresses are translated to guest physical addresses using guest paging structures. The guest physical addresses are translated into host physical addresses using the NPT. SVM can also prevent devices from using DMA to read and write portions of physical memory. This is enabled through a DMA Exclusion Vector (DEV), which is a bit vector with one bit for each physical page. If the corresponding bit is set, then the memory controller disallows DMA reads and writes to that physical memory page.

Late launch is another capability of SVM that allows an arbitrary piece of code to execute in isolation from everything else on the system except for the CPU, memory, and chipset. The use of late launch and the launched code is securely recorded in the TPM.

3 Problem Definition

3.1 Goals

Lockdown's primary goal is to enable a set of trusted software to communicate with a set of trusted sites while preserving the secrecy and integrity of these applications and the data they handle. To add robustness and defense in depth, we require that even if the trusted software is subverted, it should only be able to leak data to sites that the user already trusts with her data. Finally, we wish to achieve the above goals without modifying any hardware or software the user already employs. In other words, a user should be able to run the same OS (e.g., Windows), launch her favorite browser (e.g., Internet Explorer) and connect to her preferred site (e.g., a banking website) via the Internet in a highly secure manner while maintaining the current level of performance for applications that are not security-sensitive.

3.2 Adversary Model

We assume the adversary can execute arbitrary code within the untrusted environment. The adversary may also monitor and manipulate network traffic to and from the user's machine. However, we assume the adversary does not have physical access to the user's machine.

3.3 Design Space

We briefly discuss alternate points in the design space and contrast them with Lockdown's approach.

Two Computers. In government and military settings, users often employ two physically-distinct computers, one for highly-classified work and the other for less sensitive work. This approach provides considerable security, but it also adds significant costs (hardware and energy). It is not readily portable, and extending it to more than two environments adds even more cost.

Make Everything Trusted. If we have the technology to make some applications secure, why not secure all of them? In practice, adding security often means sacrificing performance and/or generality. With Lockdown, we argue that the user should only make these sacrifices for the limited number of applications that deal with sensitive data, while the much larger set of less sensitive applications should operate as they do today.

Virtualization. With the recent popularity of virtualization, some have suggested using a separate virtual machine (VM) for each security-sensitive task [4, 12]. Virtualization allows rapid switching between different environments. However, to allow device sharing, devices must be virtualized, leading to performance degradation and/or an increased trusted computing base (due to the inclusion of device drivers) [8]. Sharing the CPU, memory, and devices can also lead to side channels that allow malware in one VM to extract information from a security-sensitive VM.

3.4 Assumptions

1. **Trusted Software and Sites.** As we discuss in Section 5.2.1, we assume certain software packages and certain websites can be trusted to not deliberately leak private data.
2. **Lockdown Security.** We assume that Lockdown's code does not contain vulnerabilities. Given Lockdown's small size (8,471 lines), manual audits and formal analysis can validate this assumption.
3. **User Abilities.** We assume the user can be trained to perform security-sensitive operations in the trusted environment. We also assume the user will notice when the Lockdown Verifier turns red and emits a warning noise. Users who do not satisfy this assumption may not benefit from Lockdown's protections. Nonetheless, those who can fulfill these simple requirements will enjoy significantly improved security.
4. **Hardware Support.** We assume the user's computer supports hardware-based Nested Page Tables and contains a TPM chip. Both technologies are rapidly becoming ubiquitous.
5. **Trusted BIOS.** Lockdown uses the BIOS during its installation and to reset devices, so we must assume the BIOS has not been corrupted. Fortunately, most modern BIOSes require signed updates [14], preventing most forms of attack.

4 Architecture

4.1 Overview

At a high level (Figure 1), Lockdown splits system execution into two environments, trusted and untrusted, that execute non-concurrently. This design is based on the belief that the user has a set of tasks (e.g., games, photo editing,

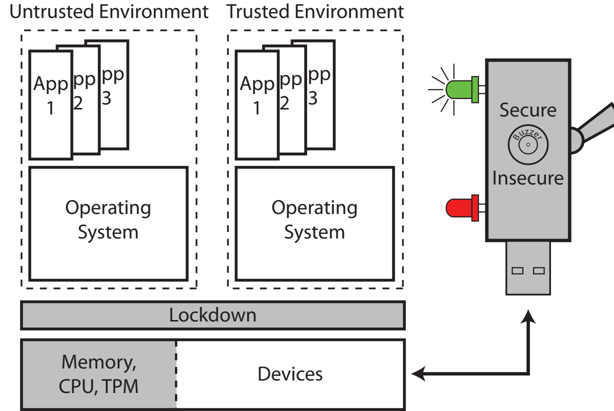


Figure 1: **Lockdown System Architecture.** *Lockdown partitions the platform into two environments; only one environment executes at a time. An external device indicates which environment is active and can be used to toggle between them. The shaded portions represent components that must be trusted.*

browsing for entertainment) that she wants to run with maximum performance, and that she has a set of tasks that are security sensitive (e.g., checking bank accounts, paying bills, making online purchases) which she wants to run with maximum security and which are infrequent and less performance-critical. The performance-sensitive applications run in the untrusted environment, which runs at near-native speed, while security-sensitive applications run in the trusted environment, which is kept pristine and protected by Lockdown.

The Lockdown architecture is based on four core concepts: (i) **hyper-partitioning**: system resources are partitioned as opposed to being virtualized. Among other benefits, this results in greater performance, since it minimizes resource interpositioning, and it eliminates most side-channel attacks possible with virtualization; (ii) **hyper-switching**: device states are saved and restored across environment switches; (iii) **external verification and trusted path**: the state of the system (trusted or untrusted) is communicated to the user in a secure and easy to understand fashion; and (iv) **trusted environment protection**: Lockdown limits code execution in the trusted environment to a small set of trusted applications and ensures that network communication is only permitted with trusted sites.

4.2 Hyper-Partitioning

Since the untrusted environment may be infected with malware, Lockdown must isolate the trusted environment from the untrusted environment. Further, Lockdown must isolate itself from both environments so that its functionality cannot be deliberately or inadvertently modified. To achieve this isolation, Lockdown partitions the CPU in time by only allowing one environment to execute at a time. Memory and device partitioning are explained below.

Using *partitioning* as opposed to *virtualization* has many benefits. First, it allows the untrusted environment to run with full generality at native speed, since it has full access to the CPU and system devices. Second, whenever environments with different trust levels share resources (e.g., when using virtualization), side channels exist that might allow information in the trusted environment to leak to malware in the untrusted environment [9]. Partitioning eliminates most of these side-channels. Finally, partitioning keeps the Lockdown codebase tiny, since it does not require device drivers supporting different devices from various vendors [8]. A small codebase permits formal analysis of Lockdown to rule out potential vulnerabilities.

