Protecting Confidential Data on Personal Computers with Storage Capsules

Kevin Borders, Eric Vander Weele, Billy Lau, and Atul Prakash
University of Michigan
Ann Arbor, MI, 48109
{kborders, ericvw, billylau, aprakash}@umich.edu

Abstract

Protecting confidential information is a major concern for organizations and individuals alike, who stand to suffer huge losses if private data falls into the wrong hands. One of the primary threats to confidentiality is malicious software on personal computers, which is estimated to already reside on 100 to 150 million machines. Current security controls, such as firewalls, anti-virus software, and intrusion detection systems, are inadequate at preventing malware infection. This paper introduces Storages Capsules, a new approach for protecting confidential files on a personal computer. Storage Capsules are encrypted file containers that allow a compromised machine to securely view and edit sensitive files without malware being able to steal confidential data. The system achieves this goal by taking a checkpoint of the current system state and disabling device output before allowing access a Storage Capsule. Writes to the Storage Capsule are then sent to a trusted module. When the user is done editing files in the Storage Capsule, the system is restored to its original state and device output resumes normally. Finally, the trusted module declassifies the Storage Capsule by re-encrypting its contents, and exports it for storage in a low-integrity environment. This work presents the design, implementation, and evaluation of Storage Capsules, with a focus on exploring covert channels.

1. Introduction

Traditional methods for protecting confidential information rely on upholding system integrity. If a computer is safe from hackers and malicious software (malware), then so is its data. Ensuring integrity in today's interconnected world, however, is exceedingly difficult. Trusted computing platforms such as Terra [8] and trusted boot [26] try to provide this integrity by verifying software. Unfortunately, these platforms are rarely deployed in practice and most software continues to be unverified. More widely-applicable security tools, such as firewalls, intrusion detection systems, and anti-virus software, have been unable to combat malware, with 100 to 150 million infected machines running on the Internet today according to a recent estimate [34]. Security mechanisms for personal computers simply cannot rely on keeping high integrity. Storage Capsules address the need for access to confidential data from compromised personal computers.

There are some existing solutions for preserving confidentiality that do not rely on high integrity. One example is mandatory access control (MAC), which is used by Security-Enhanced Linux [23]. MAC can control the flow of sensitive data with policies that prevent entities that read confidential information from communicating over the network. This policy set achieves the goal of preventing leaks in the presence of malware. However, defining correct policies can be difficult, and they would prevent most useful applications from running properly. For example, documents saved by a word processor that has ever read secret data could not be sent as e-mail attachments. Another embodiment of the same principle can be seen in an "air gap" separated network where computers are physically disconnected from the outside world. Unplugging a compromised computer from the Internet will stop it from leaking information, but doing so greatly limits its utility. Both mandatory access control with strict outbound flow policies and air gap networks are rarely used outside of protecting classified information due to their severe impact on usability.

This paper introduces Storage Capsules, a new mechanism for protecting sensitive information on a local computer. The goal of Storage Capsules is to deliver the same level of security as a mandatory access control system for standard applications running on a commodity operating system. Storage Capsules meet this requirement by enforcing policies at a system-wide level using virtual machines. The user's system can also downgrade from high-secrecy to low-secrecy by reverting to a prior state using virtual machine snapshots. Finally, the system can obtain updated Storage Capsules from a declassification component after returning to low secrecy.

Storage Capsules are analogous to encrypted file containers from the user's perspective. When the user opens a Storage Capsule, a snapshot is taken of the current system state and device output is disabled. At this point, the system is considered to be in *secure mode*. When the user is finished editing files in a Storage Capsule, the system is reverted to its original state – dis-

carding all changes except those made to the Storage Capsule – and device output is re-enabled. The storage capsule is finally re-encrypted by a trusted component.

Storage Capsules guarantee protection against a compromised operating system or applications. Sensitive files are safe when they are encrypted *and* when being accessed by the user in plain text. The Capsule system prevents the OS from leaking information by erasing its entire state after it sees sensitive data. It also stops covert communication by fixing the Storage Capsule size and completely re-encrypting the data every time it is accessed by the OS. Our threat model assumes that the primary operating system can do anything at all to undermine the system. The threat model also assumes that the user, hardware, the virtual machine monitor (VMM), and an isolated secure virtual machine are trustworthy. The Capsule system protects against covert channels in the primary OS and Storage Capsules, as well as many (though not all) covert channels at lower layers (disk, CPU, etc.). One of the contributions of this paper is identifying and suggesting mitigation strategies for numerous covert channels that could potentially leak data from a high-secrecy VM to a low-secrecy VM that runs after it has terminated.

We evaluated the impact that Storage Capsules have on the user's workflow by measuring the latency of security level transitions and system performance during secure mode. We found that for a primary operating system with 512 MB of RAM, transitions to secure mode took about 4.5 seconds, while transitions out of secure mode took approximately 20 seconds. We also compared the performance of the Apache build benchmark in secure mode to that of a native machine, a plain virtual machine, and a virtual machine running an encryption utility. Overall, Storage Capsules added 38% overhead compared to a native machine, and only 5% compared to a VM with encryption software. The common workload for a Storage Capsule is expected to be much lighter than an Apache build. In many cases, it will add only a negligible overhead.

The main contribution of this work is a system that allows safe access to sensitive files from a normal operating system with standard applications. The Capsule system is able to switch modes within one OS rather than requiring separate operating systems or processes for different modes. This paper also makes contributions in the understanding of covert channels in such a system. In particular, it looks at how virtualization technology can create new covert channels and how previously explored covert channels behave differently when the threat model is a low-security virtual machine running after a high-security virtual machine.

It is important to keep in mind that Storage Capsules do not protect integrity. There are a number of attacks that they cannot prevent. If malicious software stops the user from ever entering secure mode by crashing, then the user might be coerced into accessing sensitive files without Storage Capsules. Furthermore, malware can manipulate data to present false information that tricks the user into doing something erroneously, such as placing a stock transaction. These attacks are beyond the scope of this paper.

The remainder of this paper is laid out as follows. Section 2 discusses related work. Section 3 gives an overview of the usage model, the threat model, and design alternatives. Section 4 outlines the system architecture. Section 5 describes the operation of Storage Capsules. Section 6 examines the effect of covert channels on Storage Capsules. Section 7 presents evaluation results. Finally, section 8 concludes and discusses future work.

2. Related Work

The Terra system [8] provides multiple security levels for virtual machines using trusted computing technology. Terra verifies each system component at startup using a trusted platform model (TPM) [29], similar to trusted boot [26]. However, Terra allows unverified code to run in low-security virtual machines. One could imagine a configuration of Terra in which the user's primary OS runs inside of a low-integrity machine, just like in the Capsule system. The user could have a separate secure VM for decrypting, editing, and encrypting files. Assuming that the secure VM always has high integrity, this approach would provide comparable security and usability benefits to Storage Capsules. However, Terra only ensures a secure VM's integrity at startup; it does not protect running software from exploitation. If this secure VM ever loads an encrypted file from an untrusted location, it is exposed to attack. All sources of sensitive data (e-mail contacts, web servers, etc.) would have to be verified and added to the trusted computing base (TCB), bloating its size and impacting both management overhead and security. Furthermore, the user would be unable to safely include data from untrusted sources, such as the internet, in sensitive files. The Capsule system imposes no such headaches; it can include low-integrity data in protected files, and only requires trust in local system components to guarantee confidentiality.

