Securing Sensitive Content in a View-Only File System

Kevin Borders, Xin Zhao, Atul Prakash
University of Michigan
Department of Electrical Engineering and Computer Science
2260 Hayward Street, Ann Arbor, MI
{kborders, zhaoxin, aprakash}@eecs.umich.edu

ABSTRACT
One of the most fundamental problems in computer security is protecting sensitive digital information from unauthorized disclosure. There are a number of challenges, such as spyware, removable media, and mobile devices, which make this a very hard problem. The problem becomes even more difficult when the adversary is somebody who is authorized to view the data. This is what is commonly referred to as an insider information leak. Insider leaks often occur out of malice, but sometimes are just due to plain negligence, as was the case with a recent leak of 26 million U.S. veterans' names, birth dates, and social security numbers. Current systems make an attempt to protect against this type of disclosure, but use rudimentary techniques that can be easily bypassed by a knowledgeable attacker. Examples include disabling "print" and "save" menu options within an application or scanning network traffic for signatures of known sensitive content. This paper examines a new method for protecting sensitive content from unauthorized disclosure, a View-Only File System (VOFS). VOFS relies on trusted computing primitives and virtual machine (VM) technology to provide a much greater level of security than current systems. In VOFS, a secure virtual machine on the client authenticates itself with a content provider and downloads sensitive data. Before allowing the user to view the data in his or her non-secure VM, the VOFS client disables non-essential device output. This prevents the user, or any malicious software, from printing, uploading, or stealing the sensitive content. When the user is done viewing a sensitive file, VOFS will reset the machine to previous state and resume normal device activity. Our goal is to provide near-seamless access to view-only files, while at the same time securing them from unauthorized digital replication. This paper presents the initial design, development plan, and evaluation plan for VOFS.

Categories and Subject Descriptors
D.4.6 [**Security and Protection**]: Information Flow Controls – *sensitive information.*

General Terms
Design, Security.

DRM'06, October 30, 2006, Alexandria, Virginia, USA.

Keywords
Digital rights management, information leakage, insider abuse, virtual machines, file systems.

1. INTRODUCTION
Protecting sensitive digital information is a major concern for government agencies, corporations, and even home users. Many security mechanisms exist today in order to prevent unauthorized disclosure of information, including encryption, firewalls, and authentication [9, 13, 16]. Unfortunately, most of these mechanisms fail once sensitive content is decrypted for viewing on the end host. Furthermore, these systems are unable to deal with the most dangerous threat to security: an insider information leak. Recently, an employee of the Veterans Administration took home a laptop computer that contained the names, birth dates, and social security numbers of 26 million U.S. veterans and was later stolen. This led to a congressional investigation [4]. The heart of the problem was that the user was authorized to view the sensitive data, but not authorized to copy it to his hard drive and take it home with him. This is a very common situation where someone can view content, but should not print, send, or otherwise replicate the sensitive content. This type of data will be referred to as *sensitive finished content*. The View-Only File System aims to prevent unauthorized disclosure of sensitive finished content, regardless of whether user is authorized to view the content.

The design of VOFS is based on a set of ideal security policies for protecting sensitive finished content. In a perfect world, authorized users should be able to view sensitive information, but not be allowed to disclose the information to unauthorized parties, regardless of whether the disclosure is intentional. Furthermore, all sensitive finished content should be safe from remote access. In addition to these security policies, we impose a usability constraint: people should be able to use their standard applications on their normal workstation for displaying, searching, playing, or otherwise transforming the sensitive digital data into a viewable or audible format. Although these policies cannot be enforced in their entirety (we cannot prevent users from manually copying information or remembering and disclosing it at a later time), the following design goals aim to meet the policies insofar as possible:

1. Sensitive finished content will only be provided to authenticated and authorized users of physical machines running trusted software and hardware.
2. Once a client reads sensitive content, all device output for that machine, except for video, sound, and limited disk output, should be disabled by a trusted system component.

3. When the user is done viewing the sensitive content, he or she should be able to revert to a system state before the sensitive information was read, at which point device output may resume.

Assuming the integrity of trusted components, satisfying these goals will prevent the user's machine from being able to leak sensitive information, regardless of the user's actions or any malicious software installed on the system.

VOFS utilizes virtual machine (VM) technology to achieve isolation between trusted and un-trusted system components and fulfill its design goals. VOFS will use the Xen virtual machine monitor [5]. The Xen virtual machine monitor (VMM) runs as the lowest software layer in the system. The VMM has complete control over the computer and mediates interaction between virtual machines and the hardware. Above the VMM are guest virtual machines. The user's primary operating system will run inside of a guest VM, and will be referred to as the *primary guest VM*. This machine is not trusted and will contain all of the user's standard applications. Another guest virtual machine, referred to as *Domain 0*, has complete administrative access to the system, and is considered to be trusted. In VOFS, Domain 0 will be responsible for enabling and disabling device output, and saving and restoring the state of other virtual machines. A third guest VM, the *SVFS VM*, will be responsible for downloading sensitive finished content, storing it, telling Domain 0 when to save or restore the primary guest VM's state, and telling Domain 0 when to enable or disable device output. The SVFS VM is based on the data virtual machine from work on a Secure Virtual File System [21].

VOFS clients will also take advantage of trusted platform module (TPM) technology [18]. The clients will use an integrity measurement architecture similar to those found in [14] and [10], which use trusted boot, to enable remote verification of trusted components by content providers. In the trusted boot process, the TPM will hold cryptographic digests of trusted system components where they are safe from tampering. First, a small region of the BIOS will compute a cryptographic hash of the remainder of the BIOS and send it to the TPM. Next, the BIOS will measure the boot sector. The boot sector will in turn measure the lowest software layer, the virtual machine monitor (VMM), which will be responsible for verifying the integrity the trusted Domain 0 and SVFS virtual machines. During the process, system components do not check the next component's the hash value before allowing it to run, but they do store the hash value in a secure location on the TPM. This is different than secure boot, where each level of the system will not allow the next to run if its digest does not match a predetermined value [3]. This extra restriction in secure boot is not necessary for VOFS because the machine will not be allowed to receive sensitive content unless the hashes of its trusted components match known correct values.