Memory. Lockdown partitions the available physical memory in the system into three areas: the Lockdown memory region, the untrusted environment's memory region, and the trusted environment's memory region. Lockdown employs Nested Page Tables to restrict each environment to its own memory region. In other words, the NPT for the untrusted environment does not map physical memory pages that belong to the trusted environment and vice versa. Further, it employs hardware-based DMA-protection within each environment to prevent DMA-based access beyond each environment's memory regions.

Devices. With hyper-partitioning, both the untrusted and trusted environments use the same set of physical devices. Thus, Lockdown must ensure that device states are saved and restored across environment switches using hyper-switching (see Section 4.3). Devices that do not store persistent data, such as video, audio, and input devices can be swapped in this manner.

However, storage devices may contain persistent, sensitive data from the trusted environment, or malicious data from the untrusted environment. Thus, Lockdown ensures that each environment is provided with its own set of storage

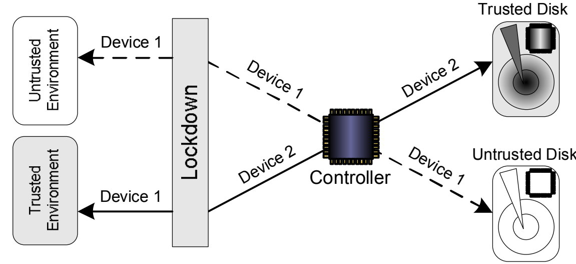


Figure 2: **Hyper-Partitioning of Storage Devices.** *Both environments believe they are using the only disk in the system. In reality, Lockdown intercepts the device selection requests and redirects disk operations to the appropriate disk, based on the current environment.*

devices and/or partitions. For example, Lockdown can assign a different hard drive to each environment. Alternately, Lockdown can assign a different partition on the same hard drive to each environment. The challenge is to create this partition without virtualizating the storage devices, while providing strong isolation that cannot be bypassed by a malicious OS.

Lockdown efficiently partitions storage devices by interposing on device selection, rather than device usage. It takes advantage of the fact that modern storage devices rely on a controller that implements the storage protocol (e.g., ATA, SATA) and directs storage operations to the attached devices. When the operating system writes to the storage controller's I/O registers (a standard set for a given controller type), Lockdown intercepts the write and manipulates the device controller to select the appropriate device for the currently executing environment (see Figure 2). All other device operations (e.g., reads and writes) proceed unimpeded by Lockdown. A similar scheme can be adopted for two partitions on the same hard disk by manipulating sector requests. Our evaluation (Section 8) shows that interposing on device selection has a minimal effect on performance, since a select operation is typically followed by many read and write operations.

Since we assume the BIOS is trusted (Section 3.4), we can be sure that Lockdown will always be booted first, and hence will always maintain its protections over the trusted disk. As additional protection, Lockdown could use disk encryption secured with the TPM, similar to Microsoft's BitLocker.

4.3 Hyper-Switching

Since Lockdown does not employ device virtualization, switching between the untrusted and trusted environment (and vice versa) is a challenge. In modern OSes, device drivers store device configuration information in memory and expect the internal state of the devices to match that configuration. Hence, simply saving the memory contents of one environment and replacing it with the other will not suffice.

Lockdown uses the ACPI Sleep States (employed by all modern OSes) to switch between the trusted and untrusted environments. Figure 3a shows how the ACPI Sleep States are handled by an ACPI-compliant OS. When a sleep command is initiated (e.g., when the user closes the lid on a laptop), the OSPM first informs all currently executing user and kernel-mode applications and drivers about the sleep signal. They, in turn, store the configuration information needed restore the system when it awakes. The device drivers use the OSPM to set desired device power levels. The OSPM then signals the ACPI Subsystem, which ultimately performs chipset-specific operations to transition the system into the desired sleep state. The OSPM polls the ACPI Subsystem for a wake signal to determine when it should reverse the process and wake the system.

With hyper-switching, Lockdown performs an environment switch by transitioning the current environment to sleep and waking up the other. Figure 3b shows the steps required for a hyper-switch, assuming the user starts in the untrusted environment. When the user toggles the switch on the Lockdown Verifier to initiate a switch to the trusted environment (Step 1), the Lockdown Verifier communicates with Lockdown which in turn instructs the OSPM in the untrusted environment to put the system to sleep (Step 2). When the OSPM in the untrusted environment issues the sleep command to the ACPI Subsystem, Lockdown intercepts the command (Step 3), resets all devices, updates the output on the Lockdown Verifier (Step 4), and issues a wake command to the OSPM in the trusted environment (Step 5). Switching back to the untrusted environment follows an analogous procedure.

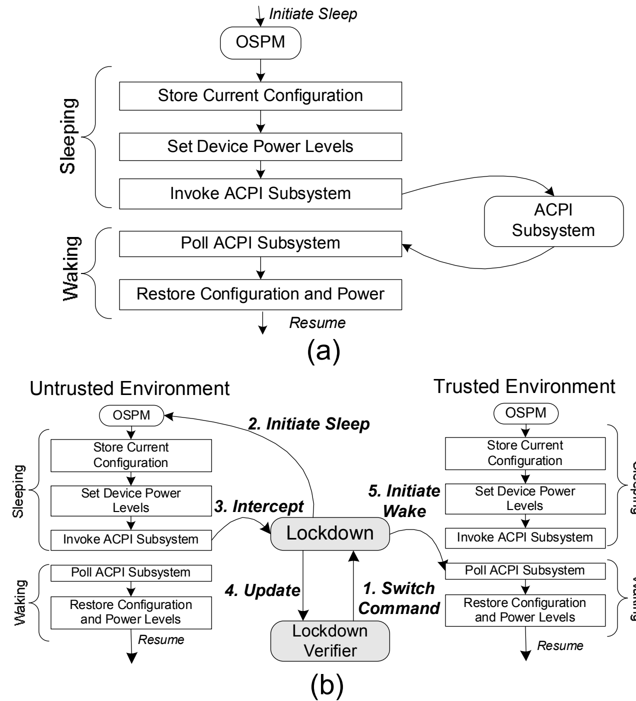


Figure 3: **Hyper-Switching.** **(a)** *Steps taken by an ACPI-compliant OS to sleep and then wake the system.* **(b)** *Lockdown modifies these steps to transition from the untrusted environment to the trusted environment upon receiving a command from the Lockdown Verifier. The transition to the untrusted environment is analogous.*

4.4 External Verification and Trusted Path

While Lockdown always knows whether the trusted or the untrusted environment is currently operating, it must create a trusted path to the user to convey this information in a way she can easily understand and trust. Otherwise, the user might be tricked into performing security-sensitive operations in the untrusted environment.

One approach to conveying whether the system is in the trusted or the untrusted environment would be via the display. However, this approach would require Lockdown to include a display driver, and malware in the untrusted environment might try to trick the user by manipulating the parts of the display that it had legitimate access to.

