Lockdown: Towards a Safe and Practical Architecture for Security Applications on Commodity Platforms

Amit Vasudevan[1], Bryan Parno[⋆2], Ning Qu[⋆⋆3], Virgil D. Gligor[1], and Adrian Perrig[1]

[1] CyLab/Carnegie Mellon University
{amitvasudevan, gligor, perrig}@cmu.edu
[2] Microsoft Research
parno@microsoft.com
[3] Google Inc.
quning@gmail.com

Abstract. We investigate a new point in the design space of red/green systems [19, 30], which provide the user with a highly-protected, yet also highly-constrained trusted ("green") environment for performing security-sensitive transactions, as well as a high-performance, general-purpose environment for all other (non-security-sensitive or "red") applications. Through the design and implementation of the Lockdown architecture, we evaluate whether *partitioning*, rather than virtualizing, resources and devices can lead to better security or performance for red/-green systems. We also design a simple external interface to allow the user to securely learn which environment is active and easily switch between them. We find that partitioning offers a new tradeoff between security, performance, and usability. On the one hand, partitioning can improve the security of the "green" environment and the performance of the "red" environment (as compared with a virtualized solution). On the other hand, with current systems, partitioning makes switching between environments quite slow (13-31 seconds), which may prove intolerable to users.

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. Red/green systems [19, 30] have been proposed as a mechanism for improving user security without abandoning the generality that has made computers so successful. They are based on the observation that users perform security-sensitive transactions infrequently, and hence enhanced security protections need only be provided *on demand* for a limited set of activities. Thus, with a red/green system, the user spends most of her time in a general-purpose, untrusted (or "red") environment which retains the full generality of her normal computer; i.e., she can install arbitrary applications that run with good performance. When the user wishes to perform a security sensitive transaction, she switches to a trusted (or "green") environment that includes stringent protections, managed code, network and services at the cost of some performance degradation.

⋆ This work was done while Bryan Parno was still at CyLab/Carnegie Mellon University
⋆⋆ This work was done while Ning Qu was still at CyLab/Carnegie Mellon University

The typical approach to creating a red/green system relies on virtualization to isolate the trusted and untrusted environments [19, 30]. While straightforward to implement, this approach has several drawbacks. First, it requires virtualizing all of the system resources and devices that may be shared between the two environments. From a security perspective, this introduces considerable complexity [16] into the reference monitor (i.e., the virtual machine monitor) responsible for keeping the two environments separate. In addition, even without compromising a reference monitor, actively sharing resources by allowing both environments to run simultaneously exposes side-channels that can be used to learn confidential information [36, 9, 31, 18]. From a performance perspective, the interposition necessary to virtualize devices adds overhead to both trusted and untrusted applications [16].

Through our design and implementation of the Lockdown architecture, we investigate whether *partitioning* resources can overcome these drawbacks. In particular, Lockdown employs a light-weight hypervisor to partition system resources across time, so that only one environment (trusted or untrusted) runs at a time. When switching between the two environments, Lockdown resets the state of the system (including devices) and leverages existing support for platform power-management to save and restore device state. This approach makes Lockdown device agnostic, removes considerable complexity from the hypervisor, and yet maintains binary compatibility with existing free and commercial operating systems (e.g., Windows and Linux run unmodified). It also allows the untrusted environment to have unfettered access to devices, resulting in near native performance for most applications, although a small performance degradation is necessary to protect Lockdown from the untrusted environment. In the trusted environment, Lockdown employs more expensive mechanisms to keep the environment pristine. For example, 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.

As an additional contribution, we study the design and implementation of a user interface for red/green systems that is independent of the choice of virtualization versus partitioning. Our design results in a small, external USB device that communicates the state of the system (i.e, trusted or untrusted) to the user. The security display is beyond the control of an adversary and cannot be spoofed or manipulated. Its simple interface (providing essentially one bit of input and one bit of output), makes it easy to understand and use, and overcomes the challenges in user-based attestation [26] to create a trusted communication channel between the user and the red/green system.

We have implemented and evaluated a full prototype of our user interface (which we call the Lockdown Verifier) plus Lockdown for Windows and Linux on commodity x86 platforms (AMD and Intel). To the best of our knowledge, this represents the first complete, end-to-end design, implementation and evaluation of a red/green system on commodity platforms; we discuss related work in § 8. The Lockdown hypervisor implementation has 10K lines of code, including the code on the Lockdown Verifier. The small size and simple design supports our hypothesis that partitioning (instead of virtualization) can improve security. Our evaluation also indicates that the performance of

untrusted applications is the same or better with partitioning (as opposed to virtualization). Lockdown only imposes a 3% average overhead for memory and 2-7% overhead for disk operations for untrusted applications. Virtualization on the other hand imposes overhead for *all* platform hardware with the overhead ranging from 3-81% depending on the resources being virtualized (§ 7.2). The primary limitation of partitioning on current systems is the time (13–31 seconds) needed to switch between the two environments. While we describe several potential optimizations that could significantly reduce this time, whether this tradeoff between security, performance, and usability is acceptable remains an open question.