There has been extensive research on controlling the flow of sensitive information inside of a computer. Intra-process flow control techniques aim to verify that individual applications do not inadvertently leak confidential data [6, 22]. However, this does not stop mali-

cious software that has compromised a computer from stealing data at the operating system or file system level. Another approach for controlling information flow is at the process level with a mandatory access control (MAC) system like SELinux [23]. MAC involves enforcing access control policies on high-level objects (typically files, processes, etc.). However, defining correct policies can be quite difficult [15] even for a fixed set of applications. MAC would have a hard time protecting personal computers that download and install programs from the internet. Very few computers use mandatory access control currently, and it is not supported by Microsoft Windows, a popular operating system for personal computers. Storage Capsules employ a similar approach to MAC systems, but do so at a higher level of granularity (system-wide) using virtual machine technology. This allows Storage Capsules to provide more practical security for commodity operating systems without requiring modification.

There are a number of security products available for encrypting and protecting files on a local computer, including compression utilities [25, 35] and full disk encryption software [1, 7, 20, 31]. The goal of file encryption is to facilitate file transmission over an untrusted medium (e.g., an e-mail attachment), or protect against adversarial access to the storage device (e.g., a lost or stolen laptop). File encryption software does safeguard sensitive information while it is decrypted on the end host. Malicious software that has control of the end host can steal confidential data or encryption keys. Capsule also uses file encryption to allow storage in an untrusted location, but it maintains confidentiality while sensitive data is decrypted on the end host.

Storage Capsules rely on the virtual machine monitor as part of the trusted computing base. VMMs are commonly accepted as less complex and more secure than standard operating systems, with the Xen VMM having under 50,000 lines of code [36], compared to 5.7 million lines in the Linux 2.6 kernel [5]. These numbers are reinforced by actual vulnerability reports, with Xen 3.x only having 9 reports up to January 2009 [27], and the Linux 2.6.x kernel having 165 reports [28] in that same time period. VMMs are not invulnerable, but they have proven to be more robust than standard kernels.

Virtualization technology has many useful properties and features that make it a well-suited platform for Storage Capsules. Despite these advantages, Garfinkel et al. warn that virtualization has some shortcomings, especially when it comes to security [9]. Most importantly, have many branches and saved states makes patching and configuration much more difficult. A user might load an old snapshot that is vulnerable to infection by an Internet worm. The Capsule system does not suffer from these limitations because it is designed to have one primary VM with a fairly straight execution path. Transitions too and from secure mode are short-lived, and should have a minimal impact on patching and management tasks.

3. Overview

3.1 Storage Capsules from a User's Perspective

From the user's perspective, Storage Capsules are analogous to encrypted file containers provided by a program like TrueCrypt [31]. Basing the Capsule system off of an existing and popular program's usage model makes it easier to gain acceptance. The primary difference between Storage Capsules and traditional encryption software is that the system enters a secure mode before opening the Storage Capsule's contents. In this secure mode, network output is disabled and any changes that the user makes outside of the Storage Capsule will be lost. The user may still edit the Storage Capsule contents with his or her standard applications. When the user closes the Storage Capsule and exits secure mode, the system reverts to the state it was in before accessing sensitive data.

One motivating example for Storage Capsules is providing a secure journal. A person, call him Bob, may want to write a diary in which he expresses controversial political beliefs. Bob might regularly write in this journal, possibly pasting in news stories or contributions from others on the internet. Being a diligent user, Bob might store this document in an encrypted file container. Unfortunately, Bob is still completely vulnerable to spyware when he enters the decryption password and edits the document. Storage Capsules support the same usage model as normal encrypted file containers, but also deliver protection against spyware while the user is accessing sensitive data.

Storage Capsules have some limitations compared to encrypted file containers. These limitations are necessary to gain additional security. First, changes that the user makes outside of the encrypted Storage Capsule while it is open will not persist. This benefits security and privacy by eliminating all traces of activity while the container was open. Storage Capsules guarantee that the OS does not inadvertently hold information about sensitive files as described by Czeskis et al. for the case of TrueCrypt [4]. Unfortunately, any work from computational or network processes that may be running in the background will be lost. One way to remove this limitation would be to fork the primary virtual machine and

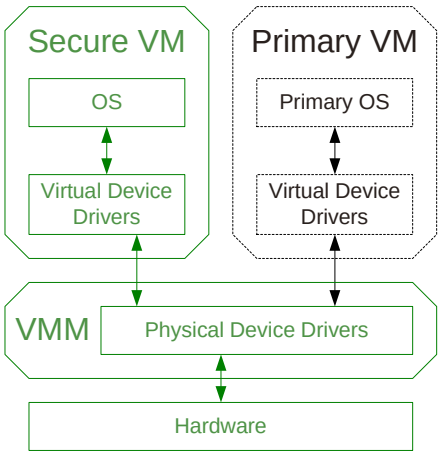

Figure 1. In the Storage Capsule architecture, the user's primary operating system runs in a virtual machine. The secure VM handles encryption and declassification. The dotted black line surrounding the primary VM indicates that it is *not* trusted. The other system components are trusted.

allow a copy of it to run in the background. Allowing low- and high-secrecy VMs to run at the same time, however, reduces security by opening up the door for a variety of covert channels.

3.2 Threat Model

Storage Capsules are designed to allow a compromised operating system to safely edit confidential information. However, some trusted components are necessary to provide security. Figure 1 shows the architecture of the Capsule system, with trusted components having solid lines and untrusted components having dotted lines. The user's primary operating system runs inside of a *primary VM*. Neither the applications, the drivers, nor the operating system are trusted in the primary VM; it can behave in any arbitrary manner. A *virtual machine monitor (VMM)* runs beneath the primary VM, and is responsible for mediating access to physical devices. The VMM is considered part of the trusted computing base (TCB). The Capsule system also relies on a *Secure VM* to save changes and re-encrypt Storage Capsules. This secure VM has only a minimal set of applications to service Storage Capsule requests, and has all other services blocked off with a firewall. The secure VM is also part of the TCB.

The user is also considered trustworthy in his or her intent. Presumably, the user has a password to decrypt each Storage Capsule and could do so using rogue software without going into secure mode and leak sensitive data. The user does not require full access to any trusted components, however. The main user interface is the primary VM, and the user should only interact with the Secure VM or VMM briefly using a limited UI. This prevents the user from inadvertently compromising a trusted component with bad input.

The threat model assumes that malicious software may try to communicate covertly within the primary VM. Storage Capsules are designed to prevent a compromised primary OS from saving data anywhere that will persist through a snapshot restoration. However, Storage Capsules do not guarantee that a malicious primary VM cannot store data somewhere in a trusted component, such as hardware or the VMM, in such a way that it can recover information after leaving secure mode. We discuss several of these covert channels in more depth later in the paper.

3.3 Designs that do not Satisfy Storage Capsule Goals

The first system design that would not meet the security goals laid out in our threat model is conventional file encryption software [1, 7, 20, 31]. Any information stored in an encrypted file would be safe from malicious software, or even a compromised operating system, while it is encrypted. However, as soon as the user decrypts a file, the operating system can do whatever it wants with the decrypted data.

Another design that would not meet the goals of Storage Capsules is the NetTop architecture [19]. With NetTop, a user has virtual machines with multiple security levels. One is for accessing high-secrecy information, and another for low-secrecy information, which may be connected to the internet. Depending on how policies are defined, NetTop either suffers from usability limitations or would have security problems. First assume that the high-secrecy VM must be able to read data from the low-secrecy VM to load files from external locations that are not part of the trusted computing base. Now, if the high-secrecy VM is prevented from writing anything back to the low-secrecy VM, then confidentiality is maintained. However, this prevents the user from making changes to a sensitive document, encrypting it, then sending it back out over a low-secrecy medium. This effectively makes everything read-only from the high-secrecy VM to the low-secrecy VM. The other alternative – letting the high-secrecy VM encrypt and declassify data – opens up a major security hole. Data that comes from the low-secrecy VM also might be malicious in nature. If the high-secrecy VM reads that information, its integrity – and the integrity of its encryption operations – may be compromised.