Lockdown eliminates such attacks by using a simple, external device to control the environment switching and to display the result of the switch to the user. Further, Lockdown ensures that the external device can verify that it is interacting with a correct version of Lockdown, preventing malware from misleading the device.

4.4.1 The Lockdown Verifier The user employs an external device called the Lockdown Verifier to switch between trusted and untrusted environments. To enable the user to trust the Lockdown Verifier, it must possess the following properties:

1. **Correct Operation.** Software executing on the Lockdown Verifier must be robust against compromise. By minimizing the code for the Lockdown Verifier, we make it amenable to formal analysis.
2. **Minimal Input Capabilities.** To minimize complexity (and hence user confusion), we wish to minimize the number of input options.
3. **Minimal Output Capabilities.** To reduce confusion, the user should be able to easily learn which environment she is working in.

To achieve these properties, the Lockdown Verifier consists of a single switch, two LEDs, and a buzzer (Figure 1). The switch can be toggled from secure to insecure (or vice versa). When the user is in the trusted environment, the green LED is lit. When the user is in the untrusted environment, the red LED is lit. To provide additional feedback to the user (e.g., after she toggles the switch), the Lockdown Verifier uses a blinking red LED to indicate processing. Thus, the user need only remember to check that the green LED is lit before performing security-sensitive tasks.

The Lockdown Verifier uses the buzzer to attract the user's attention whenever the LEDs change state. It can also create an alarm buzz if the Lockdown Verifier is unable to verify the correctness of Lockdown or if the system encounters a fatal error.

4.4.2 Secure Channel

To accurately verify the state of the system (trusted or untrusted), the Lockdown Verifier must be able to communicate securely with Lockdown. More precisely, it should not be possible for an adversary to impersonate or undetectably modify Lockdown. Lockdown achieves this goal using a combination of CPU protections and hardware attestation (via a TPM).

To create a secure channel for communicating with the Lockdown Verifier, Lockdown uses CPU protections to reserve a USB controller and to prevent both environments from accessing it. Using USB as an interface is intuitive for users, and it eliminates the need for an external power source for the Lockdown Verifier.

To convince the Lockdown Verifier that it is communicating with Lockdown, we use TPM-based attestation (Section 2.2). Initially, Lockdown is started using a *late launch* operation (Section 2.3), which records a measurement of Lockdown in the TPM. When the Lockdown Verifier is connected to the system, it sends a challenge (a cryptographic nonce) to Lockdown. Lockdown uses the TPM to generate a quote (essentially a signed statement describing the software state of the system) that it transmits to the Lockdown Verifier, which checks the attestation. If verification fails, the Lockdown Verifier halts, sets the LED state to blinking red and emits an alarm buzz. If it succeeds, the Lockdown Verifier emits an attention buzz and sets the LED state to solid red if the untrusted environment is running or to solid green if the trusted environment is running.

Since it is connected via USB, the Lockdown Verifier can also detect when the system is rebooted, since on a reboot, a USB controller sends all attached USB devices a reset signal. When this happens, the Lockdown Verifier emits the attention buzz and sets the LED state to blinking red, since it can no longer vouch for the state of the system. It then performs the procedure described above to verify that Lockdown is back in control and to learn which environment is currently active.

4.5 Trusted Environment Protection

Lockdown's trusted environment runs a commodity OS and applications. Lockdown verifies the integrity of all the files of the trusted environment during Lockdown's installation (Section 5.1). Further, Lockdown trusts the software in the trusted environment to not leak data deliberately. However, vulnerabilities within the OS or an application in the trusted environment can be exploited either locally or remotely to execute malicious code. Further, since the trusted environment and untrusted environment use the same devices, the untrusted environment could change a device's firmware to act maliciously. Lockdown uses approved code execution and network protection to ensure that only trusted code (including device firmware code) can be executed and only trusted sites can be visited while in the trusted environment.

4.5.1 Approved Code Execution

For non-firmware code, Lockdown uses page protections within the physical memory page tables to enforce a $W \oplus X$ policy on physical memory pages used within the trusted environment. Thus, a page within the trusted environment may be executed or written, but not both. Prior to converting a page to executable status, Lockdown checks the memory region against a list of trusted software (see Section 5.2.1 for how this list is established). Execution is permitted only if this check succeeds.

Previous work enforces a similar policy only on the kernel [15], or uses it to determine what applications are running [11]. In contrast, Lockdown uses these page protections to restrict the OS and the applications to a limited set of trusted code.

For device firmware code, Lockdown, during installation, scans all of the hardware installed on the system and enumerates all system and device firmware code regions. It assumes this code has not yet been tampered with and uses physical memory page tables to prevent either environment from writing to these regions.

4.5.2 Network Protection

Since users perform many security-sensitive activities online, applications executing in the trusted environment need to communicate with remote sites via the network. However, permitting network communication exposes the trusted environment to external attacks. Remote attackers may exploit flaws in the OS's network stack, or the user may inadvertently access a malicious site. While the approved code execution described above prevents many code-based attacks, the trusted environment may still be vulnerable to data-based attacks [16].

To forestall such attacks, Lockdown restricts the trusted environment to communicate only with a limited set of trusted sites. It imposes these restrictions by interposing on all network traffic to or from the trusted environment (see Figure 4). Lockdown uses hardware CPU and physical memory protections to prevent the trusted environment from seeing or accessing any physical network devices present in the system. Network communication is permitted via a virtual network interface that Lockdown installs in the guest OS. This interface forwards packets to Lockdown, which analyzes the packets and then forwards them onto the physical network interface.

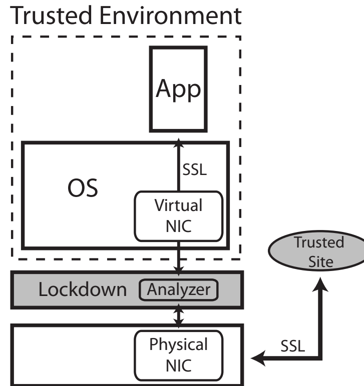


Figure 4: **Trusted Environment Network Protection.** *Lockdown's protocol analyzer only permits SSL sessions to trusted sites.*

Lockdown uses packet analysis to determine which network packets are permitted. In our current design, Lockdown only allows SSL and DNS network packets to pass through. All other packets are dropped. We argue that any site with sensitive data should be using SSL to protect it in transit. When an SSL session is initiated, Lockdown determines if the request is a valid SSL connection request. If it is, Lockdown validates the site's SSL certificate and checks it against the list of trusted sites (the creation and maintenance of this list is discussed in Section 5.2.1). If any of these checks fail, the packet is dropped. Incoming packets are permitted only if they belong to an existing SSL session or are in response to an earlier DNS request. Note that DNS attacks are forestalled by SSL certificate verification.