2 Problem Definition

Goals. The goal of a red/green system 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. Protecting trusted software that does not require network access is a strict subset of this goal. Ideally, this should be achieved 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.

Adversary Model. We assume the adversary can execute arbitrary code within the untrusted environment and may also monitor and manipulate network traffic to and from the user's machine. However, we assume the adversary is remote and cannot perform physical attacks on the user's machine.

Assumptions. The first three assumptions below are necessary for any red/green system. The last two are particular to Lockdown's implementation. (i) **Trusted Software and Sites**: As we discuss in § 3.2, we assume certain software packages and certain websites can be trusted to not deliberately leak private data; (ii) **Reference Monitor Security**: We assume that our reference monitor code does not contain vulnerabilities. Reducing the complexity and amount of code in the reference monitor (as we do with Lockdown) allows manual audits and formal analysis to validate this assumption; (iii) **User Abilities**: We assume the user can be trained to perform security-sensitive operations in the trusted environment; (iv) **Hardware Support**: We assume the user's computer supports Hardware Virtualization Extensions (with Nested Page Table support [10]) and contains a Trusted Platform Module [44] chip. Both technologies are ubiquitous; and (v) **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 [32], preventing most forms of attack.

3 Lockdown's Architecture

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, 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-

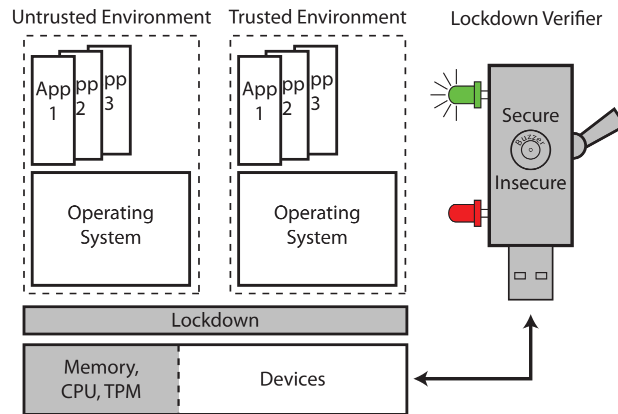


Fig. 1. Lockdown System Architecture. *Lockdown partitions the platform into two environments; only one environment executes at a time. An external device (which we call the Lockdown Verifier) verifies the integrity of Lockdown, indicates which environment is active and can be used to toggle between them. The shaded portions represent components that must be trusted to maintain isolation between the environments.*

critical. The performance-sensitive applications run in the untrusted environment with 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 two 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; and (ii) **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.

3.1 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. One way to achieve this isolation is to rely on the platform hardware to partition resources. With platform capabilities such as Single-Root I/O Virtualization (SR-IOV) [29] and additional hardware such as an IOMMU, it is possible to assign physical devices directly to an environment (untrusted or trusted) [4, 17]. This hardware capability facilitates concurrent execution of multiple partitions without virtualizing devices. Unfortunately, not all devices can be shared currently (e.g., video, audio) [5] and such platform support is not widely available today [6, 17].

CPU and Memory Partitioning. Lockdown partitions the CPU in time by only allowing one environment to execute at a time. The available physical memory in the system is partitioned into three areas: the Lockdown memory region, the untrusted environ-

ment's memory region, and the trusted environment's memory region[4]. Lockdown employs Nested Page Tables (NPT)[5] [10] 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.

Device Partitioning. With hyper-partitioning, both the untrusted and trusted environments use the same set of physical devices. Devices that do not store persistent data, such as video, audio, and input devices can be partitioned by saving and restoring their states across environment switches. 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 devices and/or partitions. For example, Lockdown can assign a different hard disk to each environment. Alternatively, Lockdown can assign a different partition on the same hard disk to each environment. The challenge is to save and restore device state in a device agnostic manner, and to partition storage devices without virtualizing them, while providing strong isolation that cannot be bypassed by a malicious OS.

Lockdown leverages the Advanced Configuration and Power-management Interface (ACPI) [14] to save and restore device states while partitioning non-storage devices. The ACPI specification defines an ACPI subsystem (system BIOS and chipset) and an Operating System 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. ACPI 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). Figure 2a shows how an OSPM handles ACPI Sleep States S3 and S4. 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 subsystem 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. Note that with this scheme, Lockdown does not need to include any device drivers or interpose on device operations. The OS contains all the required drivers that deal directly with the devices for normal operation and for saving and restoring device states.