4. System Architecture

The Capsule system has two primary modes of operation: *normal mode* and *secure mode*. In normal mode, the computer behaves the same as it would without the Capsule system. The primary operating system has access to all devices and can communicate freely over the network. In secure mode, the primary OS is blocked from sending output to the external network or to devices that can store data. Furthermore, the primary operating system's state is saved prior to entering secure mode, and then restored when transitioning back to normal mode. This prevents malicious software running on the primary OS from leaking data from secure mode to normal mode.

The Capsule system utilizes virtual machine technology to isolate the primary OS in secure mode. Virtual machines also make it easy to save and restore system state when transitioning to or from secure mode. Figure 1 illustrates the architecture of the Capsule system. The first virtual machine, labeled Primary VM, contains the primary operating system. This VM is the equivalent of the user's original computer. It contains all of the user's applications, settings, and documents. This virtual machine may be infected with malicious software and is not considered trustworthy. The other virtual machine, labeled Secure VM, is responsible for managing access to Storage Capsules. The secure VM is trusted. The final component of the Capsule system shown in Figure 1 is the Virtual Machine Monitor (VMM). The VMM is responsible for translating each virtual device I/O request into a physical device request, and for governing virtual networks. As such, it can also block device I/O from virtual machines. The VMM has the power to start, stop, save, and restore entire virtual machines. Because it has full control of the computer, the VMM is part of the trusted computing base.

The Capsule system adds three components to the above architecture to facilitate secure access to Storage Capsules. The first is the *Capsule VMM module*, which runs as service inside of the VMM. The Capsule VMM module performs the following basic functions:

- Saves and restores snapshots of the primary VM
- Enables and disables device access by the primary VM
- Catches key escape sequences from the user
- Switches the UI between the primary VM and the secure VM

The Capsule VMM module executes operations as requested by the second component, the *Capsule server*, which runs inside of the secure VM. The Capsule server manages transitions between normal mode and secure mode. During secure mode, it also acts as a disk server, handling block-level read and write requests from the *Capsule viewer*, which runs in the primary VM. The Capsule server has dedicated interfaces for communicating with the Capsule viewer and with the Capsule VMM module. These interfaces are attached to separate virtual networks so that the viewer and VMM module cannot impersonate or communicate directly with each other.

The third component, the Capsule viewer, is an application that accesses Storage Capsules on the primary VM. When the user first loads or creates a new Storage Capsule, the viewer will import the file by sending it to the Capsule server. The user can subsequently open the Storage Capsule, at which point the viewer will ask the Capsule server to transition the system to secure mode. During secure mode, the viewer presents the contents of the Storage Capsule to the user as a new mounted partition. Block-level read and write requests made by the file system are forwarded by the viewer to the Capsule server, which encrypts and saves changes to the Storage Capsule. Finally, the Capsule viewer can retrieve the encrypted Storage Capsule by requesting an export from the Capsule server. The Capsule viewer is not trusted and may cause a denial-of-service at any time. However, the Capsule system is designed to prevent even a compromised viewer from leaking data from secure mode to normal mode.

5. Storage Capsule Operation

5.1 Storage Capsule File Format

A Storage Capsule is actually an encrypted partition that is mounted during secure mode. The Storage Capsule model is based on TrueCrypt [31] – a popular encrypted storage program. Like TrueCrypt, each new Storage Capsule is created with a fixed size. Storage Capsules employ XTS-AES – the same encryption scheme as TrueCrypt – which is the IEEE standard for data encryption [13]. In our implementation, the encryption key for each file is created by taking the SHA-512 hash of a user-supplied password. In a production system, it would be beneficial to employ other methods, such as hashing the password many times and adding a salt, to make attacks more difficult. The key could also come from a biometric reader (fingerprint reader, retina scanner, etc.), or be stored on a key storage device like a smart card. Storage Capsules operation does not depend on a particular key source.

With XTS-AES, a different tweak value is used during encryption for each data unit. A data unit can be one or more AES blocks. The Storage Capsule implementation

uses a single AES block for each data unit. In accordance with the IEEE 1619 standard [13], Storage Capsules use a random 128-bit starting tweak value that is incremented for each data unit. This starting tweak value is needed for decryption, so it is stored at the beginning of the file. Because knowledge of the tweak value does not weaken the encryption [18], it is stored in the clear.

5.2 Creating and Importing a Storage Capsule

The first step in securing data is creating a new Storage Capsule. The following tasks take place during the creation process:

1. The Capsule viewer solicits a Storage Capsule file name and size from the user.
2. The viewer makes a request to the Capsule server on the secure VM to create a new Storage Capsule.
3. The viewer asks the user to enter the secure key escape sequence that will be caught by a keyboard filter driver in the VMM. This deters spoofing by a compromised primary VM.
4. After receiving the escape sequence, the VMM module will give the secure VM control of the user interface.
    a. If the escape sequence is received unexpectedly (i.e. when the VMM has not received a request to wait for an escape sequence from the Capsule server), then the VMM module will still give control of the UI to the secure VM, but the secure VM will display a warning message saying that the user is *not* at a secure password entry page.
5. The Capsule server will ask the user to select a password, choose a random starting tweak value for encryption, and then format the encapsulated partition.
6. The Capsule server asks the VMM module to switch UI focus back to the primary VM.

After the creation process is complete, the Capsule server will send the viewer a file ID that it can store locally to link to the Storage Capsule on the server.

Loading a Storage Capsule from an external location requires fewer steps than creating a new Storage Capsule. If the viewer opens a Storage Capsule file that has been created elsewhere, it imports the file by sending it to the Capsule server. In exchange, the Capsule server sends the viewer a file ID that it can use as a link to the newly imported Storage Capsule. After a Storage Capsule has been loaded, the link on the primary VM looks the same regardless of whether the Capsule was created locally or imported from an external location.

5.3 Opening a Storage Capsule in Secure Mode

At this point, one or more Storage Capsules reside on the Capsule server, and have links to them on the primary VM. When the user opens a link with the Capsule viewer, it will begin the transition to secure mode, which consists of the following steps:

1. The Capsule viewer sends the Capsule server a message saying that the user wants to open a Storage Capsule, which includes the file ID from the link in the primary VM.
2. The Capsule viewer asks the user to enter the escape sequence that will be caught by the VMM module.
3. The VMM module receives the escape sequence and switches the UI focus to the secure VM. This prevents malware on the primary VM from spoofing a transition and stealing the file password.
    a. If the escape sequence is received unexpectedly, the secure VM still receives UI focus, but displays a warning message stating the system is *not* in secure mode.
4. The VMM module begins saving a snapshot of the primary VM in the background. Execution continues, but memory and disk data is copied to the snapshot on write.
5. The VMM module disables network and other device output.
6. The Capsule server asks the user to enter the file password.
7. The VMM module returns UI focus to the primary VM.
8. The Capsule server tells the viewer that the transition is complete and begins servicing disk I/O requests to the Storage Capsule.
9. The Capsule viewer mounts a local partition that uses the Capsule server for back-end disk block storage.

The above process ensures that the primary VM gains access to the Storage Capsule contents only after its initial state has been saved and the VMM has blocked device output. The exact set of devices blocked during secure mode is discussed more in the section on covert channels.

Depending on the source of the Storage Capsule encryption key, step 6 could be eliminated entirely. If the key was obtained from a smart card or other device, then the primary VM would retain UI focus throughout the entire transition, except in the case of an unexpected

escape sequence from the user. In this case, the secure VM must always take over the screen and warn the user that he or she is not in secure mode.

