Data Protection and Rapid Recovery From Attack With A Virtual Private File Server and Virtual Machine Appliances

Jeanna Matthews, Jason Herne, Todd Deshane, Patty Jablonski, Leslie Cherian, Mike McCabe
Clarkson University
{jnm, hernejj, deshantm, jablonpa, cherialr, mccabemt}@clarkson.edu

Abstract

When a personal computer is attacked, the most difficult thing to recover is personal data. The operating system and applications can be reinstalled returning the machine to a functional state, usually eradicating the attacking malware in the process. Personal data, however, can only be restored from private backups – if they even exist. Once lost, personal data can only be recovered through repeated effort (e.g. rewriting a report) and in some cases can never be recovered (e.g. digital photos of a one time event). To protect personal data, we house it in a file server virtual machine running on the same physical host. Personal data is then exported to other virtual machines through specialized mount points with a richer set of permissions than the traditional read/write options. We implement this private file server virtual machine using a modified version of an NFS server installed in a virtual machine under various virtualization environments such as Xen and VMware. We also demonstrate that by placing the user's applications in a virtual machine rather than directly on the base machine we can provide near instant recovery from even a successful attack. Specifically, we demonstrate how an intrusion detection system can be used to stop a virtual machine in response to signs of compromise, checkpoint its current state and restart the virtual machine from a trusted checkpoint of an uncompromised state. We show how our architecture can be used to defend against 21 out of 22 recent, high-impact viruses listed at US-CERT and Symantec Security Response. Finally, to quantify the overhead costs of this architecture, we compare results of I/O intensive benchmarks running directly on a base operating system and accessing data in a local filesystem to those running in a guest operating system and accessing data in an NFS partition mounted from a file server virtual machine. We find that for Xen the overhead of read intensive workloads is at most 5% and for write intensive workloads the overhead is at most 24%. For system benchmarks that stress CPU and memory performance, we see no noticeable degradation. We argue that many users would be willing to pay these moderate overhead costs to gain the security advantages of our architecture.

1. Introduction

Worms and viruses have entered the consciousness of the majority of personal computer users. Even novice users are aware of the attacks that can come in the form of email from a friend or a pop-up ad from a web site. Fully restoring a compromised system is a painful process often involving reinstalling the operating system and user applications. This can take hours or days even for trained professionals with all the proper materials readily on hand. For average users, even assembling the installation materials (e.g. CDs, manuals, configuration settings, etc.) may be an overwhelming task, not to mention correctly installing and configuring each piece of software.

To make matters worse, the process of restoring a compromised system to a usable state can frequently result in the loss of any personal data stored on the system. From the user's perspective, this is often the worst outcome of an attack. System data may be painful to restore, but it *can* be restored from public sources. Personal data, however, can be restored only from private backups and the vast majority of personal computer users do not routinely backup their data. Once lost, personal data can only be recovered through repeated effort (e.g. rewriting a report) and in some cases can never be recovered (e.g. digital photos of a one time event).

We propose the use of a specialized virtual private file server to provide added protection for personal data and virtual machine appliances to provide rapid restoration of a functional copy of system data. Personal data is housed in the virtual private file server and exported to the virtual machine appliances through specialized mount points with a richer set of permissions than the traditional read/write options. This architecture provides a number of benefits including 1) the opportunity to separate personal data into multiple classes to which different finer grained permissions can be applied, 2) the separation of personal data from system data allowing each to be backed-up and restored appropriately, 3) the ability to rapidly install or restore virtual machines containing fully configured applications and services, and 4) rapid recovery from attack by rolling back system data to a known-good state without losing recent changes to personal data.

In Section 2, we describe our architecture and its benefits in detail. In Section 3, we compare our architecture to making regular backups and other strategies for providing protection of user data and recovery from attack. In Section 4, we describe how it can be used to protect against 21 of 22 specific attacks described in the US-CERT Current Activity Reports and Symantec Security Response. In Section 5, we quantify the overhead associated with this architecture by running a variety of benchmarks on a prototype implemented using a modified version of NFS in conjunction with virtual machines in both Xen and VMware. With Xen, we find no degradation for CPU and memory intensive workloads and 5-24% degradation for I/O intensive workloads. With VMware, we also find no degradation for CPU and memory intensive workloads and 25-41% degradation on I/O intensive workloads. However, VMware supports Windows guests which are key to demonstrating rapid recovery from attack. We discuss related work in Section 6, future work in Section 7 and finally, conclusions in Section 8.