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 stor-

[4] An implementation using ACPI S4 state for hyper-partitioning (§ 6), requires only two memory regions, Lockdown and the current environment (untrusted or trusted) since ACPI S4 results in the current environment's memory contents being saved and restored from the disk

[5] Also termed as Extended Page Tables on Intel platforms

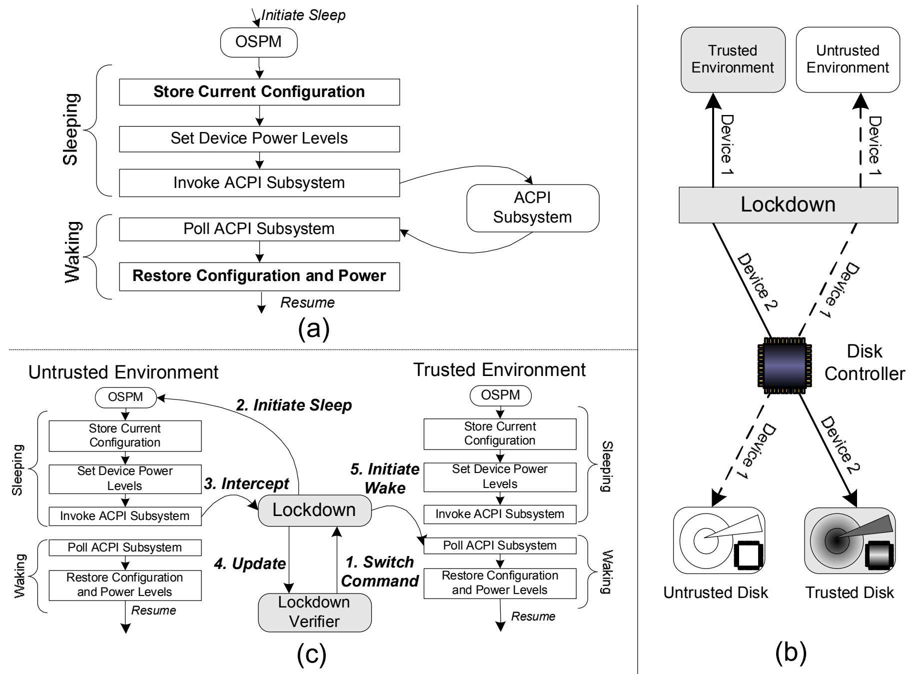


Fig. 2. Hyper-Partitioning. **(a)** *Lockdown leverages the Advanced Configuration and Power-management Interface (ACPI) OS sleep mechanism to partition (by saving and restoring states) non-storage system devices while being device agnostic.* **(b)** *Storage devices (e.g., disk) are partitioned by intercepting the device selection requests and redirecting device operations to the appropriate device, based on the current environment.* **(c)** *Environment switching is performed upon receiving a command from the Lockdown Verifier. The OS ACPI sleep steps are modified by Lockdown to transition between environments (untrusted and trusted).*

age 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 2b). 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 (§ 7) shows that interposing on device/sector selection has a minimal effect on performance. Since we assume the BIOS is trusted (§ 2), we can be sure that Lockdown will always be started first, and hence will always maintain its protections over the trusted disk.

Environment Switching. Lockdown performs an environment switch by transitioning the current environment to sleep and waking up the other. Figure 2c shows the steps taken for an environment switch, assuming the user starts in the untrusted environment. When the user toggles the switch on the trusted 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.

3.2 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. 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, as explained below.

Approved Code Execution. For non-firmware code, Lockdown uses Nested Page Tables (NPT) 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 (§ 3.2 describes how this list is established). Execution is permitted only if this check succeeds. Previous work enforces a similar policy only on the kernel [37], or uses it to determine what applications are running [21]. 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 installed hardware and enumerates all system and device firmware code regions. It assumes this code has not yet been tampered with and uses NPTs to prevent either environment from writing to these regions.

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, or a network-based attacker may perform SSL-based attacks (e.g., tricking a user into accepting a bogus certificate). While approved code execution prevents many code-based attacks, the trusted environment may still be vulnerable to script-based attacks (e.g., Javascript) and return-oriented programming attacks [38].

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. 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 proxy network driver that Lockdown installs in the guest OS. This driver forwards packets to Lockdown, which analyzes the packets and then forwards them to the physical network interface. The trusted environment can use a distinct physical network interface or reuse the same interface of the untrusted environment for network

communication (since the environments run non-concurrently). In both cases the Lockdown hypervisor will need to include the network driver for the physical interface. A simpler approach is to perform network access (either wireless or wired) using the Lockdown Verifier. In this case, the Lockdown hypervisor does not need to contain any network driver but simply forwards the packets to the verifier.

Lockdown uses packet analysis to determine which network packets are permitted. One approach, with the argument that any site with sensitive data should be using SSL to protect it in transit, would be to allow only SSL and DNS network packets to pass through to trusted sites. All other packets are dropped. 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 the following section). 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-based attacks are forestalled by SSL certificate verification. From a technical perspective, supporting other network protocols such as SSH is also possible.

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 can also trust that same organization to vet trusted software and websites. 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 the security-sensitive companies a user interacts 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 selects 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.

4 External Verification and Trusted Path

While the reference monitor (i.e., the hypervisor or virtual machine monitor) in a red/-green system 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, she might be tricked into performing security-sensitive operations in the untrusted environment. Below, we show how to eliminate such attacks by using a simple, external device to control the environment switching and to display the result of the switch to the user. We also show how the

external device can verify that it is interacting with a correct version of the red/green system, preventing malware from misleading the device.