5 Lockdown Life Cycle

5.1 Installation

Users receive the Lockdown installation kit, which consists of the installation media (CD/DVD) and the Lockdown Verifier, from a trusted source (see Section 5.2.1). The Lockdown Verifier comes preloaded with Lockdown's verification values.

Since Lockdown uses the TPM for attestation, the Lockdown Verifier must obtain the TPM's authentic public key. Numerous solutions exist [13]. As discussed in Section 3.4, we assume the BIOS can be trusted, since most BIOS vendors require signed updates [14]. As a result, the user's BIOS can be used to boot from Lockdown's installation material. Once booted, the Lockdown installer obtains the TPM's public key and installs it in the Lockdown Verifier.

To complete the installation procedure, the Lockdown installer asks the user to designate a hard disk (or partition) as trusted. It then wipes the designated drive, and installs itself there. The user then reboots the system and verifies Lockdown's successful installation using the Lockdown Verifier. Once Lockdown starts, it prompts the user to load the installation medium of the trusted OS. It verifies the medium against a list of trusted OSes and installs the OS on the trusted drive. Lockdown measures the resulting install and is now ready for normal operation.

5.2 Using Lockdown

Startup. When the user first connects the Lockdown Verifier to her system, it sounds an attention buzz and displays a blinking red light to indicate that it is assessing the state of the system. The Lockdown Verifier checks that an unmodified copy of Lockdown is running (Sections 4.4 and 7.4). After Lockdown is loaded, it boots the environment indicated by the current position of the Lockdown Verifier's switch. For example, if the Lockdown Verifier is currently set to "Insecure", Lockdown will launch the untrusted environment and inform the Lockdown Verifier. The Lockdown Verifier will sound an attention buzz and display a solid red LED.

Switching Environments. When the user wishes to perform a security-sensitive task, she flips the switch on the Lockdown Verifier to "Secure". The Lockdown Verifier displays a blinking red light to indicate the switch is in progress and signals Lockdown to begin the switch to the trusted environment (Section 4.3). Once the switch is complete, Lockdown sends a message back to the Lockdown Verifier, which sounds an attention buzz and lights up a green light, signifying that the trusted environment is ready for use. Once in the trusted environment, the user can execute trusted software and visit trusted sites.

Once the user finishes using the trusted environment, she can once again flip the switch on the Lockdown Verifier, which sends a message to Lockdown to switch back to the untrusted environment. Once the switch is complete, the

Lockdown Verifier sounds an attention buzz and lights up a solid red light.
Recovery. If the Lockdown Verifier cannot communicate with Lockdown or if its verification of Lockdown fails, it sounds an alarm buzz and lights up a blinking red light. This indicates that either the system has been altered to prevent Lockdown from loading, or that Lockdown has been modified. At present, the user's best recovery procedure is to reinstall Lockdown. Since the trusted environment is not expected to use many applications, such a reinstallation should not be as painful as a complete reinstall. Nonetheless, this is clearly a heavy-weight recovery mechanism, and we continue to investigate improvements.

5.2.1 Defining Trusted Entities To keep the trusted environment safe, Lockdown restricts the software that can execute and the sites that can be visited. To define what software and sites can be trusted, we leverage the user's existing trust in the distributor of Lockdown, i.e., the organization that provided the user with a copy of Lockdown in the first place. For example, in a corporation, the IT department would play the role of Lockdown distributor. For consumers, the role might be played by a trusted company or organization, such as RedHat, Mozilla, or Microsoft. Lockdown's key insight is that by agreeing to install Lockdown, the user is expressing their trust in the Lockdown provider, since Lockdown will be operating with maximum platform privileges on their computer. Thus, we might as well trust that same organization to vet trusted software and websites for the user. To some extent, Linux distributions already implicitly operate on this assumption. When a user installs a distribution such as Fedora or Debian Linux, they also trust those distributions to provide software that does not contain malware.

Since Lockdown focuses on protecting online transactions, the list of trusted software can be relatively small: primarily an operating system and a trusted browser. The list of trusted sites is necessarily larger, since it should include all of the various security-sensitive companies a user might interact with. However, to limit potential leaks to entities on the list that the user does not interact with, the user can customize the list.

During Lockdown's installation, the user is presented with the master list of trusted software and trusted websites and allowed to select a subset of each list. Thus, the user can choose her favorite web browser, and select the handful of websites she actually uses from the hundreds of sites on the master list. Lockdown will then prohibit the trusted environment from contacting any site not on the user's restricted list. A small application that runs in the trusted environment allows the user to update her selection at a later time.

5.3 Updates

As websites evolve and software moves to new versions, the lists of trusted software and trusted sites must be updated. This process is straightforward, since the Lockdown distributor can simply release a signed update to both lists. Lockdown can verify the signature on the update using the distributor's public key (included in the initial Lockdown installation), and update its policy files to include the latest versions of both lists. As mentioned above, the user can use a small applications in the trusted environment to update the subset of trusted software and trusted sites she wishes to use.

6 Security Analysis

6.1 Lockdown's Security Properties

Trusted Environment Isolation. Lockdown's hyper-partitioning and network protection mechanisms are designed to isolate the trusted environment from local and remote malware. Locally, Lockdown ensures that the trusted environment and untrusted environment never execute concurrently, preventing malware in the untrusted environment from directly interfering with the trusted environment's execution. Lockdown's use of Nested Page Tables ensures that software in the untrusted environment cannot even address the trusted environment's memory region, thus protecting its secrecy and integrity. To prevent device-based attacks, Lockdown uses AMD's DEV to prevent DMA-based reads and writes to sensitive areas, and it ensures that all devices are reset during a hyper-switch. Devices with persistent data (such as storage devices) are partitioned between the two environments to prevent secrets from leaking out of the trusted environment, and to prevent maliciously crafted inputs from penetrating into the trusted environment.

Remotely, Lockdown's network protections prevent untrusted entities from contacting the trusted environment. To provide defense-in-depth, these protections also prevent the trusted environment from contacting untrusted sites. Thus, even if a bug in the trusted OS or applications results in a data leak, the data can only travel to sites the user already trusts with her data.
Code Integrity. Lockdown's approved execution ensures that only measured code that appears on Lockdown's list of trusted software can run within the trusted environment. Further, once the code is measured, Lockdown renders it

immutable. Lockdown thus prevents a significant class of attacks that modify existing code or execute new malicious code. However, this approach does not check interpreted code (e.g., JavaScript). Hence, if a trusted site is compromised, it may allow an attacker to manipulate the trusted environment. Thus, one drawback of Lockdown's current approach is that a compromise at one of the user's trusted sites can affect the security of her transactions at other sites. Improving browser-based isolation can mitigate these concerns [3, 20], but eventually, we anticipate a trusted environment for each trusted site.