When the user wishes to view a view-only file in VOFS, his or her primary VM will make a request to the SVFS VM, which will contact the content provider to start an authorization session. This session has built-in mechanisms to prevent message reordering, deletion, and replay [18]. Next, the content provider will send a request to verify that the integrity measurements stored in the TPM match trusted values.. The TPM will sign the integrity measurements using an asymmetric private key only known to the TPM. If the server accepts the integrity values, it will send the encrypted sensitive content to the client. The client will only be able to decrypt the content using a key on the trusted platform module. This all occurs in one authorization session to prevent the machine from restarting with malicious software in between messages. Next, when the SVFS VM has obtained the decryption key for the sensitive finished content, it will instruct the virtual machine monitor to take a snapshot of the guest VM's state and disable its device output. It will finally decrypt the sensitive content and allow the user's virtual machine to display it. When the user is done viewing the sensitive content the VMM will discard it, restore the old state of the primary guest VM, and re-enable device I/O.

Although VOFS provides a much greater level of security than current systems, it still relies on the integrity of trusted components and cannot protect against all attacks. The threat model for our system assumes that the following characteristics about trusted system components:

- The attacker *cannot* probe memory or any bus on the system's motherboard.
- The attacker *cannot* flash the part of the BIOS that makes the initial TPM measurement.
- The attacker *cannot* steal the TPM chip.
- The attacker *cannot* modify trusted software components at run time (The VMM, Domain 0, and SVFS VM) by tampering with or replacing the CPU, memory, or disk.
- If an attacker steals an entire machine, an administrator will notice and revoke the machine's credentials before the attacker can modify the machine and obtain sensitive content from the content server.
- The attacker *can* install arbitrary software on the primary guest VM while it is running.
- The attacker *can* replace the operating system on the VOFS client while it is turned off.
- The attacker *can* remove the hard drive or any other removable media and access it offline.
- The attacker *can* leak sensitive information through any device output channel, including the serial port, parallel port, universal serial bus (USB), and network interface.

We believe the assumptions about the attacker's ability to probe hardware, modify the BIOS, and steal machines to be reasonable in an organizational environment. They may not be reasonable, however, for machines completely under the user's control. VOFS does not provide the same level of security and is not recommended for end consumer digital rights management. VOFS does rely on the integrity of the VMM, Domain 0, and SVFS VM at runtime. These components are much more secure than a traditional operating system because they run very few services (no network services), and only perform basic functions. Although it may be possible to compromise a trusted component, doing so would be much more difficult than compromising the user's primary guest VM. Our threat model also does *not* encompass all covert communication channels. These channels are generally considered to be low-bandwidth and would most likely be slower than photographing the screen or copying down information by hand, which are beyond the scope of our threat model.

Today, most workstation machines do not use virtual machine technology. Deploying a virtual-machine based security system in an enterprise network could be time-consuming from an

administrative standpoint, and may reduce system performance. However, recent research on creating a virtual-machine based root kit [11] shows that it is possible to install a virtual machine monitor and hoist the machine's current operating system into a guest VM without any interaction from the user (and even without the user noticing). VOFS could be installed in a similar manner to avoid administrative overhead. Furthermore, a common user will probably not be able to tell the difference between a virtual machine and a standard operating system because their levels of performance are very close on modern virtual machine monitors such as Xen [5], and VMWare [19].

The remainder of this proposal is laid out a follows: Section 2 discusses related work. Section 3 outlines the design of VOFS. Section 4 describes the details of the VOFS Client Architecture. Section 5 briefly talks about preliminary and future work. Section 6 presents our evaluation plan and expected performance issues. Finally, section 7 concludes.

2. RELATED WORK

Some work has been done on methods for local verification and remote attestation of software integrity [3, 14, 10]. These methods ensure that trusted system components are correct when they start running. The Terra platform [10] goes even further by having a trusted virtual machine monitor that can verify the integrity of individual virtual machines. VOFS operates in a very similar fashion; it computes the cryptographic hash of the trusted Domain 0 and SVFS virtual machines during startup. Although trusted system components are a critical part of VOFS, its focus is on building an application on top of trusted primitives.

VOFS's information flow policies share a lot of similarities with mandatory access control [7, 15]. In the initial state of the system, you can think of the user's primary guest VM, all of its applications, and all external devices as being low-security objects. The sensitive content is a high-security object. Now, a low security object cannot have access to a high-security object unless it is upgraded to be a high-security object itself, and has all information flows to low-security objects cut off. This is essentially what happens when the user's machine views protected content; it is upgraded to be a high-security object and has its device output (flow to low-security objects) disabled. VOFS, unlike traditional mandatory access control systems such as SELinux [12], allows for downgrading of the user's machine to a low-security object again by reverting to a previous state. Without this capability, users would be unable to use the same applications for accessing sensitive and non-sensitive information, which would severely impact usability.

The primary benefit, however, of VOFS over a traditional SELinux deployment is that it is more secure. Although SELinux does a great job of preventing an unauthorized user or process from accessing sensitive content, it does not do a good job of preventing an authorized process from leaking the content. If an application knows it is going to be reading a sensitive file and wants to send it to a low-security process, it can open up a pipe or shared memory region beforehand, open the file, then send it out over the communication channel to the unauthorized process. This will not be possible with VOFS because *all* device output from the system will be disabled. Processes can steal and leak information between each other as much as they want, but there is no avenue for leaking the sensitive data to an external entity.