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: (i) **Correct Operation**: Software executing on the Lockdown Verifier must be robust against compromise. By minimizing the code for the verifier, we make it amenable to formal analysis;(ii) **Minimal Input Capabilities**: To minimize complexity (and hence user confusion), we wish to minimize the number of input options; and (iii) **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 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. The verifier can also create an alarm buzz if it is unable to verify the correctness of the reference monitor (e.g., Lockdown) or if the system encounters a fatal error.

Secure Channel. To accurately verify the state of the system (trusted or untrusted), the Lockdown Verifier must be able to communicate securely with the red/green reference monitor (i.e., the hypervisor or virtual machine monitor). More precisely, it should not be possible for an adversary to impersonate or undetectably modify the reference monitor. We can achieve this goal using a combination of CPU protections and hardware attestation via a TPM [44]. To create a secure channel for communicating with the Lockdown Verifier, the reference monitor uses CPU protections to reserve a USB controller and to prevent both environments from accessing it. We use USB as an interface as it is intuitive for users and eliminates the need for an external power source for the verifier. To convince the Lockdown Verifier that it is communicating with the correct reference monitor, we use TPM-based attestation. Initially, the reference monitor is started using a *measured launch* operation [15, 7] which securely records a hash of the reference monitor's code in the TPM. When the verifier is connected to the system, it sends a challenge (a cryptographic nonce) to the reference monitor. The reference monitor uses the TPM to generate a quote (essentially a signed statement describing the software state of the system) that it securely transmits to the Lockdown Verifier using the reserved USB controller. The Lockdown Verifier then checks the attestation based on the TPM public key (setup during installation). 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 the reference monitor is back in control and to learn which environment is currently active. Note that the measured launch operation coupled with the TPM-based attestation and the reserved USB controller/channel eliminates the need to setup and share a secret key between the reference monitor and the Lockdown Verifier.

5 Security Analysis

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 hardware DMA protections to prevent DMA-based reads and writes to sensitive areas, and it ensures that all devices are reset during an environment switch. 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 [46, 12], but eventually, we anticipate a trusted environment for each trusted site.

Trusted Path. Lockdown is designed to create a trusted path to the user, i.e., to 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 (§ 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 and a guarantee that the hardware memory protections are in place.

5.1 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 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, users 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, users can obtain strong assurance if they elect to participate.

6 Implementation

We implemented a complete prototype of Lockdown on both AMD and Intel x86 platforms with Windows 2003 Server as the OS in both the trusted and untrusted environments. To demonstrate that Lockdown's hyper-partitioning is a generic primitive that works 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 on the AMD platform.

Our Lockdown prototype consists of a Lockdown Loader and the Lockdown Runtime. The SKINIT instruction is used to perform a *late-launch* [7] operation which ensures that the Lockdown Loader runs in a hardware-protected environment and that its measurement (cryptographic hash) is stored in the TPM's Platform Configuration Register (PCR) 17. The trusted Lockdown Loader loads the Lockdown Runtime and protects the Lockdown Runtime's memory region from DMA reads and writes (using AMD's Device Exclusion Vector [7]). 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 19. The Lockdown Loader then initializes the USB controller on the host for communication with the Lockdown Verifier, creates the Nested Page Tables [10] for the trusted and untrusted environments and transfers control to the Lockdown Runtime. 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 § 4. The Lockdown Runtime launches the environment currently indicated on the Lockdown Verifier in a hardware virtual machine, 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, and protection of the trusted environment is described below.

6.1 Hyper-Partitioning

To implement hyper-partitioning for non-storage devices under the Windows OS, Lockdown makes use of the ACPI S4 (hibernate) sleep state. ACPI S3 (standby) would offer faster switching times, but 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. Memory and storage device partitioning are described below.

Memory. In our current implementation (on systems with 4 GB of physical memory), Lockdown reserves 186 MB for itself and 258 MB for the system's firmware. The rest of physical memory is available to the trusted or untrusted environments. Isolation between the environments and Lockdown is maintained by using Nested Page Tables; the page-table 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.

Storage Devices. Our prototype can assign a different hard drive to each environment (trusted and untrusted), or it can partition a single hard drive into separate regions for each environment. Lockdown assigns each environment its own hard drive by intercepting read and write accesses to the ATA/SATA drive-select and command port (e.g., 0x1F6/7). This 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. Lockdown isolates partitions within a single disk by intercepting write accesses to the ports which are required to set the LBA (Logical Block Address) sector addresses (e.g., ports 0x1F3/4/5) and the sector count (e.g., port 0x1F2) in addition to the command port. When a sector read or write command is initiated by the environments using the command port, Lockdown verifies that the sector LBA address and count are within limits of the partition of the current environment before forwarding the command to the disk controller.

Environment Switching. Lockdown establishes control over the system's ACPI modes by intercepting the trusted and untrusted environments' attempts to access the ACPI Sleep and Status registers. The Lockdown Runtime determines the I/O location of these registers by parsing the ACPI Fixed Address Descriptor Table. 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 to the ACPI Sleep Register. Lockdown captures this write and instead clears the switch flag and up-

dates 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.