Trusted Path. Lockdown is designed to create a trusted path to the user. In other words, Lockdown should provide the user with the confidence that she is communicating securely with the party she intends to contact. Lockdown achieves this property by providing a simple indicator (a green LED) on the Lockdown Verifier to signal when the user is operating in the trusted environment. This indicator is only provided in response to a message received from Lockdown over the secure channel that the Lockdown Verifier establishes with Lockdown (Section 4.4). This channel is protected by Lockdown's exclusive access to the USB controller combined with the TPM's ability to provide a verifiable summary of the system's software.

6.2 Other Attacks

Denial of Service. Lockdown's hyper-switching mechanism triggers the sleep state in the OSPM of the untrusted environment in order to switch to the trusted environment. However, malware in the untrusted environment can modify the OSPM to ignore the sleep command. Thus, malware in the untrusted environment can keep the trusted environment from loading. However, it cannot do so undetectably. Before Lockdown triggers the sleep state in the OSPM of the untrusted environment, it lights up a blinking red LED on the Lockdown Verifier and sounds an attention buzz to indicate processing. If the untrusted environment ignores the sleep command, then the switch to the trusted environment will never complete, and hence the Lockdown Verifier LED will never glow green. Lockdown relies on the user to wait for a green LED before performing any security-sensitive tasks.

Corrupt Lockdown Distributor. Lockdown depends on an external party to define the master list of trusted software and trusted sites. If this party were corrupted, the user might install malicious software in the trusted environment or visit malicious sites. However, consumers already depend on remote entities for software updates. For example, if an attacker could corrupt the Windows Update Service, then he could perform a similar attack to load malware onto millions of machines. Lockdown merely leverages this existing trust to more precisely define what can be done in the trusted environment.

Social Engineering. A clever attacker may convince the user to perform a security-sensitive task in the untrusted environment, rather than in the trusted environment. Lockdown cannot prevent such an attack; it can only rely on the user to check the system's status as displayed by the Lockdown Verifier, and to switch to the trusted environment for security-sensitive tasks. With sufficient user education, this may be a reasonable expectation.

7 Implementation

We implemented a complete prototype of Lockdown with Windows 2003 Server SP1 as the OS in both the trusted and untrusted environments. To demonstrate that Lockdown's hyper-partitioning and hyper-switching are generic primitives that work with other ACPI-compliant OSes, we also developed a prototype using Linux guests. Neither prototype required changing any code in the OS kernels. Due to space constraints, we focus on describing our Windows prototype.

7.1 Lockdown Components

Lockdown Loader. The Lockdown Loader consists of two pieces: the Boot Loader and the Loader Core. The untrusted Boot Loader uses the SKINIT instruction to perform a late-launch (Section 2.3) of the Loader Core. Using the SKINIT instruction ensures that the Loader Core runs in a hardware-protected environment and that a measurement (cryptographic hash) of the Loader Core is stored in the TPM's PCR 17.

The trusted Loader Core is responsible for loading the Lockdown Runtime. To do so, it first protects the Lockdown Runtime's memory region from DMA reads and writes using AMD's DEV protections. It then verifies the integrity of the Lockdown Runtime and extends a measurement (a cryptographic hash) of the Lockdown Runtime's code into the TPM's PCR 18.

The Loader Core must also initialize Lockdown's devices and setup the initial hyper-partitioning. It first initializes the physical network card used for communication on behalf of the trusted environment. It also initializes the USB

controller for communication with the Lockdown Verifier. The Loader Core then creates the Nested Page Tables (NPTs) for the trusted and untrusted environments, and it creates Virtual Machine Control Blocks (VMCB) to ensure that each environment uses the appropriate set of page tables and to prohibit certain privileged operations. Finally, it transfers control to the Lockdown Runtime.

Lockdown Runtime. The Lockdown Runtime implements all of Lockdown's functionality. When first launched, the Lockdown Runtime requests a challenge from the Lockdown Verifier. The Lockdown Runtime and the Lockdown Verifier then engage in the authentication protocol described in Section 4.4. The Lockdown Runtime loads the VMCB for the environment currently indicated on the Lockdown Verifier, and informs the Lockdown Verifier once the environment has been launched, so that the Lockdown Verifier can sound the attention buzz and light the appropriate LED. The Lockdown Runtime's role in hyper-partitioning, hyper-switching, and protection of the trusted environment is described below.

7.2 Hyper-Partitioning

Memory. In our current implementation, Lockdown allocates 1.75 GB of physical memory to each of the trusted and untrusted environments, while it reserves 186 MB for itself and 258 MB for the system's firmware. The isolation between the two environments is maintained by creating two sets of non-overlapping NPTs. In both sets of NPTs, the NPT entries which point to Lockdown's physical memory regions are marked not-present, while the entries for the system firmware are set to prohibit writes.

Disks. Our prototype is designed for a machine with two ATA hard disks. The master drive is used as the untrusted drive, and the slave drive is used as the trusted drive. As per the ATA specification, before any I/O can be performed on a drive, the drive must first be identified. This identifier is a single bit (bit 4) in a byte command written the primary controller drive-identification port (port 0x1F6). If the bit is clear, the master drive is selected, else the slave. A read to port 0x1F6 returns the status of the selected drive.

To partition the disks, Lockdown configures the VMCBs of both the trusted and untrusted environments to intercept read and write accesses to port 0x1F6. Intercepting writes allows Lockdown to prevent the trusted environment from accessing the untrusted disk (and vice versa). For example, if the trusted environment writes a request to port 0x1F6 to select the master drive, an exception is generated, returning control to Lockdown. Lockdown writes to the disk controller's register and selects the slave (trusted) disk instead. A similar procedure prevents the untrusted environment from selecting the trusted disk.

Intercepting read requests allows Lockdown to "hide" one of the disks from each environment. For example, if the current environment is trusted and there is a read to port 0x1F6, Lockdown ensures that bit 4 in the status register is cleared, which convinces the trusted environment that the trusted drive is actually the master drive on the system. Similarly, if the untrusted environment writes to port 0x1F6 to select the slave drive, Lockdown returns a drive not present status on the read.

7.3 Hyper-Switching

To implement hyper-switching, Lockdown makes use of the ACPI S4 (hibernate) Sleep state. Sleep states S1–S3 might offer faster switching times, but most BIOSes only implement S3 and S4. Windows' ACPI implementation only saves and restores device state during an S4 Sleep, and hence we cannot use S3 with Windows without modifying its source code.

The Loader Core is responsible for establishing Lockdown's control over the system's ACPI modes. The ACPI sleep modes are controlled by the ACPI Sleep Register and the ACPI Status Register. The Loader Core determines the I/O location of these registers by parsing the ACPI Fixed Address Descriptor Table (FADT), which is constant for a given chipset. The Loader Core also uses the FADT to determine the system-specific value that must be written to the ACPI sleep register to activate the S4 sleep state. Finally, the Loader Core configures the trusted and untrusted environments' VMCBs to intercept all attempts to access the ACPI Sleep and Status registers.