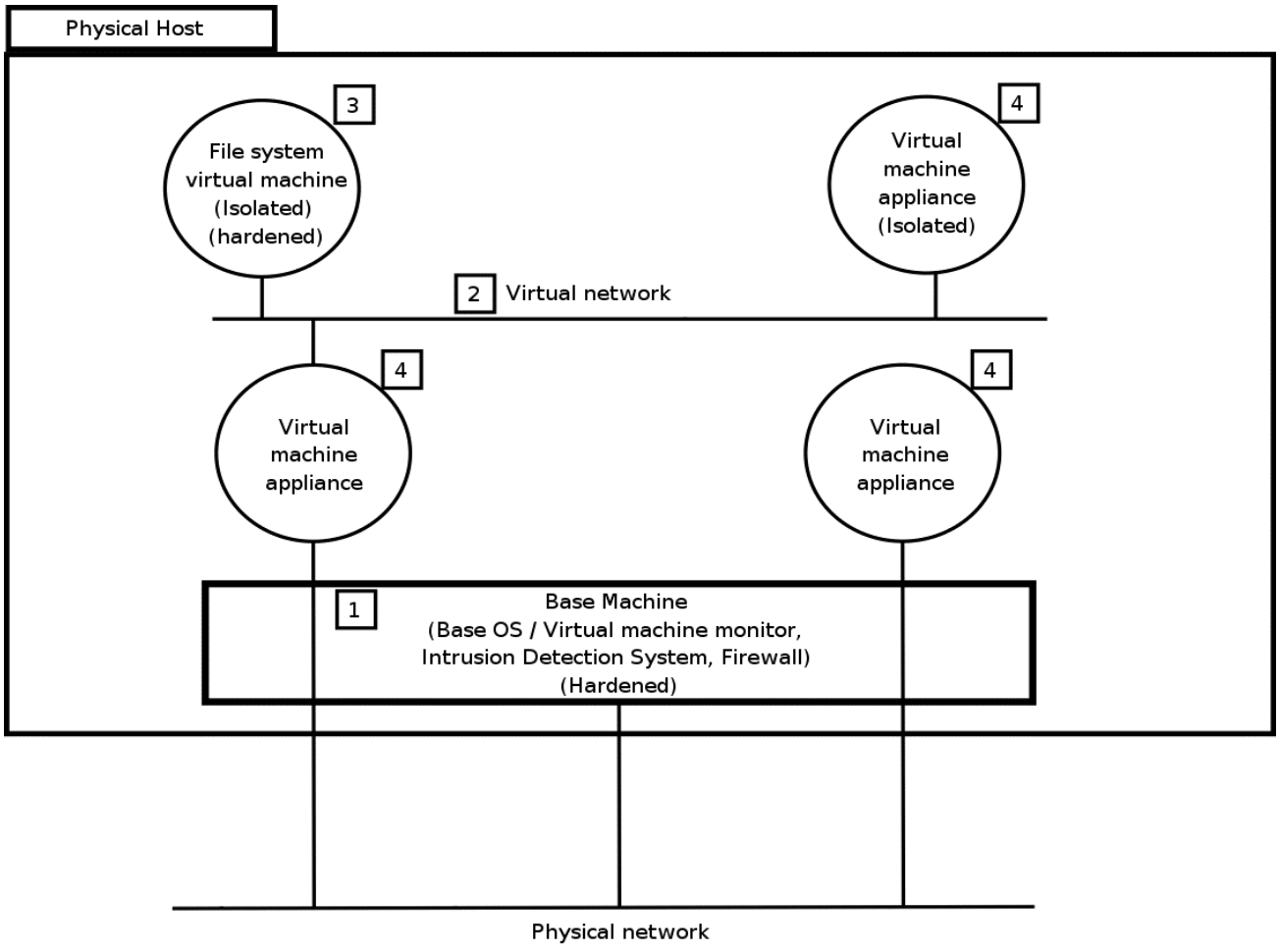
2. Architecture

Figure 1 illustrates the main components of our architecture. A single physical host is home to multiple virtual machines. First, there is the base machine (labeled with a 1 in the diagram). This base machine contains a virtualization environment which can be implemented as a base operating system running a virtual machine system such as VMware or as a virtual machine monitor such as Xen. Second, there is a virtual network (labeled with a 2 in the diagram) that is accessible only to this base machine and any virtual machine running on this host. Third, there is a file system virtual machine (labeled with a 3 in the diagram) which has only one network interface on the local virtual network. This file system virtual machine is the permanent home for personal data and exports subsets of this personal data store via specialized mount points to local clients. Fourth, there are virtual machine appliances (labeled with a 4 in the diagram). These virtual machines house system data such as an operating system and user applications. They can also house locally created personal data temporarily.

Virtual machine appliances can have two network interfaces – one on the physical network bridged through the base machine and one on the local virtual network. Depending on its function, a virtual machine appliance may not need one or both of these network interfaces. For example, you may choose to browse the web in a virtual machine appliance with a connection to the physical network but with no interface on the local virtual network to prevent an attack from even reaching the file server virtual machine. Similarly, you might choose to configure a virtual machine with only access to the local virtual network if it has no need to reach the outside world.