6.2 Protecting the Trusted Environment

Approved Code Execution. To enforce approved code execution, Lockdown uses page-level code hashing, similar to the approach used by previous work [21, 37]. Prior to executing the trusted environment, Lockdown sets its Nested Page Table (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 corresponding 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 original executable.
Network Protection. To provide network protection for the trusted environment, we developed an untrusted network driver for Windows, and an SSL Protocol Analyzer within Lockdown. The analyzed network packets are sent to the Lockdown Verifier using Lockdown's USB driver, and ultimately out to the network. The Lockdown Verifier has an ethernet port and a dedicated network chipset. Our OS-level network driver sends and receives network packets to and from the SSL Protocol Analyzer via a hypercall. Our SSL Protocol Analyzer is based on `ssldump`[6]. We added support for SSL session tracking and event handling depending on the SSL packet (e.g., Certificate, ServerHello). The certificate handler is used to compare a site's SSL certificate against Lockdown's list of trusted-site certificates.

6.3 External Verification and Trusted Path

We built the Lockdown Verifier using a low-cost LPC 2148 development board. The board is equipped with a 60Mhz ARM7 CPU, 512 KB flash, 42 KB RAM and an ethernet chipset/port. We attached a red and a green LED, a switch, and a buzzer to the board. The Lockdown Runtime contains USB and TPM drivers that communicate with the Lockdown Verifier and the host system TPM respectively. The 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 19, as well as the nonce that was provided. The verifier uses the TPM's public-key (installed during Lockdown's installation) 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.

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

Hypervisor/Micro-kernel	TCB	Supports *any* unmodified OS	Free from device Virtualization
Lockdown	**10KLOC**	✓	✓
L4Ka-Pistachio	25KLOC	×	×
NOVA	36KLOC	×	×
VMWare ESXi	200KLOC	✓	×
Xen + Linux	400KLOC	✓	×
KVM + Linux + QEMU	470KLOC	✓	×
Hyper-V + Windows	5000KLOC	✓	×

Fig. 3. Lockdown's TCB and Features. *Comparison with popular, general-purpose hypervisors and micro-kernels. Note: We assume a Linux kernel with only the required device drivers for a host platform. For our test system this came up to 300KLOC. As VMWare, Hyper-V and Windows are closed-source, we rely on publicly available information to estimate their SLOC [1, 20, 3]. QEMU's TCB with only x86 support is around 150KLOC.*

7 Evaluation

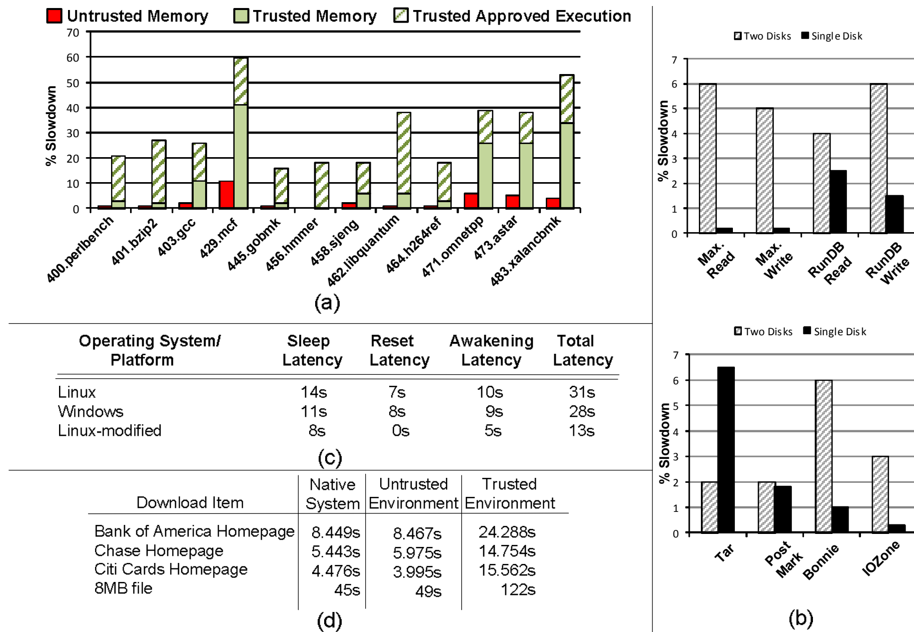
7.1 Trusted Computing Base (TCB)

Like all security systems, Lockdown must assume the correctness and security of its core components. This assumption is more likely to hold if we reduce the amount of code that must be trusted, keep the design simple and minimize the external interface. This reduces opportunities for bugs and makes the code more amenable to formal analysis. Lockdown's total TCB is only 10KLOC, placing Lockdown within the reach of formal verification and manual audit techniques. Lockdown's design is simple and greatly reduces the attack surface. Lockdown does not expose any interface while the untrusted environment is running and interposes only on memory and disk accesses. When the trusted environment is executing, Lockdown also intercepts execution on memory pages for approved code execution. These operations are handled transparently via well-defined CPU intercepts. Further, in the trusted environment, Lockdown exposes a single hypercall interface to the guest OS network driver. The arguments to this hypercall interface are the type of operation (read or write), the network packet length and the packet data which are sanity checked by the Lockdown Runtime.