Some applications also try to prevent unauthorized replication of digital content, though they have no support from the operating system. One such application is Adobe Acrobat, the standard viewing application for the popular portable document format (PDF) [1]. Acrobat has a feature where the publisher of a document can specify that it is protected, which causes the viewer to disable certain menu options, such as "print", when displaying the file. This protection mechanism provides no real security. Non-standard PDF viewers can open the same document and print it freely. The user can also copy the file from its original location onto any disk or send it across the network to an unauthorized party. VOFS, on the other hand, protects sensitive documents by only allowing the system to decrypt and open them after all device output has been disabled. So, the user's application may have a "print" menu option, but selecting it will yield an error message that a printer is not connected to the system.

Another application that attempts to protect sensitive digital information is Apple's iTunes [2], which uses FairPlay digital rights management technology [8]. Again, iTunes has no support from the operating system and has only has limited success at preventing unauthorized replication. Although iTunes uses what appear to be cryptographically sound methods for encrypting sensitive content, it is unable to store the key used to decrypt these files in a secure location. Applications have already been developed [20] to decrypt protected files from iTunes. Even if iTunes were able to securely store its key, however, it would run into the problem of an attacker who controls the operating system being able to read the decrypted from its memory. Digital rights management provided by iTunes is analogous to placing a steel padlock on a cardboard box. VOFS, however, provides a complete solution by fully isolating the virtual machine that is viewing the sensitive content.

Cryptographic file systems attempt to address some of the same threats as VOFS. In particular, they are concerned with unauthorized access to sensitive files on disk [6]. One shortcoming of cryptographic file systems is their overhead, which can be quite high [6]. In contrast, VOFS will only need to perform cryptographic operations when viewing sensitive files, and will not affect performance during normal operation. Cryptographic file systems also assume that authorized users are trusted and it does nothing to prevent them from leaking sensitive files.

There are some network-based solutions that try to prevent authorized users from sending data to an external network [17]. These solutions are only somewhat successful, however, because they usually cannot identify obfuscated or encrypted information. Another way of preventing information leakage over a network is to physically disconnect the network from the outside world. This is known as an air gap, and is the preferred method of security for some organizations. The problem with an air gap, however, is that some users on the internal network may still be unauthorized to view certain sensitive data. Furthermore, restricting network connectivity does nothing to stop a malicious insider from printing sensitive files or copying them to removable media and taking them home. A more complete solution would be a computer with no I/O devices other than a keyboard, mouse, video card, sound card, and a read-only drive such as a CD-ROM player. This extreme solution, however, is almost completely unusable, and would still not provide as good of security as VOFS because content providers are unable to revoke access if a key is compromised or a machine is stolen.

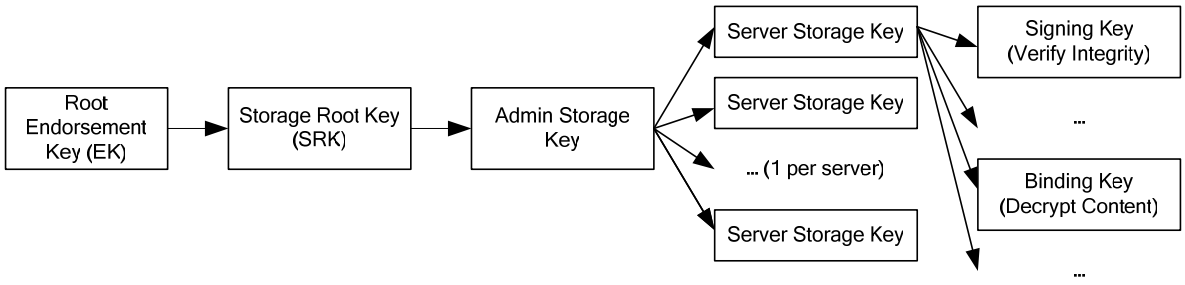

Figure 1. TPM trust hierarchy on each VOFS machine. Arrows indicate that they key on the left was used to create they key on the right

3. VOFS DESIGN

The design of VOFS is broken up into two sections. In this section, we will give an overview of trusted platform module technology and its use in VOFS. Next, we will present the content transfer protocol that enables content providers to securely send protected data to authorized users on machines running trusted hardware and software. In the following section, we will talk about the SVFS client machine architecture.

3.1 Trusted Platform Module Overview

In order to give some background into the basis of the content transfer protocol, we will give a brief overview of trusted platform module (TPM) technology and how to use it to verify the hardware and software running on a remote machine. A Trusted Platform Module is a chip that goes into the motherboard of a computer, and has a number of capabilities. The primary use of a trusted platform module is protecting secret asymmetric cryptographic keys and using those keys to sign digests of firmware and software components. When a TPM-enabled machine boots up, the first thing that happens after the processor's Power on Self-Test is a small trusted portion of the system BIOS calculates a cryptographic hash of the rest of the BIOS, and sends the hash to the TPM. The main part of the BIOS then begins running and calculates a cryptographic hash of the initial boot block and sends it to the TPM. The BIOS then executes the boot block, which will send a hash of the operating system (In the case of VOFS, the virtual machine monitor) to the TPM.

One key property of trusted platform modules is that they are based on a request-response architecture. This means that a small portion of the BIOS needs to be trusted to send a correct measurement of the rest of the BIOS to the TPM. Furthermore, the CPU, main memory, and all of the communication channels between the TPM, BIOS, CPU, and memory need to be trusted. Otherwise, an adversary could send an incorrect hash value to the trusted platform module, allowing malicious modification of software to go unnoticed.

Trusted platform modules use a chain of trust to assert the integrity of each key. The chain of trusted keys on a VOFS machine can be seen in Figure 1. The basis of trust is a root endorsement key (EK). Once the EK is set inside of a TPM, it cannot be changed. The TPM uses the EK to authenticate itself when generating certain child keys, including the storage root key (SRK). The "owner" of the TPM, which in this case is a system administrator, can use the storage root key to create other keys on the system. Whenever an entity "uses" a key to perform some operation, that entity must provide an encrypted shared secret to the TPM, such as a password. Even if the public portion of the storage root key is known, only the owner is able to use the storage root key to create other keys, because only he or she knows the shared secret, which is also stored securely on the TPM.