When the user toggles the switch on the Lockdown Verifier, Lockdown sets an internal switch flag and signals the Lockdown Monitor inside the current environment to initiate the sleep state. The Lockdown Monitor is an untrusted application which uses the SetSuspendState Windows API in order to trigger an S4 Sleep. The OSPM in Windows then prepares the system for hibernation, saves the memory contents to disk, and writes the special value to the ACPI Sleep Register. Lockdown captures this write and instead clears the switch flag and updates the Lockdown Verifier to indicate the newly active environment. Lockdown then resets the system via a soft-reset to reset the device states. Finally, Lockdown launches the target environment by waking it from hibernation. The Windows OS in the

target environment loads the hibernation image from the disk, restores the device states, and transfers control to the Windows Kernel.

7.4 External Verification and Trusted Path

We built a Lockdown Verifier using the LPC P2148 microcontroller development board, which is equipped with a USB 2.0 interface. The board has a 32-bit ARM7TDMI-S processor executing at 60 MHz, with 512 KB of flash memory and 42 KB of RAM. We attached a red and a green LED, a switch, and a buzzer to the board. The Lockdown Runtime contains a tiny OHCI-compliant USB driver that is used to communicate with the Lockdown Verifier. The Lockdown Runtime also contains a small TPM TIS 1.2 driver which communicates with the Broadcom TPM.

The code on the Lockdown Verifier consists of a USB stack for communicating with Lockdown and a set of routines to control the LED and buzzer and to respond to the user toggling the switch. The Lockdown Verifier upon reset or power-up waits for a challenge request from Lockdown. Upon receiving the challenge request, the Lockdown Verifier transmits a cryptographic nonce and receives a TPM-generated attestation from Lockdown. The attestation contains the TPM's signature over the current values of PCRs 17 and 18, as well as the nonce that was provided. The Lockdown Verifier uses the TPM's public-key (installed during Lockdown's installation – see Section 5.1) to verify the attestation. If the verification succeeds, the Lockdown Verifier goes into a trusted communication mode with Lockdown and responds to commands to set LEDs and report on the switch's status, until the system is reset or turned off.

7.5 Protecting the Trusted Environment

7.5.1 Approved Code Execution To enforce approved code execution, Lockdown uses page-level code hashing, similar to the approach used by previous work [11, 15]. Prior to executing the trusted environment with the VMRUN instruction, Lockdown sets its NPT entries to prevent execution of those pages. When the trusted environment attempts to execute a page, it causes a fault that returns control to Lockdown. Lockdown computes a hash of the faulting page and compares it to the hashes in its list of trusted software. If a match is found, the page's NPT entry is updated to allow execution but prevent writes. If the trusted environment later writes to this page, a write fault will be generated. Lockdown will re-enable writing but disable execution.

Matching a code page to the list of approved software is straightforward. In Windows, an application's entire executable is mapped into memory, so the executable's header and relocation tables are always present at runtime. Lockdown uses this information to compute the inverse of the relocation operation and compare the page to hashes of the software's original executable.

7.5.2 Network Protection To provide network protection for the trusted environment, we developed a tiny network driver within Lockdown, an untrusted network driver for the guest OS, and an SSL Protocol Analyzer.

Lockdown's network driver is a simplified version of the Linux Intel Pro/1000 driver. It employs ring-buffer DMA transfers for sending and receiving network packets. Our OS-level network driver is a Windows NDIS deserialized miniport driver. This driver sends and receives network packets to and from the SSL Protocol Analyzer via a VMMCALL instruction.

We used `ssldump`[1] as our code base for the SSL Protocol Analyzer. We removed OS dependencies and added support for SSL session tracking and event handling depending on the SSL packet (e.g, Certificate, ServerHello etc.). The certificate event handler is used to compare a site's SSL certificate against Lockdown's list of trusted-site certificates. The Analyzer tracks ongoing SSL sessions and drops any packets that do not belong to an existing session. The one exception is to allow outbound DNS requests (for trusted site domain names) and the corresponding DNS replies.

8 Evaluation

In this section we evaluate our Lockdown prototype using two metrics: code size and performance.

8.1 Trusted Computing Base (TCB)

Like all security systems, Lockdown must assume the correctness and security of its core components. One way to make this assumption more likely to hold is to reduce the amount of code that must be trusted. This reduces the opportunities for bugs and makes the code more amenable to formal analysis.

[1] http://www.rtfm.com/ssldump/

Lockdown Component	SLOC		Total
	ANSI-C	ASM	
Lockdown Loader	2307	84	2391
Lockdown Runtime Core	1994	304	2298
Lockdown Runtime Approved Execution + Network Protection	2658	0	2658
Lockdown Verifier	1053	71	1124

(a)

Hypervisor/Micro-Kernel	SLOC
Lockdown	8,471
L4 Micro-kernel (x86-32)	25,066
VMWare ESX	~500,000
Xen + Linux Kernel 2.6.23 as DOM0	~30,000,000
KVM + Linux Kernel 2.6.23 + QEMU (x86-32)	~30,000,000
Hyper-V + Windows	~150,000,000

(b)

Figure 5: **Lockdown's TCB.** **(a)** *Code sizes of Lockdown's components.* **(b)** *Comparison with popular, general-purpose hypervisors and micro-kernels.*

We use the `sloccount`[2] utility to measure the size of our prototype for Windows (Figure 5a). We divide the prototype into four components. The Lockdown Loader initializes the CPU, memory, and Lockdown's runtime state following the SKINIT instruction. The Lockdown Runtime Core handles hyper-partitioning and hyper-switching. The rest of the Lockdown Runtime protects the trusted environment. The final component is the code running on the Lockdown Verifier.

Combining all of these components, Lockdown's total TCB is only 8,471 SLOC, placing Lockdown within the reach of formal verification and manual audit techniques. Lockdown's TCB compares favorably with other popular hypervisors and VMMs (Figure 5b), which tend to be orders of magnitude larger, despite not providing Lockdown's protection's for a trusted environment. Xen, KVM, and Hyper-V include an entire OS in the TCB for administrative purposes, dramatically increasing their TCBs. While VMware does not require such an OS, it still includes a large TCB, since it employs full virtualization of devices and hence must include device drivers for all of the platforms it wishes to support. Only the L4 micro-kernel [6] approaches Lockdown's TCB size. However, porting an OS to L4 requires considerable modification to the OS kernel.

8.2 Performance Measurements

We use our prototype to determine a rough estimate of Lockdown's performance, with a focus on the version for Windows. All measurements were taken on an AMD Opteron (quad-core, 2.0GHz) system with 4 GB of physical memory. The system has two 40 GB ATA hard disks, an Intel 82572EI PCI-Express gigabit network adapter, two USB controllers, and a v1.2 Broadcom TPM chip.