5.4 Storage Capsule Access in Secure Mode

When the Capsule system is running in secure mode, all reads and writes to the Storage Capsule are sent to the Capsule server. The server will encrypt and decrypt the data for each request as it is received, without performing any caching itself. The Capsule server instead relies on the caches within the primary VM and its own operating system to minimize unnecessary encryption and disk I/O. The disk cache in the primary VM sits above the driver that sends requests through to the Capsule server. On the secure VM, disk read and write requests from the Capsule server go through the local file system cache before they are sent to the disk. Later, we measure Storage Capsule disk performance during secure mode and demonstrate that is comparable to current encryption and virtualization software.

During secure mode, the VMM stores all writes to the primary VM's virtual disk in a file. This file contains differences with respect to the disk state at the time of the last snapshot operation (during the transition to secure mode). In the Capsule system, this difference file is stored on a partition that resides in main memory, commonly referred to as a RAM disk. Our implementation uses the ImDisk Virtual Disk Driver [14]. Storing the files on a RAM disk prevents potentially sensitive data that the primary VM writes to its virtual disk from reaching the physical disk. Although malicious access to the physical disk is not the focus of this paper's threat model, allowing confidential data to touch the disk increases the overall risk of leakage and the cost of hardware destruction.

The main downside to storing virtual disk modifications in memory is that the system requires more memory in secure mode. However, the size of primary disk modifications in secure mode should be small under normal circumstances because they are all temporary. Under normal circumstances, the user has no reason to store big files in a location where they will soon be deleted. If the change buffer does fill up due to a denial-of-service by malware or non-standard usage, then writes to the main virtual disk will fail until the system reverts to normal mode. If there turns out to be a legitimate need for large amounts of temporary storage in secure mode, then the change buffer could be stored on the physical disk and the VMM could use an encrypted file system for added protection.

5.5 Reverting to Normal Mode

Transitioning the Capsule system from secure mode back to normal mode is the most security-sensitive operation. Care must be taken to prevent leaks from secure mode back to normal mode insofar as practical for the desired level of security. A full discussion of the channels through which information might flow during this transition and countermeasures can be found in the next section. The Capsule system begins reverting to normal mode when the user enters a key escape sequence. Here, the escape sequence is not to prevent spoofing, but instead to reduce the primary VM's ability to leak data through a timing channel. After the user hits the escape sequence, the following steps take place:

1. The VMM module notifies the Capsule server of the pending transition, which in turn notifies the Capsule viewer.
2. The Capsule server waits up to 30 seconds for the primary VM to flush disk writes to the Storage Capsule. In our experiments, flushing always took less than one second, but uncommon workloads could make it take longer. We chose 30 seconds because it is the default maximum write-back delay for linux.
3. The secure VM reboots in order to flush any state that was affected by the primary VM. (This blocks some covert channels that are discussed in the next section.)
4. The VMM module halts the primary VM, and then reverts its state to the snapshot taken before entering secure mode and resumes execution.
5. The VMM module re-enables network and other device output for the primary VM.

After the Capsule system has reverted to normal mode, all changes that were made in the primary VM during secure mode, except those to the Storage Capsule, are lost. Also, when the Capsule viewer resumes executing in normal mode, it queries the Capsule to see what mode it is in (if the connection fails due to the reboot, normal mode is assumed). This is a similar mechanism to the return value from a fork operation. Without it, the Capsule viewer cannot tell whether secure mode is just beginning or the system has just reverted to normal mode, because both modes start from the same state.

5.6 Exporting Storage Capsules

After modifying a storage capsule, the user will probably want to back it up or transfer it to another person or computer at some point. Storage Capsules support this use case by providing an export operation. The Capsule viewer may request an export from the Capsule server at any time during normal mode. When the Capsule server

exports an encrypted Storage Capsule back to the primary VM, it is essential that malicious software can glean no information from the difference between the Storage Capsule at export compared to its contents at import. This should be the case even if malware has full control of the primary VM during secure mode and can manipulate the Storage Capsule contents in a chosen-plaintext attack.

For the Storage Capsule encryption scheme to be secure, the difference between the exported cipher-text and the original imported cipher-text must appear completely random. If the primary VM can change specific parts of the exported Storage Capsule, then it could leak data from secure mode. To combat this attack, the Capsule server re-encrypts the entire Storage Capsule using a new random 128-bit starting tweak value before each export. There is a small chance of two exports colliding. For any two Storage Capsules, each of size 2 GB (2^{27} encryption blocks), the chance of random 128-bit tweak values partially colliding would be approximately 1 in $2 * 2^{27} / 2^{128}$ or 1 in 2^{100}. Because of the birthday paradox, however, there will be a reasonable chance of a collision between a pair of exports after only 2^{50} exports. This number decreases further with the size of Storage Capsules. Running that many exports would still take an extremely long time (36 million years running 1 export / second). We believe that such an attack is unlikely to be an issue in reality, but could be mitigated if future tweaked encryption schemes support 256-bit tweak values.

5.7 Key Escape Sequences

During all Capsule operations, the primary VM and the Capsule viewer are not trusted. Some steps in the Capsule system's operation involve the viewer asking the user to enter a key escape sequence. If the primary VM becomes compromised, however, it could just skip asking the user to enter escape sequences and display a spoofed UI that looks like what would show up if the user did hit the escape sequence. This attack would steal the file decryption password while the system is still in normal mode. The defense against this attack is that the user should be accustomed to entering the escape sequence and therefore hit it anyway or notice anomalous behavior.

It is unclear how susceptible real users would be to spoofing attack that omits asking for an escape sequence. The success of such an attack is likely to depend on user education. Formally evaluating the usability of escape sequences in the Capsule system is future work. Another design alternative that may help if spoofing attacks are found to be a problem is reserving a secure area on the display. This area would always tell the user whether the system is in secure mode or the secure VM has control of the UI.

6. Covert Channel Analysis

The Storage Capsule system is designed to prevent any direct flow of information from secure mode to normal mode. However, there are a number of covert channels through which information may be able to persist during the transition from secure to normal mode. This section tries to answer the following questions about covert channels in the Capsule system as best as possible:

- Where can the primary virtual machine store information that it can retrieve after reverting to normal mode?
- What defenses might fully or partially mitigate these covert information channels?

We do not claim to expose all covert channels here, but list many channels that we have uncovered during our research. Likewise, the proposed mitigation strategies are not necessarily optimal, but represent possible approaches for reducing the bandwidth of covert channels. Measuring the maximum bandwidth of each covert channel requires extensive analysis and is beyond the scope of this paper. There has been a great deal of research on measuring the bandwidth of covert channels [2, 16, 21, 24, 30, 33], which could be applied to calculate the severity of covert channels in the Capsule system in future work.

The covert channels discussed in this section can be divided into five categories:

1. Primary OS and Capsule – Specific to Storage Capsule design
2. External Devices – Includes floppy, CD-ROM, USB, SCSI, etc.
3. External Network – Changes during secure mode that affect responsiveness to external connections
4. VMM – Arising from virtual machine monitor implementation, includes memory mapping and virtual devices
5. Core Hardware – Includes CPU and disk drives.

The Capsule system prevents most covert channels in the first three categories. It can use the VMM to mediate the primary virtual machine's device access and completely erase the primary VM's state when reverting to normal mode. The Capsule system also works to prevent timing channels when switching between modes of operation, and does respond to external network traffic while in secure mode.

Storage Capsules do not necessarily protect against covert channels in the last two categories. There has been a lot of work on identifying, measuring, and mitigating covert channels in core hardware for traditional MLS systems [16, 17, 21, 30]. Similar methods for measuring and mitigating those core channels could be applied to Storage Capsules. Covert channels arising from virtualization technology have not received much attention. This research hopes to highlight some of the key mechanisms in a VMM that can facilitate covert communication. The remainder of this section explores covert channels in each of these categories, including mitigation strategies and their required trade-offs.