In VOFS, the administrator, who is also the TPM owner on all client machines, will create an "admin" storage key, which is a direct child of the storage root, with the same shared secret on all clients. (The "password" to use the admin key on each machine will be the same, even though the actual keys will be different). The admin storage key is the third key from the left in Figure 1. The administrator will then use the admin key to create one server storage key per content provider on all of the client machines. He or she will then set the shared secret for each server to be same on all clients. Again, the actual server keys will be different, which allows the server to uniquely identify them, but the provider will be able to access each client using the same shared secret. Each server should maintain an access control list that specifies the rights of each client, allowing restriction of file permissions based on the client machine. This is especially important if a client machine is stolen; the administrator needs to be able to revoke access rights on a per-client basis. Optionally, if one is worried about the server and client shared secret becoming compromised (due to a server security breach), then the administrator can externally certify each of the server storage keys to avoid impersonation of a trusted client. Finally, the content providers can use their server storage keys to create signing keys, which are in turn used to verify the machine's integrity, and binding keys, which are able to decrypt content coming from the server.

3.2 Content Transfer Protocol

The purpose of the content transfer protocol is to satisfy the first design goal: sensitive finished content will only be provided to authenticated and authorized users of physical machines running trusted hardware. The content transfer protocol assumes that all of the client TPMs have been configured by an administrator as specified in the previous section. It also assumes that the content provider has its shared secret, allowing it to authenticate and perform operations with the server storage key on each client.

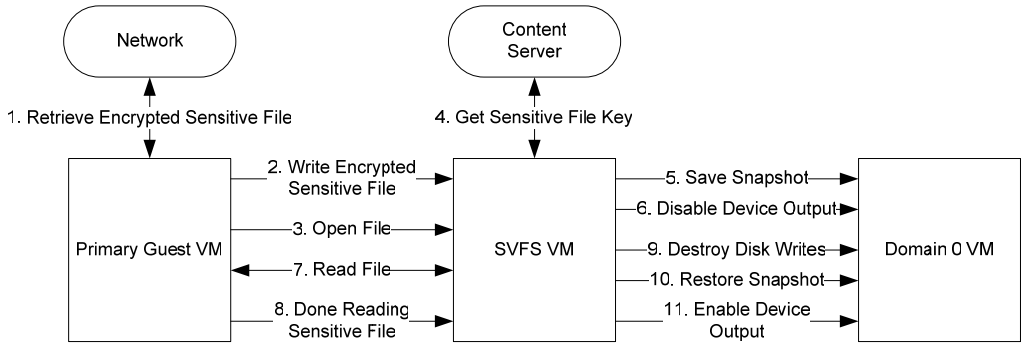

Figure 3. Architecture of the VOFS Client

Before the server transfers any content, each piece of content should be encrypted with a unique file key. Here, a piece of content corresponds to one e-mail, one document, etc. The server can then send the encrypted content, which may be very large, to any client because it will be unreadable without the secret file key. For file encryption, we plan on using 256-bit AES with forward chaining and random keys generated by the trusted platform module.

Once a client has an encrypted file, it needs to obtain the file key from the content provider. The content provider and client will use the following protocol to authenticate and securely transfer the file key. For this protocol, two-way authentication is not necessary. The server only needs to verify the authenticity of the client, and not the other way around. (VOFS only protects the secrecy of content on the client.)

1. The server initiates an OSAP authorization session [18] with the client, authenticating the client TPM with the shard secret.
2. If a signing key that is a child of the server storage key does not exist on the client, the server creates it using TPM_CreateWrapKey. (TPM_CreateWrapKey creates and returns a new cryptographic key of a specified type, see [18])
3. The server obtains the hash chains and integrity measurements of the BIOS, boot loader, operating system, and trusted components using TPM_Quote. (TPM_Quote signs and returns a PCR integrity register value, see [18])
4. The server compares the hash chain values to a list of trusted values. If the hash chains are not trusted, the server terminates the connection.
5. If a binding key that is a child of the server storage key does not exist on the client, the server creates it using TPM_CreateWrapKey.
6. The server encrypts the file key with the binding key and sends it to the client. The server sends a command to the client's TPM to decrypt the file key with the server's binding key using TPM_UnBind. (TPM_Unbind decrypts and returns a value encrypted with a binding key, see [18])

In this communication protocol, the server assures that the client is running trusted software before sending it the file key. All of the above commands also occur in a single authorization session. In an authorization session, each message contains the nonce of the previous message and a new nonce for the next message [18], protecting the messages from replay, deletion, and reordering. Messages in the same session are also resilient to an attack where the client reboots with un-trusted software between messages because the session state on the TPM is reset during machine reboot.

4. CLIENT ARCHITECTURE

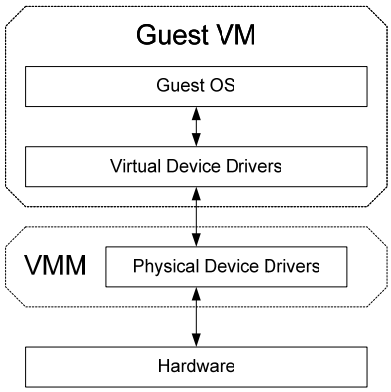

Figure 2. Virtual Machine Architecture

So far we have treated the client as a black box, assuming that it will adequately protect sensitive information given that it is running trusted hardware and software. This section delves into the details of the client's architecture, highlighting its methods for protecting content and file keys once they have been decrypted. The client architecture is based on the last two design goals for VOFS:

1. Once a machine reads sensitive finished content, all device output for that machine, except for video, sound, and limited disk output, should be disabled.
2. A user of a machine with disabled device output should be able to revert to a system state before sensitive information was read, at which point device output may resume.