Results Overview. Lockdown aims to preserve the performance of the frequently-used untrusted environment, and in our experiments, we find that the untrusted environment only experiences an average of 3% CPU overhead and 2–7% storage overhead. All other devices run at full speed, allowing the untrusted environment to have, for example, native-speed networking and graphics performance that virtualization cannot provide.

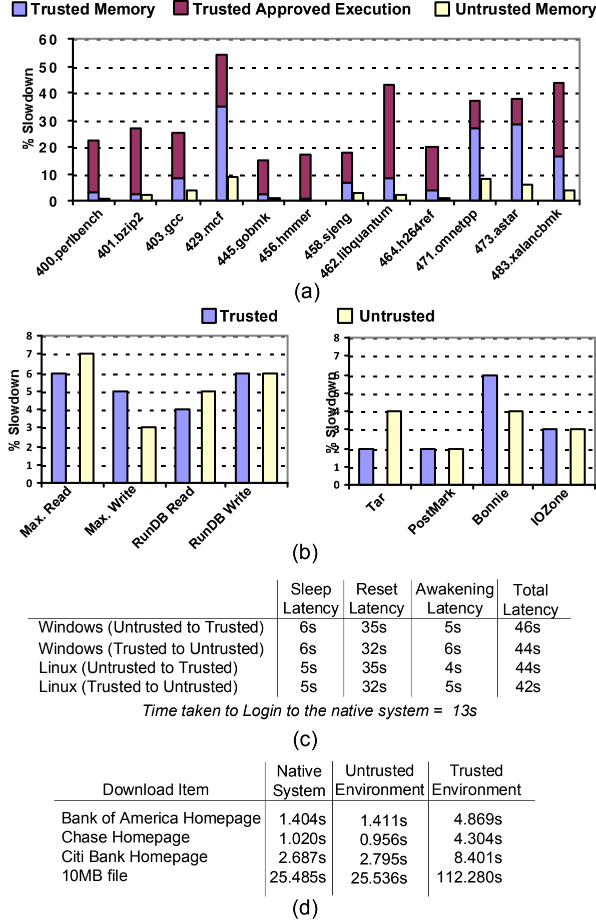
Lockdown achieves this performance (in part) by shifting the performance overhead to the infrequent operation of switching into the trusted environment to perform security-sensitive activities. The switch currently takes 42–46 seconds, though we expect soon-to-be-deployed hardware standards to significantly decrease this time. The additional protections Lockdown provides for the trusted environment also come at a performance cost (15-55% CPU overhead and 3–6 seconds additional network latency downloading web pages). However, since the security tasks tend to occur less frequently and often demand less performance, we believe this is an acceptable tradeoff for the security provided.

8.2.1 CPU and Memory Overhead Lockdown's use of hardware-supported Nested Page Tables (NPTs) to hyper-partition memory adds latency to memory accesses, since it adds an extra layer of indirection when resolving addresses. AMD and Intel are actively working to improve the performance of this recently-added feature [1]. Lockdown also adds overhead to code execution in the trusted environment due to its approved code verification.

To measure the CPU and memory overhead, we use benchmarks from the SPECint 2006 suite. We run the benchmarks in the trusted environment, in the untrusted environment, and on the native system. We also run the benchmarks in the trusted environment with approved code protection disabled to allow us to distinguish between overhead added by these protections and that added by the NPTs.

Figure 6a shows Lockdown's overhead as a percentage of the native system's performance. In the untrusted en-

[2]http://www.dwheeler.com/sloccount/



	Sleep Latency	Reset Latency	Awakening Latency	Total Latency
Windows (Untrusted to Trusted)	6s	35s	5s	46s
Windows (Trusted to Untrusted)	6s	32s	6s	44s
Linux (Untrusted to Trusted)	5s	35s	4s	44s
Linux (Trusted to Untrusted)	5s	32s	5s	42s

Time taken to Login to the native system = 13s

(c)

Download Item	Native System	Untrusted Environment	Trusted Environment
Bank of America Homepage	1.404s	1.411s	4.869s
Chase Homepage	1.020s	0.956s	4.304s
Citi Bank Homepage	2.687s	2.795s	8.401s
10MB file	25.485s	25.536s	112.280s

(d)

Figure 6: **Lockdown Performance Measurements.** **(a)** *CPU and memory overhead relative to native (smaller is better),* **(b)** *Storage micro- and macrobenchmarks compared to native (smaller is better),* **(c)** *Switching latency,* **(d)** *Network-protection latency.*

vironment, performance is only slightly worse than native (3% average overhead). The trusted environment adds considerably more overhead (15–55%). Even without including the overhead of approved code execution, the trusted environment is still slower than the untrusted environment due to its use of smaller NPT pages. NPTs can be instantiated with 4 KB pages or with 2 MB pages. In the untrusted environment, we use the 2 MB pages to improve performance. However, in the trusted environment, we also use the NPTs to check for approved code at a page granularity, and hence the trusted environment must use the smaller 4 KB NPTs, making it less efficient. Nonetheless, this performance seems reasonable for infrequent security tasks, such as online banking, that are less performance intensive.

8.2.2 Storage Overhead To partition the system's disks between the trusted and untrusted environments, Lockdown intercepts both environments' drive selection commands, adding overhead to disk I/O.

To measure this overhead with microbenchmarks, we employ Iometer, an industry-standard disk benchmarking tool. We use Iometer to measure Lockdown's maximum throughput for direct reads and writes, as well as reads and writes from a database workload. For macrobenchmarks, we use a variety of standard disk-bound applications, including Postmark (with 10000 files and 10000 transactions), IoZone (with a 2 GB file), Bonnie (with a 2 GB file), and `tar` (on the Windows installation folder).

Figure 6b shows the results of these benchmarks relative to the native system's performance. As expected (since Lockdown treats both environments equally when partitioning storage devices), the two environments perform similarly. On these disk-bound tests, Lockdown imposes relatively modest overheads of 2–7%.

8.2.3 Switching Latency Using the ACPI infrastructure to switch between environments takes a non-trivial amount of time. We split the time into three parts: **(a)** sleep latency: the time taken from when the user flips the switch on the Lockdown Verifier to the time the guest OS finishes preparing for sleep and invokes the ACPI subsystem, **(b)** reset

latency: the time taken for Lockdown to reset the system's devices, and transfer control to the target environment's OSPM and, **(c)** awakening latency: the time taken by the OSPM in the target environment to resume normal operations.

Figure 6c shows the measurements for Lockdown's hyper-switching latency for both Windows and Linux. The switch currently requires 44–46 seconds on Windows and 42–44 seconds on Linux. While longer than ideal, we expect users to swap between the two environments relatively infrequently.

Currently, the reset phase occupies 70–95% of the switching time. This is largely due to Lockdown's use of the BIOS to reset the system's devices. The BIOS performs a far more extensive reset than Lockdown needs, completely reinitializing the CPU, chipset, memory and devices. The reset process should be significantly accelerated as computers adopt the new PCI-Express 2.0 bus standard. With this standard, Lockdown can use a single PCI-bus command to reset each device in the system, instead of using the BIOS.