Lockdown's TCB compares favorably with other popular hypervisors and VMMs (Figure 3), 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 device access and administrative purposes, dramatically increasing their TCBs. While VMware ESXi 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 supported platforms. Only L4Ka-Pistachio [2] and NOVA [40] approach Lockdown's TCB size. However, the L4Ka-Pistachio requires non-trivial OS porting and cannot run OSes such as Windows. While NOVA is designed to run an unmodified OS, it currently only runs Linux due to its minimal device support; its virtualization architecture also requires device drivers to be written from scratch.

7.2 Performance Measurements

We use our prototype to determine Lockdown's performance on a recent laptop with a dual-core AMD Phenom-II N620 CPU, 4GB RAM, 250GB SATA hard disk, a v1.2 TPM and two USB controllers.



Operating System/ Platform	Sleep Latency	Reset Latency	Awakening Latency	Total Latency
Linux	14s	7s	10s	31s
Windows	11s	8s	9s	28s
Linux-modified	8s	0s	5s	13s

Download Item	Native System	Untrusted Environment	Trusted Environment
Bank of America Homepage	8.449s	8.467s	24.288s
Chase Homepage	5.443s	5.975s	14.754s
Citi Cards Homepage	4.476s	3.995s	15.562s
8MB file	45s	49s	122s

Fig. 4. 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)** *Environment switch latency,* **(d)** *Network-protection latency.*

CPU and Memory Overhead. Lockdown's use of Nested Page Tables (NPT) 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 [10]. Lockdown also adds overhead to code execution in the trusted environment due to its verification of approved code. 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 overhead added by the NPTs. Figure 4a shows Lockdown's overhead as a percentage of the native system's performance. In the untrusted environment, performance is only slightly worse than native (3% average overhead). The trusted environment adds considerably more overhead (15–59%). 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 page size. In the untrusted environment, we use the 2 MB pages to improve performance. However, in the trusted environment, we also use NPTs to check for approved code at a page granularity, and hence the trusted environment must use the smaller 4 KB pages, making it less efficient. Nonetheless, this performance is appropriate for infrequent tasks, such as online banking, that are less performance intensive.

Storage Overhead. To partition the system's disks between the trusted and untrusted environments, Lockdown intercepts both environments' drive/sector 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 (10000 files and 10000 transactions), IoZone (2GB file), Bonnie (2GB file), and `tar` (on the Windows installation folder). Figure 4b 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%.

Environment Switch Latency. We split Lockdown's environment switch latency 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 4c shows the measurements for Lockdown's environment switch latency. The switch currently requires 31 seconds on Windows and 13–28 seconds on Linux. While longer than ideal, we expect users to swap between the two environments relatively infrequently. Our results indicate that the direction of the switch has a relatively small impact on the switching time. The reset latency 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 (more than 25% of the switch time), completely reinitializing the CPU, chipset, memory and devices. BIOS vendors are actively working to greatly reduce this latency with the Unified Extensible Firmware Interface (UEFI) [33]. The reset process can also 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. Further, if OS device drivers are architected in a way that no prior state assumptions are made about the device (as in Linux), we can completely eliminate the reset latency by modifying the kernel to restart itself without issuing a platform reset.

Network Protection Overhead. Since Lockdown interposes on the trusted environment's network connections, we expect performance to be worse in the trusted environment. Since the untrusted environment has full access to the network interface, it 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 8 MB file. We averaged the download times over 5 runs, clearing the Firefox cache each time. Figure 4d summarizes our results. As expected, the untrusted environment's performance is equivalent to the native system (within experimental error). The trusted environment takes longer, because all network packets traverse via the SSL protocol analyzer and over USB through the Lockdown Verifier. Fortunately, most security-sensitive online transactions involve small network transmissions that makes the download times usable.

Comparison with Virtualization. Finally, we compare Lockdown's performance with traditional virtualization approaches. We choose the popular Xen (3.4.2) hypervisor

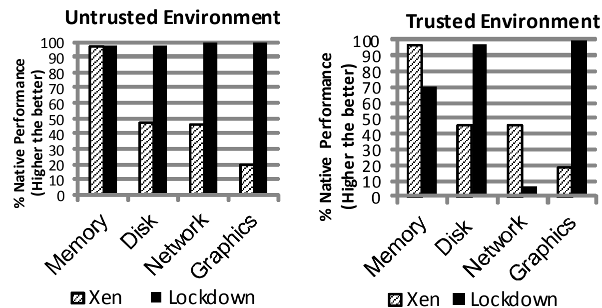


Fig. 5. Comparison of Partitioning (Lockdown) with Virtualization (Xen). *Partitioning penalizes only the trusted environment while virtualization treats both environments as equal and imposes similar overheads.*