The client relies heavily on virtual machine technology to make both device output restriction and machine state restoration easier and more secure. A diagram of a virtual machine system can be seen in Figure 2. The lowest software layer is the virtual machine monitor (VMM). The VMM is responsible for mediating access to hardware resources by virtual machines. The virtual machine monitor is a very thin layer, with few services and a small code base. One benefit of having a small code base is that the virtual machine monitor is much more secure from attack, whether it is from the network or from a guest virtual machine. Directly above the virtual machine monitor are the guest virtual machines. The

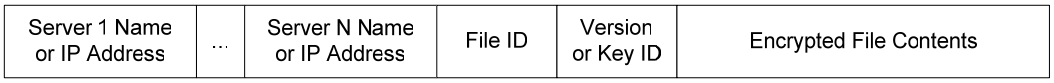

Figure 4. View-Only File Format

operating systems on the guest virtual machines interface with a number of virtual device drivers that closely resemble physical device drivers. The primary difference is that access to a virtual device goes through a physical device driver inside of the virtual machine monitor, which may examine, modify, or block requests.

In order to protect sensitive files, the VOFS client will need a way of intercepting file system requests made by a guest virtual machine. However, doing this outside of the guest VM is not straightforward because the guest VM only sends the VMM block-level disk commands. Inferring file-level operations from block commands would require complex modeling of the file system. Instead of taking this approach, we plan on using SVFS, a secure virtual file system, to govern access to sensitive files [21]. SVFS is based on NFS, a network file system, with the SVFS server running in its own virtual machine. Part or all of the guest VM's file system can reside on SVFS. The primary benefit of SVFS over a real network-based file system is that when a client VM makes a request for a file, communication between the client VM and SVFS server VM occurs via fast virtual RPCs instead of over the network. Fast VRPCs exploit co-locality of the client and server VMs to deliver performance nearly equivalent to standard virtual disk access.

The VOFS client will consist of a primary guest virtual machine, which contains all of the user's applications, an SVFS virtual machine, and the privileged Domain 0 virtual machine, which has full access to the system. The user will only be able to interact via keyboard and mouse with the primary guest VM; the other two VMs are trusted, and the user, who is not trusted, should not have access to them. A diagram of interaction between the three guest virtual machines can be seen in Figure 3. The first step in viewing a sensitive file is for the primary guest VM to load it from an external source in encrypted format. The primary guest then makes a call to open the file, which is intercepted by the SVFS VM. The SVFS VM obtains the file key, then tells Domain 0 to save system state and disable device output. Finally, it allows the primary guest VM to read the file. When the user is finished viewing the file, the primary guest tells the SVFS VM that it is finished. The SVFS VM will again contact Domain 0 to destroy any disk writes that occurred while viewing the sensitive file, restore the primary guest VM's system state, and re-enable device output. The remainder of this section discusses the details of these operations and how the various VOFS client components will support them.

4.1 The Primary Guest Virtual Machine

The primary guest VM contains the user's main operating system. The user has complete control over everything in the guest virtual machine including settings and applications. In the VOFS client, the primary guest is initially un-trusted. The primary guest VM also has an SVFS drive for view-only files, and a special application for accessing view-only files. Initially, we plan to implement a simple view-only manager that will take a file as a command line parameter and launch the viewing application. Our final goal, however, is to modify the operating system so that access is completely transparent to all applications.

When a user wants to access a view only file, the first step is to download the file. We plan to use the extension ".vo" at the end of a file having a standard extension to denote that the file is encrypted and view-only (i.e. "resume.doc.vo"). In addition to the encrypted contents of the original file, a VO file will contain a list of server names or IP addresses that have the file key. A VO file will also have a unique identifier that specifies the original file name on the server and a unique identifier that specifies the file version or key ID (in case the same file was encrypted multiple times with different file keys). This meta-information will allow a client who receives the file to contact the appropriate server and obtain the correct file key. A diagram of a VO file's contents can be seen in Figure 4.

4.1.1 The View-Only Manager

Once the user downloads a VO file from an e-mail attachment, website, or other location, it is time to decrypt and display the file. The manager will take the following steps to open the file, launc the display application, and return the guest VM to its original state before viewing the file:

1. If the file is not stored on the SVFS drive, copy the file to the SVFS drive.
2. Display an icon or status bar indicating that the machine is in view-only mode so that the user knows all changes are non-persistent.
3. Launch the standard viewing application for the file type or an application specified by the user. Tell the application to load the file from the SVFS drive.
4. When the application tries to open the file, SVFS will contact the content server to obtain the file key, decrypt the file, and instruct the VMM to disable device output.
5. Wait for the user to close the viewing application or click on a "Done Viewing" button, depending on the user's preference.
6. Make a special call to SVFS, *SvfsVoDestroy*, which will revert the entire state of the guest VM to a point right before it read the sensitive content. SVFS will return permission denied as the result of the file open call to let the View-Only Manager know that the user is done viewing the file and it can terminate (similar to the return value of a UNIX fork() call).

The View-Only Manager provides near-transparent access to sensitive files. During the development of the View-Only Manager, we plan to explore different options for automatically opening view-only files and reverting to the original system state. Also, we plan to investigate methods for allowing the user to continue running the original virtual machine, "fork off" the view-only VM, and seamlessly switch between the two VMs. This may be preferable to disabling all device output and restoring system state if the user was executing a task in the background, especially one involving active network connections. In this case, the sensitive file open will also act as a fork call, and the VO Manager will instead destroy the current virtual machine in step 6.