6.1 Primary OS and Storage Capsule Channels

In addition to covert channels that one might find in other systems, the design of Storage Capsules creates opportunities for covert communication. This section focuses on the interactions between the primary VM and the secure VM. The primary VM should not be able to affect the secure VM's state during secure mode in a way that it can detect during normal mode. Here we examine three attack vectors that a compromised primary VM might use to steal data, and corresponding counter-measures.

Transition Timing

When the VMM restores the primary virtual machine from a snapshot, the restored state contains the exact same information as prior to entering secure mode. Following the restoration, the primary VM can obtain additional information by fetching the system time, which tells it when the system returned to normal mode. If the primary VM can manipulate the transition timing from secure mode, then it can leak data.

The Capsule system prevents direct leaks through transition timing by requiring a secure key escape sequence to switch back to normal mode. However, the primary VM may still be able to affect transition timing depending on how long the Capsule server waits for the primary VM to flush disk writes to the Storage Capsule upon leaving secure mode. One option that would preclude manipulation of transition timing would be for the Capsule server to wait a fixed time-period for disk writes to complete. This would be the most secure approach. Unfortunately, this strategy would either impact usability or threaten correctness. If the server were to stall 30 seconds – the default write-back delay for Linux – then the transition time would be unnecessarily long in most cases. On the other hand, if it only waited a few seconds, then some delayed writes might fail and corrupt the Storage Capsule.

For our implementation of the Capsule system, we chose a transition timing policy that sacrifices a few bits of potential information leakage per transition, but is more usable. The Capsule server will wait for the primary VM to flush Storage Capsule writes for up to 30 seconds. If the primary VM finishes the disk flush in less than 30 seconds, the server will immediately proceed with restoration. This opens up the possibility of the primary VM stalling a precise length of time after flushing writes so that it can read back timing information after returning to normal mode. The maximum bandwidth of this timing channel is somewhat limited, however, because the snapshot restoration adds noise to the transition time. Depending on how much memory the primary VM has, the amount of time between exiting secure mode and resuming in normal mode can vary by several seconds. By notifying the server at a specific offset from the most recent half-minute, the primary VM will only be able to communicate a few bits of data. One possible counter-measure to this type of timing channel would be to deliberately skew the clock during secure mode so the primary VM cannot accurately judge system time.

Storage Capsule Contents

The ideal place for a compromised primary VM to store data would be in the Storage Capsule itself. It is essential that the Capsule server cut off this channel. First, Storage Capsules must have a fixed size. This imposes a small usability limitation, which we believe is reasonable given that other popular systems like TrueCrypt [31] fix the size of encrypted file containers. Enforcing the next constraint required to cut off storage channels is slightly more complicated. No matter what changes the primary VM makes to the Storage Capsule in secure mode, it must not be able to deduce what has been changed after the Capsule server exports the Storage Capsule. As discussed earlier, XTS-AES encryption with a different tweak value for each export satisfies this requirement. Whether the primary VM changes every single byte or does not touch anything, the resulting exported Storage Capsule will be random with respect to its original contents.

Social Engineering Attacks

If the primary virtual machine cannot find a way to leak data directly, then it can resort to influencing user behavior. The most straightforward example of a social engineering attack would be for the primary VM to deny service to the user by crashing at a specific time, and then measuring transition time back to normal mode. There is a pretty good chance that the user would respond to a crash by switching back to normal mode immediately, especially if the system is prone to crash-

ing under normal circumstances. In this case, the user may not even realize that an attack is taking place. Another attack that is higher-bandwidth, but perhaps more suspicious, would be for the primary VM to display a message in secure mode that asks the user to perform a task that leaks information. For example, a message could read "Automatic update failed, please open the update dialog and enter last scan time '4:52 PM' when internet connectivity is restored." Users who do not understand covert channels could easily fall victim to this attack. In general, social engineering is difficult to prevent. The Capsule system currently does not include any counter-measures to social engineering. In a real deployment, the best method of fighting covert channels would be to properly educate the users.

6.2 External Device Channels

Any device that is connected to a computer could potentially store information. Fortunately, most devices in a virtual machine are virtual devices, including the keyboard, mouse, network card, display, and disk. In a traditional system, two processes that have access to the keyboard could leak data through the caps-, num-, and scroll-lock state. The VMware VMM resets this device state when reverting to a snapshot, so a virtual machine cannot use it for leaking data. We did not test virtualization software other than VMware to see how it resets virtual device state.

Some optional devices may be available to virtual machines. These include floppy drives, CD-ROM drives, sound adapters, parallel ports, serial ports, SCSI devices, and USB devices. In general, there is no way of stopping a VM that is allowed to access these devices from leaking data. Even devices that appear to be read-only, such as a CD-ROM drive, may be able to store information. A VM could eject the drive or position the laser lens in a particular spot right before switching back to normal mode. While these channels would be easy to mitigate by adding noise, the problem worsens when considering a generic bus like USB. A USB device could store anything or be anything, including a disk drive. One could allow access to truly read-only devices, but each device would have to be examined on an individual basis to uncover covert channels. The Capsule system prevents these covert channels because the primary VM is not given access to external devices. If the primary VM needs access to external devices, then they would have to be disabled during secure mode.

6.3 External Network Channels

In addition to channels from the Primary VM in secure mode to normal mode, it is also important to consider channels between the Storage Capsule system and external machines during secure mode. If malware can utilize so many resources that it affects how responsive the VMM is to external queries (such as pings), then it can leak data to a colluding external computer.

The best way to mitigate external network channels is for the VMM to immediately drop all incoming packets with a firewall, not even responding with reset packets for failed connections. If the VMM does not require any connections during secure mode, which it did not for our implementation, then this is the easiest and most effective approach.

6.4 Virtual Machine Monitor Channels

In a virtualization system, everything is governed by the virtual machine monitor, including memory mapping, device I/O, networking, and snapshot saving/restoration. The VMM's behavior can potentially open up new covert channels that are not present in a standard operating system. These covert channels are implementation-dependent and may or may not be present in different VMMs. This section serves as a starting point for thinking about covert channels in virtual machine monitors.

Memory Paging

Virtual machines are presented with a virtual view of their physical memory. From a VM's perspective, it has access to a contiguous "physical" memory segment with a fixed size. When a VM references its memory, the VMM takes care of mapping that reference to a real physical page, which is commonly called a machine page. There are a few different ways that a VMM can implement this mapping. First, it could directly pin all of the virtual machine's physical pages to machine pages. If the VMM uses this strategy, and it keeps the page mapping constant during secure mode and after restoration, then there is no way for a virtual machine to affect physical memory layout. However, this fixed mapping strategy is not always the most efficient way to manage memory.

Prior research describes resource management strategies in which the VMM may over-commit memory to virtual machines and page some of the VM's "physical" memory out to disk [11, 32]. If the VMM employs this strategy, then a virtual machine can affect the VMM's page table by touching different pages within its address space. The residual effects of page table manipulation may be visible to a VM after a snapshot restoration, unless the VMM first pages in all of the VM's memory.

A snapshot restoration should page in all of a VM's memory in most cases. But, if it is a "background" restoration, then accessing a memory location that has not been loaded from the snapshot yet *and* is paged out to disk might incur two page faults instead of one, which can be measured and may leak information.

VMware ESX server employs a number of tricks that allow it to over-commit memory to virtual machines [32]. Each of the mechanisms described by Waldspurger allow efficient server consolidation, but also create an opportunity for covert channels. The first mechanism he describes is *ballooning*. Guest virtual machines contain balloon drivers that allocate extra memory and then tell the VMM it can page out the corresponding "physical" pages. With ballooning, a misbehaving guest OS can touch the balloon driver's pages and cause the VMM to load them in from disk. When the VM is subsequently reverted to a snapshot, the compromised OS can again touch all of the balloon driver pages to see whether they have already been faulted in prior to the snapshot restoration. The VMM could mitigate this ballooning channel by restoring the state of its page table to match the original set of ballooned pages when restoring a snapshot. Of course, ballooning may be used as a covert channel between simultaneously executing VMs in systems other than Capsule.