for our comparison even though it does not provide the same high level of protection as Lockdown. We instantiate two virtual machines (VMs) with identical configuration for the untrusted and trusted environments within Xen. For measurement purposes, we benchmark the core platform subsystems comprising the memory, disk, network and graphics. To measure the memory overhead we use benchmarks from the SPECint 2006 suite. We use Tar, Bonnie, Postmark and IoZone (with the same parameters as discussed previously) as our disk macrobenchmarks. We use Flashget to measure the average network throughput and the PassMark 2D benchmark suite to measure the graphics performance. Figure 5 shows the performance of Lockdown and Xen as a percentage of the native system's performance, for both the untrusted and trusted environments[7]. Our results show that virtual machine monitors in general (including Xen) virtualize the underlying platform resources and therefore introduce similar performance latency in all VMs (untrusted and trusted). The slowdown is particularly high for the disk (54%), network (55%) and graphics (81%) subsystems. In contrast, Lockdown only imposes restrictions for the trusted partition and lets the untrusted partition run near native speed (only a 3% average overhead for memory and 2-7% overhead for disk). These results demonstrate the efficiency advantage of partitioning vs virtualization. The performance degradation in the trusted partition is higher for memory and network due to Lockdown's approved execution and network protection mechanisms. However, for the disk and graphics subsystem the overhead introduced is less than Xen.

8 Related Work

Following our earlier preliminary work on hyper-partitioning [45], systems such as NoHype and SecureSwitch, like Lockdown, advocate the use of partitioning in order to minimize TCB. However, both these systems are fairly different from Lockdown in many ways.

NoHype [17] uses static partitioning of devices leveraging specialized platform hardware capabilities such as Single-Root I/O Virtualization (SR-IOV) [29] and aims

[7] Note that we could not compare Xen with direct device assignment [4], as that requires special platform support that is not widely available today [6, 29]. Futher, not all devices can be assigned currently (e.g., video, audio) [5, 29].

to run commodity operating systems. However, SR-IOV capabilities are only found in few high-end server platforms today. Also, NoHype does not have a particular operating model in mind and treats all VMs equally, as opposed to Lockdown which has a particular operating model in mind, i.e., trusted and untrusted, and takes steps to keep the trusted partition more secure. Furthermore, NoHype lacks the trusted path provided by Lockdown for assessing and switching between environments.

SecureSwitch [41] attempts to provide isolation between untrusted and trusted OSes with low switch times. However, unlike Lockdown, it does not provide any trusted environment protections (approved execution and trusted network access) or user-verifiable trusted path for input and output. In addition, their isolation mechanism requires changing the system BIOS and relies on specialized hardware (southbridge DIMM isolation, dual disk controllers with disk locking feature and motherboard jumpers for switching) that are not commodity.

In contrast, Lockdown represents a *complete end-to-end solution* of a red/green system on *commodity platforms* (without specialized platform hardware) and does not require changes to the system BIOS or the OSes. The Lockdown Verifier is an *external* USB device that communicates verified system state ("red" or "green") to the user and enables trustworthy switching between the "red" and "green" environments.

Systems such as NetTop [27], HAP [28], NGSCB [30], Terra [13], Qubes [43], Virtics [34], or Overshadow [11] use virtualization to isolate code running at different security levels. As discussed in § 7, virtualization allows rapid switching (orders of magnitude faster than Lockdown) between multiple environments. However, virtualization increases side-channels that may leak sensitive information. Device virtualization also degrades performance and increases the amount of trusted code by orders of magnitude. Several proposals use virtualization to isolate one web application from another [46, 12], 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 [42], Nizza [39], Flicker [23, 25, 24] and TrustVisor [22] 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.

OS level approaches such as Apiary [35] and WindowBox [8] modify the OS kernel or leverage specific OS features (e.g., FreeBSDs jails) to enforce application specific execution containers. However, as these containers share the same OS kernel, memory and system devices, any vulnerability within the OS can be exploited to subvert the protection mechanisms.

9 Conclusion

We evaluated a new point in the design space of red/green systems by using partitioning, rather than virtualization to share critical system resources and devices. Our implementation and results indicate that partitioning offers increased security (by reducing the size of the reference monitor to 10K lines of code and by reducing opportunities for side channels) and performance (by giving the untrusted environment unfettered access to system devices) at the cost of slow switching times (on current systems). Determining whether the switching times can be reduced to an acceptable level, or whether the

security and performance benefits can be adopted by virtualization-based approaches, are interesting directions for future research.

Acknowledgement

This research was supported by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389, W911NF-09-1-0273, W911NF10C0037, and MURI W 911 NF 0710287 from the Army Research Office, and by support from NSF under awards CCF-0424422 and CNS-0831440. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, NSF or the U.S. Government or any of its agencies.