4.2 SVFS Instrumentation

In the previous section, we briefly discussed instrumentation of the file open procedure call, and the addition of a *SvfsVoDestroy* call to the SVFS VM. In this section, we will discuss methods for securely

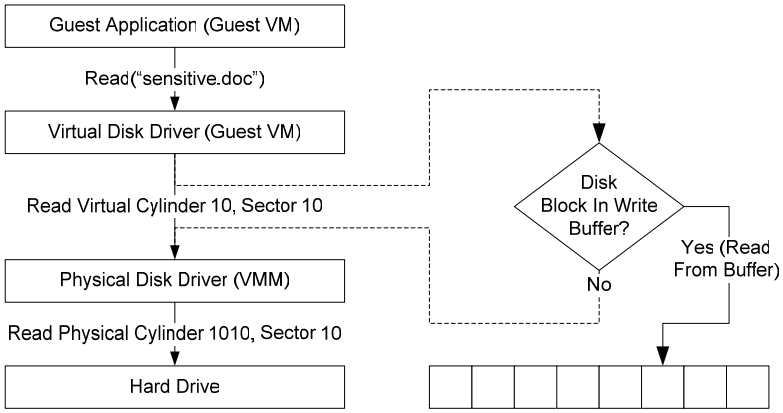

Figure 5 (a) Translation of file read to virtual disk read to physical disk read. (b) Intermediate driver with read buffer for non-persistent file system.

decrypting files inside of the SVFS virtual machine and reverting to a clean system state. Decryption of files and file keys will be done entirely inside of the SVFS virtual machine to prevent any possible information leaks. The primary guest is un-trusted, and will not have access to sensitive content until device output is disabled.

4.2.1 File Open Instrumentation

The first thing that the SVFS virtual machine does when it receives a sensitive file open call is obtain the file key from the content server using the communication protocol discussed earlier. After SVFS has obtained the file key, it is critical that SVFS stores it in physical memory and that it does not mistakenly get paged out to disk. If this were to happen and an attacker was able to steal the disk, he or she would have the file key and be able to decrypt the sensitive content. If SVFS is unable to reach the server or the server denies access to SVFS, it should return a status code indicating the nature of the failure.

After SVFS has obtained the file key, it should instruct Domain 0 to save the state of the primary guest VM's operating system. Once main memory has been saved, SVFS needs to disable all device output, including disk writes. It will do this by making a call, *VoOutputDisable*, to the Domain 0 VM. From this point forward, all device output will fail with the exception of video, audio, and disk writes. Disk writes will appear to succeed, but will be cached in memory and not persist past reboot. If main memory is unable to cache all disk writes, or does not want to for performance reasons, blocks can be written to disk in encrypted format. The details of the non-persistent file system are discussed in greater detail in the next section on Domain 0 instrumentation.

Now that device output is disabled, the SVFS VM can safely decrypt the file as it is read by the client without worrying about the data going to the physical disk in a readable format. If it does not encounter any errors, SVFS will return a value indicating success to the primary guest VM.

4.2.2 SvfsVoDestroy

After the user is done viewing the view-only file, the SVFS VM must restore the system to a clean state before it had access to sensitive data. This is accomplished by making a call to *SvfsVoDestroy*. When SVFS receives the call, the first thing it does is contact Domain 0 and call *VoDestroyNPFS* to destroy all writes to the non-persistent file system. Now that all traces of sensitive data have been removed from files written by the primary guest, SVFS can instruct Domain 0 to restore its memory using the original snapshot. Finally, the SVFS VM will tell domain 0 to re-enable device output by calling *VoEnableDeviceO*.

4.3 Domain 0 Instrumentation

Domain 0 is the most privileged execution environment. It has full control of the actual hardware on the system. As such, it runs as few services as possible to minimize its communication with outside world, and hence its chances of becoming compromised. Although the SVFS machine is responsible for all secure communication and encryption, Domain 0 provides most of the protection for the client. Domain 0 is responsible for controlling device output and managing non-persistent file access.

4.3.1 Disabling and Enabling Device Output

When the SVFS VM receives a call to decrypt a file, it first has to disable all device output from the primary guest VM. It will do this by calling *VoOutputDisable* in Domain 0. This will cause Domain 0 to cut off communication between most of its physical device drivers and the primary guest VM. It is important here to be very thorough. All serial ports, parallel ports, and USB devices are potential avenues for information leakage, not to mention network cards, infrared ports, and removable media. Even sneakier low-bandwidth channels may also be available, such as oscillating the "__ Lock" outputs to the keyboard and capturing them. In general, the only device outputs that should be allowed are local analog outputs such as the monitor and audio. If the user has a USB sound device attached, this may create a problem. Having a USB audio device may enable the primary guest VM to send arbitrary digital output to the universal serial bus. Any misbehaving device or passive sniffer can monitor the bus and read the data. USB 1.1 specifications allow for data transfer at up to 12 megabits per second. Even if the maximum rate of audio transfer is lower or there are other active devices, allowing this fairly high-bandwidth digital information leakage channel could potentially compromise the security of VOFS. We plan to initially disable all USB devices except for the mouse and keyboard, and would like to explore methods for mitigating digital information leakage channels while still allowing USB audio output.

Disabling device output on Domain 0 will require different actions depending on the type of device. For the network, packets already go through an iptables firewall in Xen [5]. Disabling network output from a particular guest VM can be accomplished by blocking all packets at the firewall. Other devices, however, will require modification to the physical drivers on the VMM. Instead of automatically translating and allowing requests from virtual device drivers, the physical drivers must be modified to check the source virtual machine and deny requests from particular VMs. We expect modification of these drivers to be relatively straightforward. They only need to enforce "deny all" or "allow all" policies based on the source virtual machine.

4.3.2 Creating a Non-Persistent File System

One part of VOFS's threat model is is theft or removal of persistent storage devices such as hard drives, USB keys, etc. In order to guarantee that no sensitive information is ever written to persistent storage in un-encrypted format, Domain 0 needs to intercept all disk write calls and prevent them from reaching the physical disk.