There has also been research on sharing memory pages with the same content in virtual machines [3, 32]. If the VMM maps multiple pages to one machine page and marks them as copy-on-write, then the virtual machine will encounter a page fault when writing to one of them. If a VM sets a large number of pages to have the same value during secure mode, then a much larger number of page faults will take place when restoring a snapshot. However, these page faults will only be noticeable to the VM if execution resumes in normal mode before all of the VM's memory pages are loaded from the snapshot file. In this case, the VM can measure the total restoration time or write to pages before they have loaded and test for page faults to recover information. If the VM resumes execution after its memory has been fully restored and pages have been re-scanned for duplication, then this covert channel will not work.

The Capsule system does not over-commit memory for virtual machines, so the memory saving techniques mentioned above do not take effect. Our implementation of the Capsule system does not employ any counter-measures to covert channels based on memory paging.

Virtual Networks

The Capsule system blocks external network access during secure mode, but it relies on a virtual network for communication between the secure VM and the primary VM. While the virtual network itself is stateless (to the best of our knowledge), anything connected to the network could potentially be a target for relaying information from secure mode to normal mode. The DHCP and NAT services in the VMM are of particular interest. A compromised virtual machine may send arbitrary packets to these services in an attempt to affect their state. For example, a VM might be able to claim several IP addresses with different spoofed MAC addresses. It could then send ARP requests to the DHCP service following snapshot restoration to retrieve the spoofed MAC addresses, which contain arbitrary data. The Capsule system restarts both the DHCP and NAT services when switching back to normal mode to avert this covert channel.

Any system that allows both a high-security and low-security VM to talk to a third trusted VM (the secure VM in Capsule) exposes itself another covert channel. Naturally, all bets are off if the primary VM can compromise the secure VM. Even assuming the secure VM is not vulnerable, the primary VM may still be able to convince it to relay data from secure mode to normal mode. Like the DHCP service on the host, the secure VM's network stack stores information. For example, the primary VM could send out TCP SYN packets with specific source port numbers that contain several bits of data right before reverting to normal mode. When the primary VM resumes execution, it could see the source ports in SYN/ACK packets from the secure VM.

It is unclear exactly how much data can be stashed in the network stack on an unsuspecting machine and how long that information will persist. The only way to guarantee that a machine will not inadvertently relay state over the network is to reboot it. This is the approach we take to flush the secure VM's network stack state when switching back to normal mode in Capsule.

Guest Additions

The VMware VMM supports additional software that can run inside of virtual machines to enhance the virtualization experience. The features of guest additions include drag-and-drop between VMs and a shared clipboard. These additional features would undermine the security of any virtual machine system with multiple confidentiality levels and are disabled in the Capsule system.

6.5 Core Hardware Channels

Core hardware channels allow covert communication via one of the required primary devices: CPU or disk. Memory is a core device, but memory mapping is handled by the VMM, and is discussed in the previous section. Core hardware channels might exist in any multi-level secure system and are not specific to Storage Capsules or virtual machines. One difference between prior research and this work is that prior research focuses on a threat model of two processes that are executing simultaneously on the same hardware. In the Capsule system, the concern is not with simultaneous processes, but with a low-security process (normal-mode VM) executing on the same hardware after a high-security process (secure-mode VM) has terminated. This constraint rules out some traditional covert channels that rely on resource contention, such as a CPU utilization channel.

CPU State

Restoring a virtual machine's state from a snapshot will overwrite all of the CPU register values. However, modern processors are complex and store information in a variety of persistent locations other than architecture registers. Many of these storage areas, such as branch prediction tables, are not well-documented or exposed directly to the operating system. The primary method for extracting this state is to execute instructions that take a variable number of clock cycles depending on the state and measure their execution time, or exploit speculative execution feedback. Prior research describes how one can use these methods to leak information through cache misses [24, 33].

There are a number of counter-measures to covert communication through CPU state on modern processors. In general, the more instructions that execute in between secure mode and normal mode, the less state will persist. Because the internal state of a microprocessor is not completely documented, it is unclear exactly how much code would need to run to eliminate all CPU state. One guaranteed method of wiping out all CPU state is to power off the processor. However, recent research on cold boot attacks [12] shows that it may take several minutes for memory to fully discharge. This strategy would lead to an unreasonably long delay when switching from secure mode to normal mode.

The ideal solution for eliminating covert CPU state channels in Capsule and other virtualization systems would be with hardware support. The latest CPUs already support hardware virtualization, which allows them to fully emulate instruction sets for virtual machines. An additional mechanism is needed when switching between virtual machines that not only restores register and memory mappings, but also restores all state that could affect VM execution. This operation would load all of the cache data (L1, L2, and instruction), the branch prediction table, and any other intermediate state. It would also be sufficient to provide an instruction that would erase all of this data.

Although the Capsule system does not take counter-measures to prevent CPU state covert channels, the VMM restores a significant portion of the primary VM's original memory state before the VM can resume execution. This restoration will likely eliminate any residual cache lines from secure mode. It will also add a great deal of noise to any other CPU state, such as branch predictor tables, due to the number of instructions that execute during restoration. Computing the maximum bandwidth of CPU state channels in the Capsule system is future work.

Disk State

The caching and layout of files on disk, and in the file system, can be used to transmit information. Another covert channel exists in the positioning of the disk arm [10]. If the primary VM can manipulate disk state in a way that is readable after transitioning back to normal mode, then it can leak data. There are a few aspects of the Capsule system's design that make such an attack much more difficult. Any writes to the virtual disk during secure mode actually go to flat difference file that stores changes since the last snapshot operation. The primary VM has no control over where the data goes on disk, only how much is written, which would make exploitation of such a channel difficult. The VMM deletes this file when transitioning back to normal mode. This frees up any blocks that were allocated for the difference file. Finally, the VMM reads in the primary VM's original memory state from disk during snapshot restoration. This moves the position of the disk arm and adds considerable noise to the cache.

Despite the noise from transitioning back to normal mode, the disk cache is the most likely candidate for a covert channel. All writes are sent to a difference file and deleted, but reads touch the same virtual disk file during secure mode and normal mode. Consider the primary VM sequentially reading every block on its disk during secure mode to transmit a 1, or reading very little to transmit a 0. The primary VM might be able to tell what happened in secure mode by reading back disk blocks and measuring if they hit the disk cache. One would need to flush all of the disk caches to eliminate this channel. The Capsule system does not take any steps to mitigate disk state channels.

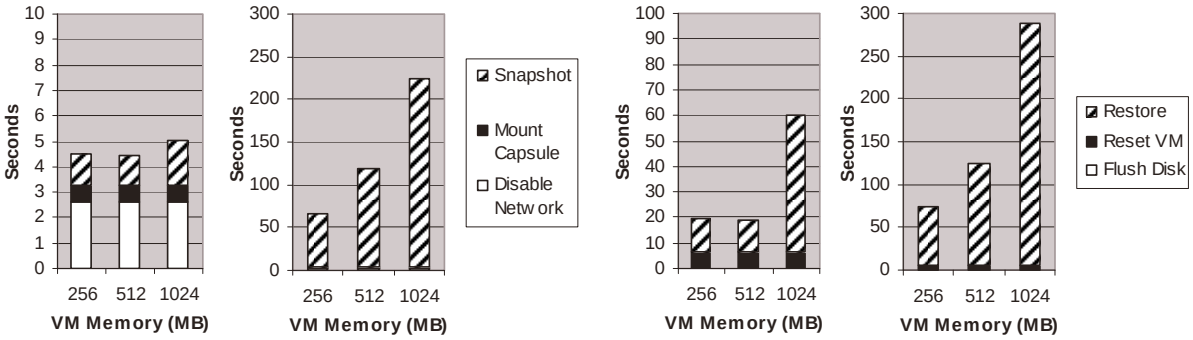

Figure 2. Transition times for different amounts of primary VM memory.
(a) to secure mode with background snapshot, (b) to secure mode with full snapshot
(c) to normal mode with background restore, and (d) to normal mode with full restore.