References

1. Vmware esx server node evaluators guide. http://www.vmware.com/pdf/esx_vin_eval.pdf.
2. The l4ka project. http://www.l4ka.org, 2011.
3. Source lines of code. http://en.wikipedia.com/wiki/Source_lines_of_code, 2011.
4. Xen pcipassthrough. http://wiki.xensource.com/xenwiki/XenPCIpassthrough, Oct. 2011.
5. Xen vgapassthrough. http://wiki.xensource.com/xenwiki/XenVGAPassthrough, Oct. 2011.
6. Xen vtdhowto. http://wiki.xensource.com/xenwiki/VTdHowTo, Oct. 2011.
7. Advanced Micro Devices. AMD64 architecture programmer's manual: Volume 2: System programming. AMD Publication no. 24594 rev. 3.11, Dec. 2005.
8. D. Balfanz and D. R. Simon. Windowbox: A simple security model for the connected desktop. In *In Proceedings of the 4 th USENIX Windows Systems Symposium*, 2000.
9. D. J. Bernstein. Cache-timing attacks on aes. http://cr.yp.to/papers.html, Apr. 2005.
10. R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating two-dimensional page walks for virtualized systems. In *ASPLOS*, Mar. 2008.
11. 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*, 2008.
12. R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen. A safety-oriented platform for web applications. In *IEEE S&P*, pages 350–364, May 2006.
13. T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *SOSP*, Oct. 2003.
14. Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification. Revision 3.0b, Oct. 2006.
15. Intel Corporation. Trusted execution technology – preliminary architecture specification and enabling considerations. Document number 31516803, Nov. 2006.
16. P. Karger and D. Safford. I/O for virtual machine monitors: Security and performance issues. *IEEE Security and Privacy*, 6(5):16–23, 2008.
17. E. Keller, J. Szefer, J. Rexford, and R. B. Lee. Nohype: virtualized cloud infrastructure without the virtualization. In *International symposium on Computer architecture*, 2010.
18. B. Lampson. A note on the confinement problem. *Comm. of the ACM*, 16(10), 1973.
19. B. Lampson. Usable security: How to get it. *Comm. of the ACM*, 52(11), 2009.
20. D. Leinenbach and T. Santen. Verifying the microsoft hyper-v hypervisor with vcc. In *International Symposium on Formal Methods*, 2009.
21. L. Litty, H. A. Lagar-Cavilla, and D. Lie. Hypervisor support for identifying covertly executing binaries. In *Proceedings of the USENIX Security Symposium*, 2008.
22. J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB reduction and attestation. In *IEEE S&P*, May 2010.

23. J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *EuroSys*, Apr. 2008.
24. J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri. Minimal TCB code execution (extended abstract). In *IEEE Symposium on Security and Privacy*, May 2007.
25. J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri. How low can you go? Recommendations for hardware-supported minimal TCB code execution. In *ACM ASPLOS*, Mar. 2008.
26. J. M. McCune, A. Perrig, A. Seshadri, and L. van Doorn. Turtles all the way down: Research challenges in user-based attestation. In *USENIX Workshop on Hot Topics in Security*, 2007.
27. R. Meushaw and D. Simard. Nettop: Commercial technology in high assurance applications. *VMware Tech Trend Notes*, 9(4):1–8, 2000.
28. National Security Agency. High assurance platform program. Online at `http://www.nsa.gov/ia/programs/h_a_p/index.shtml`, Jan. 2009.
29. PCI SIG. Single Root I/O Virtualization and Sharing Specification. V. 1.1, 2010.
30. M. Peinado, Y. Chen, P. England, and J. Manferdelli. NGSCB: A trusted open system. In *In 9th Australian Conference on Information Security and Privacy*, 2004.
31. C. Percival. Cache missing for fun & profit. In *BSDCan*, 2005.
32. Phoenix Technologies. TrustedCore: Foundation for secure CRTM and BIOS implementation. https://forms.phoenix.com/whitepaperdownload/docs/trustedcore_wp.pdf, 2006.
33. Phoenix Technologies. Transitioning the Plug-In Industry from Legacy to Unified Extensible Firmware Interface (UEFI). Intel Developer Forum, Sept. 2009.
34. M. Piotrowski and A. D. Joseph. Virtics: A system for privilege separation of legacy desktop applications. Technical Report UCB/EECS-2010-70, EECS Department, University of California, Berkeley, May 2010.
35. S. Potter and J. Nieh. Apiary: Easy-to-use desktop application fault containment on commodity operating systems. In *USENIX Annual Technical Conference*, 2010.
36. T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *ACM CCS*, 2009.
37. 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.
38. H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *ACM CCS*, 2007.
39. L. Singaravelu, C. Pu, H. Haertig, and C. Helmuth. Reducing TCB complexity for security-sensitive applications: Three case studies. In *EuroSys*, 2006.
40. U. Steinberg and B. Kauer. Nova: A microhypervisor-based secure virtualization architecture. In *EuroSys*, 2010.
41. K. Sun, J. Wang, F. Zhang, and A. Stavrou. Secureswitch: Bios-assisted isolation and switch between trusted and untrusted commodity oses. In *NDSS*, 2012.
42. R. Ta-Min, L. Litty, and D. Lie. Splitting interfaces: Making trust between applications and operating systems configurable. In *OSDI*, 2006.
43. The Qubes OS. http://qubes-os.org/Home.html.
44. Trusted Computing Group. Trusted Platform Module Main Specification. V. 1.2, 2007.
45. A. Vasudevan, B. Parno, N. Qu, V. D. Gligor, and A. Perrig. Lockdown: A safe and practical environment for security applications. Technical Report CMU-CyLab-09-011, CyLab, Carnegie Mellon University, July 2009.
46. 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. In *USENIX Security Symposium*, 2009.