In the virtual machine architecture, each guest OS is assigned one or more virtual disks to store files. A virtual disk can be stored in a local file, a disk partition, or even a network file system. Because of the isolation provided by the VMM, a guest OS can only access its own virtual disk(s). As Figure 5a shows, on a guest virtual machine applications access files by invoking standard system calls such as *read()* and *write()*, as they do in a native operating system. The guest OS handles these calls and translates them into virtual disk I/O commands, which contain block-level parameters such as cylinder and sector numbers. The virtual disk I/O commands are then processed by physical disk driver on the VMM, which converts them to physical disk commands with the actual cylinder and sector numbers. The VMM can then access the physical disk with the translated cylinder and sector numbers.

Instead of allowing the disk commands to go through directly from the guest's virtual disk driver to the physical disk driver, Domain 0 can interpose an intermediate driver that creates a write buffer as seen in Figure 5b. When the virtual disk driver tries to write a disk block, the write will instead go to the write buffer. Now, when it reads a block, the VMM will first check if the block has been written and, if it has, retrieve it from the write buffer. If it has not been written, the VMM will read the block from disk. Maintaining a write buffer in memory will prevent any sensitive data from going to disk, while still giving the SVFS VM and the primary guest VM the illusion of a writable disk.

If the write buffer becomes full, the intermediate driver can encrypt parts and write them to swap space on disk. Here the intermediate driver should create an ephemeral 256-bit AES key using the TPM's random number generation capability. This key should be stored in memory, and *never written to disk*. So, if the machine is turned off, none of the encrypted write buffer, which may contain sensitive information, can be read. When encrypting disk blocks, the intermediate driver should also compute the XOR of the 256-bit key with the unique disk block number. This ensures each block is encrypted with a different key, and prevents an attacker from being able to determine if two blocks contain the same content. With this encryption scheme, it will be impossible for a guest VM to directly leak any information to disk.

5. PRELIMINARY AND FUTURE WORK

Currently, we have implemented a stand-alone version of the Secure Virtual File System [21]. In our SVFS implementation, all of each guest virtual machine's sensitive read-only files are stored within a SVFS virtual machine. Unlike a traditional networked file system, SVFS takes advantage of co-locality of the client and server on the physical hardware with its unique virtual RPC mechanism to deliver near-native disk performance. When building the Linux kernel, a standard networked file with the client and server running in separate VMs ran 51% slower than a native virtual file system. SVFS, however, ran with only 8.3% overhead. These results suggest that storing some files in a separate virtual machine is not likely to have a significant effect on system performance.

Most of the VOFS project remains to be implemented, including client and server applications for content transfer that utilize TPM technology, disabling of output for non-network devices, instrumentation of Domain 0, and instrumentation of the SVFS VM. Some exceptions include snapshot save and restore capability, which is already performed by Xen, and disabling network I/O for guest virtual machines, which we can currently do using iptables firewall rules [13]. During the implementation of the remainder of the system, we plan to iteratively perform a thorough security evaluation. The hope is to identify and eliminate any covert communication channels or weaknesses in the underlying system that could be used to bypass security mechanisms and leak sensitive information.

6. PERFORMANCE AND EVALUATION

VOFS will do an excellent job of preventing information leakage, and address a number of threats, such as hard drive theft and insider attacks, which previous systems fail to handle. However, all of these security properties come with a cost. For our evaluation, we will compare the performance of our system to a standard, un-instrumented virtual machine system. In particular, we will examine the following operations, which we believe will be the most costly:

- Saving main memory to disk before accessing a sensitive file
- Restoring main memory from disk after accessing a sensitive file
- Using a write buffer for disk I/O

6.1 Memory Save and Restore

The first intensive operation, saving a snapshot of the primary guest VM's memory to disk, can be optimized by running the snapshot in the background. What this means is that the virtual machine monitor will begin saving the virtual machine's memory, but allow the machine to continue running. If the VM happens to make a write request to a memory block that the VMM has not yet saved, then the VMM will skip ahead and save that block, then continue writing the rest of memory. The Xen virtual machine monitor performs snapshots in the background by default, and we plan to use its capability to optimize snapshots in VOFS.

In contrast to taking a snapshot, restoring main memory is more costly. This is because the entire snapshot is typically restored before allowing the machine to resume execution. It would be possible to let the system begin running immediately and restore specific portions of memory as they are read, but doing this may actually decrease performance. If memory is read non-sequentially, restoration may incur additional disk seek and rotation penalties. Without allowing immediate execution, snapshot restoration takes

approximately seven seconds for a virtual machine with 256 MB of memory on a 3.0 GHz Pentium IV machine with 1 GB of physical memory.

We hope to investigate methods for optimizing snapshot restoration, such as immediately resuming execution and paging in memory as it is read, and plan to evaluate restoration speed. In the immediate restoration scenario, we will begin using the restored virtual machine as soon as it starts executing, and compare the wall clock time required to restore the entire snapshot to that of a simple restore. If the two are comparable, and the guest virtual machine is usable during the restoration, then we will use background restoration. Otherwise, we plan to use the traditional method of restoring virtual machines.

6.2 Disk Write Buffer

Creating an intermediate write buffer in memory between the physical disk driver and virtual disk driver can add extra overhead for disk operations. The amount of overhead for reads that end up having to go to the real hard drive will be minimal; the time required to check if a block is in the write buffer will be very small compared to the disk seek and read time to grab the block. For reads that hit in the write buffer, performance may actually be better because reading from memory is much faster than reading from disk. In this case, the write buffer acts as a file cache. Writes should also go to the write buffer in memory, which again will be faster than going to disk. One major issue, however, will be the size of the write buffer. Depending on how long the primary guest VM runs in view-only mode, it could potentially write a lot of data. If this happens, then the intermediate driver will need to encrypt disk blocks, which is a CPU intensive operation, and write them to disk. Similarly, it will need to read the same blocks back from disk and decrypt them as it receives read requests. Fortunately, the primary guest VM is not likely to perform many disk writes because the user knows none of them will persist. Writes should only be needed for temporary files, which will probably not occupy more than 100 MB. Also, if disk writes do exceed the capacity of memory, the primary guest VM's only function in view-only mode is to display content. Almost no viewing applications are performance-intensive, with the possible exception of video playback, and are unlikely to slow down noticeably if they have to go through an encryption layer.