6.6 Mitigating VMM and Core Hardware Covert Channels

The design of Storage Capsules centers around improving local file encryption with a minimal impact on existing behavior. The user only has to take a few additional steps, and no new hardware is required. The current implementation is designed to guard against many covert channels, but does not stop leakage through all of them, such as the CPU state, through which data may leak from secure to normal mode. If the cost of small leaks outweighs usability and the cost of extra hardware, then there is an alternative design that can provide additional security.

One way of cutting off almost all covert channels would be to migrate the primary VM to a new isolated computer upon entering secure mode. This way, the virtual machine would be running on different core hardware and a different VMM while in secure mode, thus cutting off covert channels at those layers. VMware ESX server already supports live migration, whereby a virtual machine can switch from one physical computer to another without stopping execution. The user would have two computers at his or her desk, and use one for running the primary VM in secure mode, and the other for normal mode. When the user is done accessing a Storage Capsule, the secure mode computer would reboot and then make the Storage Capsule available for export over the network. This extension of the Capsule system's design would drastically reduce the overall threat of covert channels, but would requires additional hardware and could add usability impediments that would not be suitable in many environments.

7. Performance Evaluation

There are three aspects of performance that are important for Storage Capsules: (1) transition time to secure mode, (2) system performance in secure mode, and (3) transition time to normal mode. It is important for transitions to impose only minimal wait time on the user and for performance during secure mode to be comparable to that of a standard computer for common tasks. This section evaluates Storage Capsule performance for transitions and during secure mode. The experiments were conducted on a personal laptop with a 2 Ghz Intel T2500 processor, 2 GB of RAM, and a 5200 RPM hard drive. Both the host and guest operating systems (for the secure VM and primary VM) were Windows XP Service Pack 3, and the VMM software was VMware Workstation ACE Edition 6.0.4. The secure VM and the primary VM were both configured with 512 MB of RAM and to utilize two processors, except where indicated otherwise.

The actual size of the Storage Capsule does not affect any of the performance numbers in this section. It does, however, influence how long it takes to run an import or export. Both import and export operations are expected to be relatively rare in most cases – import only occurs when loading a Storage Capsule from an external location, and export is required only when sending a Storage Capsule to another user or machine. Importing and exporting consist of a disk read, encryption (for export only), a local network transfer, and a disk write. On our test system, the primary VM could import a 256 MB Storage Capsule in approximately 45 seconds and export it in approximately 65 seconds. Storage Capsules that are imported and exported more often, such as e-mail attachments, are likely to be much smaller and should take only a few seconds.

7.1 Transitioning to and from Secure Mode

The transitions to and from secure mode consist of several tasks. These include disabling/enabling device output, mounting/dismounting the Storage Capsule, saving/restoring snapshots, waiting for an escape sequence, and obtaining the encryption key. Fortunately, some operations can happen in parallel. During the transition to secure mode, the system can do other things while waiting for user input. The evaluation does not count this time, but it will reduce the delay experienced by the user in a real deployment. VMware also supports both background snapshots (copy-on-write) and background restores (copy-on-read). This means that execution may resume in the primary VM before memory has been fully saved or restored from the snapshot file. The system will run slightly slower at first due to page faults, but will speed up as the snapshot or restore operation nears completion. A background snapshot or restore must complete before another snapshot or restore operation can begin. This means that even if the primary VM is immediately usable in secure mode, the system cannot revert to normal mode until the snapshot is finished.

Figure 2 shows the amount of time required for transitioning to and from secure mode with different amounts of RAM in the primary VM. Background snapshots and restorations make a huge difference. Transitioning to secure mode takes 4 to 5 seconds with a background snapshot, and 60 to 230 seconds without. The time required for background snapshots, mounting the Storage Capsule, and disabling network output also stays fairly constant with respect to primary VM memory size. However, the full snapshot time scales linearly with the amount of memory. Note that the user must wait for the full snapshot time before reverting to normal mode.

The experiments show that reverting to normal mode is a more costly operation than switching to secure mode, especially when comparing the background restore to the background snapshot operation. This is because VMware allows a virtual machine to resume immediately during a background snapshot, but waits until a certain percentage of memory has been loaded in a background restore. Presumably, memory reads are more common than memory writes, so copy-on-read for the restore has worse performance than copy-on-write for the snapshot. VMware also appears to employ a non-linear strategy for deciding what portion of a background restore must complete before the VM may resume execution. It waited approximately the same amount of time when a VM had 256 MB or 512 MB of RAM, but delayed significantly longer for the 1 GB case.

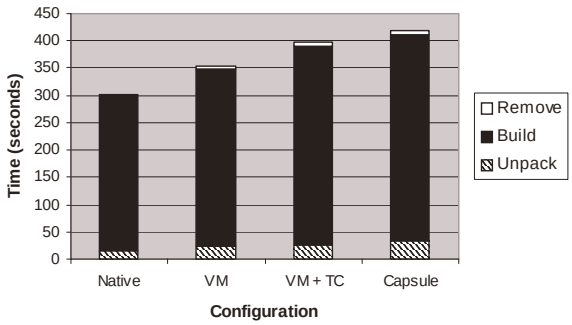

Figure 3. Results from building Apache with a native OS, a virtual machine, a virtual machine running TrueCrypt, and Capsule. Storage Capsules add only a 5% overhead compared to a VM with TrueCrypt, 18% slower than a plain VM, and 38% overhead compared to a native OS.

The total transition times to secure mode are all reasonable. Many applications will take 4 or 5 seconds to load a document anyway, so this wait time imposes little burden on the user. The transition times back to normal mode for 256 MB and 512 MB are also reasonable. Waiting less than 20 seconds does not significantly disrupt the flow of work. However, 60 seconds may be long wait time for some users. It may be possible to optimize snapshot restoration by using copy-on-write memory while the primary VM is in secure mode. This way, the original memory would stay in tact and the VMM would only need to discard changes when transitioning to normal mode. Optimizing transition times in this manner is future work.

7.2 Performance in Secure Mode

Accessing a Storage Capsule imposes some overhead compared to a normal disk. A Storage Capsule read or write request traverses the file system in the primary VM, and is then sent to the secure VM over the virtual network. The request then travels through a layer of encryption on the secure VM, out to its virtual disk, and then to the physical drive. We compared the disk and processing performance of Storage Capsules to three other configurations. These configurations consisted of a native operating system, a virtual machine, and a virtual machine with a TrueCrypt encrypted file container. For the evaluation, we ran an Apache build benchmark. This benchmark involves decompressing and extracting the Apache web server source code, building the code, and then removing all of the files. The Apache build benchmark probably represents the worst case scenario for Storage Capsule usage. We expect that the primary use of Storage Capsules will be for less disk-intensive activities like editing documents or images, for which the overhead should be unnoticeable.

Figure 3 shows the results of the Apache build benchmark. Storage Capsules performed well overall, only running 38% slower than a native system. Compared to a single virtual machine running similar encryption software (TrueCrypt), Storage Capsules add an overhead of only 5.1% in the overall benchmark and 31% in the unpack phase. This shows that transferring reads and writes over the virtual network to another VM has a reasonably small performance penalty. The most significant difference can be seen in the remove phase of the benchmark. It executes in 1.9 seconds on a native system, while taking 5.5 seconds on a VM, 6.5 seconds on a VM with TrueCrypt, and 7.1 seconds with Storage Capsules. The results from the VM and VM with TrueCrypt tests show, however, that the slowdown during the remove phase is due primarily to disk performance limitations in virtual machines rather than the Capsule system itself.