Our results indicate that the direction of the switch has a relatively small impact on the switching time. The untrusted to trusted direction takes slightly longer during the reset phase since Lockdown must initiate the physical network device it uses to communicate on the trusted environment's behalf. Windows and Linux currently take 4–6 seconds on the sleep and awake phases. This is because they write all memory contents to a hibernation file on disk every time the S4 sleep state is used. With the accelerated reset process using the PCI-Express 2.0 bus standard, we could eliminate this latency by writing a small untrusted storage driver wrapper which simply discards the IOCTL for hibernation reads and writes.

8.2.4 Network Protection Overhead Since Lockdown interposes on the trusted environment's network connections, we expect performance to be worse in the trusted environment. However, the untrusted environment has full access to the network interface, and hence should be comparable to native.

To measure Lockdown's network overhead, we use Firefox with the YSlow add-on to measure the time necessary to load three popular banking websites, as well as the time required to download a 10 MB file. We averaged the download times over 10 runs, clearing the Firefox cache each time. The tests were performed in the middle of the night, when the network traffic to these sites is reduced.

Figure 6d summarizes our results. As expected, the untrusted environment's performance is equivalent to the native system (within experimental error). The trusted environment takes longer, but the web page times are still within the realm of user tolerance. The trusted environment's network overhead becomes more noticeable on the large file download. Fortunately, most security-sensitive online transactions involve small network transmissions that are more similar to the web pages than the large file download.

9 Discussion

Limitations. One of Lockdown's fundamental limitations is that the trusted and untrusted environments do not execute concurrently. Less fundamental, but still of concern, is the time needed to switch between the two environments. We hypothesize that users perform security-sensitive tasks infrequently and will tolerate the need for an explicit switch before performing such tasks. A user study would help validate this hypothesis.

Data Transfer. Lockdown's current design does not permit data transfer between the trusted and untrusted environments. This policy provides the strongest security, since it prevents secrets from leaking out of the trusted environment, and it prevents malware in the untrusted environment from injecting malicious data into the untrusted environment.

From an technical perspective, other policies are certainly possible. For example, Lockdown could allow the trusted environment write-only access to the untrusted disk by allowing writes but blocking read accesses, so as to prevent attacks from the untrusted environment. Since software in the trusted environment is trusted, it may be acceptable to trust that neither it nor the user will transfer sensitive data to the untrusted disk.

Choosing the best policy depends on the user and the installation environment. More nuanced policies may also be enabled by improved defenses against, for example, data-based attacks [16].

10 Related Work

Systems such as NetTop [12], Terra [4], or Overshadow [2] use virtualization to isolate code running at different security levels. As discussed in Section 3.3, virtualization allows rapid switching (orders of magnitude faster than Lockdown) between multiple environments. However, virtualization leads to side-channels that may leak sensitive information. Device virtualization also degrades performance and increases the amount of code that must be trusted by orders of magnitude (see Section 8.1).

Several proposals use virtualization to isolate one web application from another [3, 20], but they do not protect the web browser from other code on the system. However, this work would be complementary to Lockdown if used within the trusted environment to prevent a compromise of one trusted site from affecting the other trusted sites.

Specialized hypervisor systems such as Proxos [18] and Nizza [17] allow a small, specially-crafted piece of code to run in isolation from the rest of the system. However, they typically do not protect general-purpose applications or provide full access to system devices.

Garriss et al. use a cellphone to verify a VMM on a public kiosk [5]. While convenient, cellphones are increasing in complexity and hence becoming more vulnerable to malware. This approach also has the drawbacks associated with virtualizing the entire platform.

Lockdown's design is inspired in part by Lampson's proposal for a Red/Green system [10]. Lockdown goes beyond the proposal by adding a trusted path, in the form of the Lockdown Verifier, to the user. It also provides a design to actually provide the necessary isolation, as well as an implementation to evaluate the system's performance.

11 Conclusion

We designed and built Lockdown, a system that significantly increases the level of security for online transactions, even on a platform containing malicious code. We implemented a complete prototype of Lockdown for the Windows OS in 8,471 lines of code, as well as a partial prototype for Linux (testing our core concepts of hyper-partitioning and hyper-switching), without needing to change a single line of code within the OSes. Our initial experience shows that we can provide immediate improvements to security-sensitive tasks while imposing minimal performance overhead on other applications.

References

[1] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating two-dimensional page walks for virtualized systems. In *ASPLOS*, Mar. 2008.

[2] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. K. Ports. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. In *ASPLOS*, Mar. 2008.

[3] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen. A safety-oriented platform for web applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 350–364, May 2006.

[4] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of Symposium on Operating System Principles (SOSP 2003)*, Oct. 2003.

[5] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and personalized computing on public kiosks. In *MobiSys*, 2008.

[6] H. Härtig, M. Hohmuth, J. Liedtke, J. Wolter, and S. Schönberg. The performance of μ-kernel-based systems. In *SOSP*, 1997.

[7] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification. Revision 3.0b, Oct. 2006.

[8] P. Karger and D. Safford. I/O for virtual machine monitors: Security and performance issues. *IEEE Security and Privacy*, 6(5):16–23, 2008.

[9] B. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[10] B. Lampson. Accountability and freedom. `http://research.microsoft.com/Lampson/Slides/AccountabilityAndFreedomAbstract.htm`, 2005.

[11] L. Litty, H. A. Lagar-Cavilla, and D. Lie. Hypervisor support for identifying covertly executing binaries. In *USENIX Security Symposium*, 2008.

[12] R. Meushaw and D. Simard. Nettop: Commercial technology in high assurance applications. *VMware Tech Trend Notes*, 9(4):1–8, 2000.

[13] B. Parno. Bootstrapping trust in a "trusted" platform. In *USENIX Workshop on Hot Topics in Security*, July 2008.

[14] Phoenix Technologies. TrustedCore: Foundation for secure CRTM and BIOS implementation. `https://forms.phoenix.com/whitepaperdownload/docs/trustedcore_wp.pdf`, 2006.

[15] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *SOSP*, 2007.

[16] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *ACM CCS*, 2007.

[17] L. Singaravelu, C. Pu, H. Haertig, and C. Helmuth. Reducing TCB complexity for security-sensitive applications: Three case studies. In *EuroSys*, 2006.

[18] R. Ta-Min, L. Litty, and D. Lie. Splitting interfaces: Making trust between applications and operating systems configurable. In *OSDI*, 2006.

[19] Trusted Computing Group. Trusted Platform Module Main Specification. Version 1.2, Revision 103, 2007.

[20] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter. The multi-principal OS construction of the gazelle web browser. Technical Report MSR-TR-2009-16, Microsoft Research, 2009.