In our evaluation, we plan to investigate the effects of a write buffer. First, we will try to get an idea of how large the write buffer needs to be by running standard viewing applications on various types of files and observing temporary file size. Next, we will force the entire contents of the write buffer to go through an encryption layer and eventually to the disk. We will view various types of content with a disk-only encrypted write buffer and see if it creates any delay noticeable by a human. We also plan to run a number of benchmarks that reflect less likely usage scenarios for VOFS, such as the Andrew benchmark, and record their wall clock times compared to a normal virtual machine system. We expect that more complex operations will experience greater slow-down, but standard viewing will not take much memory and may even be faster due to in-memory caching of disk writes.

7. CONCLUSION

In this paper we present the design of a View-Only File System, a promising method for protecting finished content. VOFS will prevent unauthorized disclosure of sensitive "view-only" content by only allowing access by authorized users running trusted hardware and software. Furthermore, VOFS will disable all device output from the user's primary operating system, except for video and audio, before allowing it to access sensitive finished content. When the user is finished, VOFS will revert the user's operating system to state before it had access to the sensitive data. These security mechanisms enable VOFS to prevent sensitive information leaks, even by users who are authorized to view the information.

VOFS will utilize trusted boot technology, based on the TPM chip, to enable remote verification of software integrity. Content servers first check to make sure that clients are running the correct VOFS software before sending them sensitive information. When the sensitive content is on a client machine, VOFS uses virtual machine technology to provide isolation between trusted and un-trusted components. Sensitive data is protected inside of an SVFS VM until a Domain 0 VM shuts off device output from the user's primary OS. Once output is disabled, the SVFS VM allows the primary guest to access and display sensitive content. In this manner, VOFS provides both a high level of security and usability, allowing the user's primary guest virtual machine to access sensitive data without requiring trust or verification from the system.

While implementing the remainder of VOFS, we plan to perform thorough and iterative security tests to explore possible covert communication channels and eliminate potential vulnerabilities in the system. We also plan on evaluating the overhead of resource-intensive operations including snapshot save, snapshot restore, and disk write buffering. Our goal is to create a system that is as seamless as possible and does not hinder the user from performing authorized everyday activities, while still providing stringent security guarantees to protect sensitive finished content.

8. ACKNOWLEDGMENTS

We would like to thank Trevor Mudge for his advice on trusted hardware technology as well as the Intel Corporation and the National Science Foundation (Grant #0325332) for their support.

9. REFERENCES

[1] Adobe Systems Incorporated. Adobe Acrobat. June 2006, http://www.adobe.com/ products/acrobat.

[2] Apple Computer, Inc. iTunes. June 2006, http://www.apple.com/itunes.

[3] W. Arbaugh, D. Farber, and J. Smith. A Secure and Reliable Bootstrap Architecture. In *IEEE Computer Society Conference on Security and Privacy*, pp. 65-71, 1997.

[4] The Associated Press. VA Chief: Security Fix Will Take Time. *MSNBC*, June 8, 2006.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho R. Neugebaur, I. Pratt, A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symp. On Operating Systems Principles,* pp. 164-177, 2003.

[6] M. Blaze. A Cryptographic File System for UNIX. In *Proc. of the 1st ACM Conference on Computer and Communications Security*, pp. 9-16, 1993.

[7] D. Denning. A lattice model of secure information flow. *Communications of the ACM, v.19 n.5*, pp.236-243, 1976.

[8] K. Fisher. Apple's FairPlay DRM Cracked. ArsTechnica, June 2006. http://arstechnica.com/news/posts/ 1081206124.html.

[9] A. Freier, P. Karlton, P. Kocher. The SSL Protocol, V3.0. http://www.netscape.com/ eng/ssl3/draft302.txt.

[10] T. Garginkel, B. Pfaff, J. Chow, M. Rosenblum, D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted computing. In *Proc. of the 9th ACM Symposium on Operating Systems Principles*, pp. 193-206, 2003.

[11] S. King, P. Chen. SubVirt: Implementing Malware with Virtual Machines. In *Proc. of 2006 IEEE Symposium on Security and Privacy*, pp. 314-327, 2006.

[12] National Security Agency. Security-Enhanced Linux. June 2006, http://www.nsa.gov/ selinux.

[13] NetFilter/IPTables Project Team. What is netfilter/iptables? June 2006. http://www.netfilter.org.

[14] R. Sailer, X. Zhang, T. Jaeger, and L. Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Thirteenth Usenix Security Symposium*, pp. 223-238, 2004.

[15] R. Sandhu. Lattice-Based Access Control Models. *Computer, v.26 n.11*, pp. 9-19, 1993.

[16] J. Steiner, C. Neuman, J. Schiller. Kerberos: An authentication service for open network systems. In *Proc. USENIX Winter Conf.*, pp. 191-202, 1988.

[17] Tablus, Inc. Tablus Content Alarm NW Honored with SC Magazine Award for Best Intellectual Property Protection Solution. Februrary 2006. http://www.tablus.com/ press.php?id=29.

[18] Trusted Computing Group. *Trusted Platform Module Main Specification*. June 2006, Version 1.2, Revision 94, http://www.trustedcomputinggroup.org.

[19] VMWare, June 2006. http://www.vmware.com.

[20] H. Wen. JHymn Goes Behind Atoms and Apple To Bring DRM-Free Music. *OSDir.com*, January 2005. http://osdir.com/Article3823.phtml.

[21] Xin Zhao, Kevin Borders, and Atul Prakash. Towards protecting sensitive Files in a compromised system. *In 3rd International IEEE Security in Storage Workshop*, 2005.