8. Conclusion and Future Work

This paper introduced Storage Capsules, a new mechanism for securing files on a personal computer. Storage Capsules are similar to existing encrypted file containers, but protect sensitive data from malicious software during decryption and editing. The Capsule system provides this protection by isolating the user's primary operating system in a virtual machine. The Capsule system turns off the primary OS's device output while it is accessing confidential files, and reverts its state to a snapshot taken prior to editing when it is finished. One major benefit of Storage Capsules is that they work with current applications running on commodity operating systems.

Covert channels are a serious concern for Storage Capsules. This research explores covert channels at the hardware layer, at the VMM layer, in external devices, and in the Capsule system itself. It looks at both new and previously examined covert channels from a novel perspective, because Storage Capsules have different properties than side-by-side processes in a traditional multi-level secure system. The research also suggests ways of mitigating covert channels and highlights their usability and performance trade-offs. Finally, we evaluated the overhead of Storage Capsules compared to both a native system and standard virtual machines. We found that transitions to and from secure mode were reasonably fast, taking 5 seconds and 20 seconds, respectively. Storage Capsules also performed well in an Apache build benchmark, adding 38% overhead compared to a native OS, but only a 5% penalty when compared to running current encryption software inside of a virtual machine.

In the future, we plan to further explore covert channels discussed in this work. This includes measuring their severity and quantifying the effectiveness of mitigation strategies. We also hope to conduct a study on usability of keyboard escape sequences for security applications. Storage Capsules rely on escape sequences to prevent spoofing attacks by malicious software, and it would be beneficial to know how many users of the Capsule system would still be vulnerable to such attacks.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0705672. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation

References

[1] M. Blaze. A Cryptographic File System for UNIX. In *Proc. of the 1st ACM Conference on Computer and Communications Security*, Nov. 1993.

[2] R. Browne. An Entropy Conservation Law for Testing the Completeness of Covert Channel Analysis. In *Proc. of the 2nd ACM Conference on Computer and Communication Security (CCS)*, Nov. 1994.

[3] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. *ACM Transactions on Computer Systems*, 15(4), Nov. 1997.

[4] A. Czeskis, D. St. Hilair, K. Koscher, S. Gribble, and T. Kohno. Defeating Encrypted and Deniable File Systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications. In *Proc. of the 3rd USENIX Workshop on Hot Topics in Security (HOTSEC '08)*, Aug. 2008.

[5] M. Delio. Linux: Fewer Bugs than Rivals. *Wired Magazine*, `http://www.wired.com/ software/coolapps/news/2004/12/66022`, Dec. 2004.

[6] D. Denning and P. Denning. Certification of Programs for Secure Information Flow. *Communications of the ACM*, 20(7), Jul. 1977.

[7] C. Fruhwirth. LUKS – Linux Unified Key Setup. `http://code.google.com/p/ cryptsetup/`, Jan. 2009.

[8] T. Garfinkel, B. Pfaff, J, Chow, M. Rosenblum, D. Boneh. Terra: a Virtual Machine-based Platform for Trusted Computing. In *Proc. of the 19th*

ACM Symposium on Operating Systems Principles (SOSP), Oct. 2003.

[9] T. Garfinkel and M. Rosenblum. When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. In *Proc. of the 10th Workshop on Hot Topics in Operating Systems*, Jun. 2005.

[10] B. Gold, R. Linde, R. Peeler, M. Schaefer, J. Scheid, and P. Ward. A Security Retrofit of VM/370. In *AFIPS Proc., 1979 National Computer Conference*, 1979.

[11] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular Disco: Resource Management Using Virtual Clusters on Shared-Memory Multiprocessors. In *Proc. of the Symposium on Operating System Principles*, Dec. 1999.

[12] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proc. of 17th USENIX Security Symposium*, Jul. 2008.

[13] IEEE Computer Society. IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. *IEEE Std 1619-2007*, Apr. 2008.

[14] O. Lagerkvist. ImDisk Virtual Disk Driver. `http://www.ltr-data.se/opencode.html#ImDisk`, Dec. 2008.

[15] T. Jaeger, R. Sailer, and X. Zhang. Analyzing Integrity Protection in the SELinux Example Policy. In *Proc. of the 12th USENIX Security Symposium*, Aug. 2003.

[16] M. Kang and I. Moskowitz. A Pump for Rapid, Reliable, Secure Communication. In *Proc. of the 1st ACM Conference on Computer and Communication Security (CCS)*, Nov. 1993.

[17] R. Kemmerer. An Approach to Identifying Storage and Timing Channels. In *ACM Transactions on Computer Systems*, 1(3), Aug. 1983.

[18] M. Liskov, R. Rivest, and D. Wagner. Tweakable Block Ciphers. In *Advances in Cryptology – CRYPTO '02*, 2002.

[19] R. Meushaw and D. Simard. NetTop: Commercial Technology in High Assurance Applications. `http://www.vmware.com/pdf/Tech-TrendNotes.pdf`, 2000.

[20] Microsoft Corporation. BitLocker Drive Encryption: Technical Overview. `http://technet.microsoft.com/en-us/library/cc732774.aspx`, Jan. 2009.

[21] I. Moskowitz and A. Miller. Simple Timing Channels. In *Proc. of the IEEE Symposium on Security and Privacy*, May 1994.

[22] A. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Proc. of the 26th ACM Symposium on Principles of Programming Languages (POPL)*, Jan. 1999.

[23] National Security Agency. Security-enhanced Linux. `http://www.nsa.gov/selinux`, Jan. 2008.

[24] C. Percival. Cache Missing for Fun and Profit. In *Proc. of BSDCan 2005*, May 2005.

[25] A. Roshal. WinRAR Archiver, a Powerful Tool to Process RAR and ZIP Files. `http://www.rarlab.com/`, Jan. 2009.

[26] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn. Design and Implementation of a TCG-Based Integrity Measurement Architecture. In *Proc. of the 13th USENIX Security Symposium*, Aug. 2004.

[27] Secunia. Xen 3.x – Vulnerability Report. `http://secunia.com/product/15863/?task=statistics`, Jan. 2009.

[28] Secunia. Linux Kernel 2.6.x – Vulnerability Report. `http://secunia.com/product/2719/?task=statistics`, Jan. 2009.

[29] Trusted Computing Group. Trusted Platform Module Main Specification. `http://www.trustedcomputinggroup.org`, Ver. 1.2, Rev. 94, June 2006.

[30] J. Trostle. Multiple Trojan Horse Systems and Covert Channel Analysis. In *Proc. of Computer Security Foundations Workshop IV*, Jun. 1991.

[31] TrueCrypt Foundation. TrueCrypt – Free Open-Source On-the-fly Encryption. `www.truecrypt.org`, Jan. 2009.

[32] C. Waldspurger. Memory Resource Management in VMware ESX Server. In *Proc. of the 5th Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[33] Z. Wang and R. Lee. Covert and Side Channels Due to Processor Architecture. In *Proc. of the 22nd Annual Computer Security Applications Conference (ACSAC)*, Dec. 2006.

[34] T. Weber. Criminals 'May Overwhelm the Web'. *BBC News*, `http://news.bbc.co.uk/1/hi/business/6298641.stm`, Jan. 2007.

[35] WinZip International LLC. WinZip – The Zip File Utility for Windows. `http://www.winzip.com/`, Jan. 2009.

[36] XenSource, Inc. Xen Community. `http://xen.xensource.com/`, Apr. 2008.