2.1. Base Machine

We have implemented two prototypes of this architecture using Xen and VMware as the virtual machine monitors. In both implementations, the base machine is used to



create the local virtual network, the file system virtual machine and the virtual machine appliances. It is used to assign resources to each of these guests. It can also be used to save or restore checkpoints of virtual machine appliance images.

We also use the base machine as a platform for monitoring the behavior of each guest. For example, in our prototype, we run an intrusion detection system on the base machine. (The base machine could also be used as a firewall or NAT gateway to further control access to virtual machine appliances with interfaces on the physical network.) The intrusion detection system can detect both attack signatures in incoming traffic and unexpected behavior in outgoing traffic. For example, it could indicate that all outgoing network traffic from a particular virtual machine appliance should be POP or SMTP. In such a configuration, unexpected traffic such as an outgoing ssh connection that would normally not raise alarms could be considered a sign of an attack.

The security of the base machine is key to the security of the rest of the system. Therefore, in our prototype, we "hardened" the base machine by strictly limiting the types of applications running on the base machine. Normal user activity takes place in the virtual machine appliances. We also closed all network ports on the base machine. (Alternatively, it would be possible to open a limited number of ports for remote administration, but since each open port is a potential entry point for attack, it is important to carefully secure each open port.)

2.2. File System Virtual Machine

We implemented the file system virtual machine using a modified version of Sun's Network File System (NFS) version 3 running in a Linux guest virtual machine. Much like the base machine, the file system virtual machine is hardened against attack by stripping away any unnecessary applications and closing all unnecessary network ports. It is easier to secure a system with a limited number of well-defined services than a general purpose machine. All the software in the file system virtual machine is focused on exporting personal data to local clients and to facilitating maintenance on that data such as backup, the creation of particular exported volumes and the setting of permissions that each client can have to the exported volumes.

The file system virtual machine is additionally protected by only being reachable over the local virtual network. Attacks cannot target the file system virtual machine directly. They could only reach the file system virtual machine by first compromising a virtual machine appliance. This would require two successful exploits – one against an application running in a virtual machine appliance and one running against the NFS server running on the file server virtual machine.

Personal data is housed in the file system virtual machine and subsets of it are exported to virtual machine appliances. This allows you to restrict both the subset of data a virtual machine can access as well its access rights to that data.. For example, if you have a virtual machine appliance running a web server, you limit it to read-only access to a directory containing the data you want to make available on the web.

You can export portions of your user data store with different permissions in different virtual machine appliances. For example, you may mount a picture collection as read only in the virtual machine you use for most tasks and then only mount it writeable in a virtual machine used for importing and editing images. This would prevent your collection of digital photos from being deleted by malware that compromises your normal working environment. Similarly, you may choose to make your financial data accessible within a virtual machine running only Quicken or you may choose to make old, rarely changing data read-only except temporarily in the rare instance that you actually do want to change it.

We also implemented a richer set of mount point permissions that allow "write-rarely" or "read-some" semantics. Specifically, we modified the NFS server to add read and write rate-limiting capability to each mount point in addition to full read or write privileges. One can specify the amount of data that can be read or written per unit of time. For example, a mount point could be classified as reading at most 1% of the data under the mount point in 1 hour. Such a rule could prevent malicious code from rapidly scanning the user's complete data store.

Figure 2 shows an example of an /etc/exports file with read and write limits. The first line of the example exports file will allow the client at 192.168.0.2 to write 30000 bytes in a 3600 second (1 hour) time frame. The second line limits the client at 192.168.0.3 such that it can only read 1k of data in a 20 minute period. Read-limiting and write-limiting parameters can be used separately or together in the same export to achieve maximum flexibility.

```
/share1 192.168.0.2(rw,sync,writelimit=30000,wlimreset=3600)
/share1 192.168.0.3(rw,readlimit=1024,rlimreset=1200)
/share2 192.168.0.4(rw,readlimit=1024,rlimreset=1200,
                    writelimit=30000,wlimreset=3600)
```

Figure 2: Example /etc/exports file.

In order to facilitate this type of mount point permission configuration, modifications had to be made to the NFSv3 server implementation inside the Linux kernel. Specifically, modifications were made to the functions that process all NFS write and read requests (nfsd_write and nfsd_read). Code was added to track the amount of data that each client reads and writes. If the client accessed more that the specified limit, the new code will deny the access. Once per time interval, the variables that track the amount of data read/written for that client are reset to 0. The changes in the Linux kernel are estimated at 500 lines and changes to nfs-util to support parsing the new options in /etc/exports are estimated at 200 lines. Note that unmodified NFSv3 clients can be used with our modified server.

These read and write limits are good for preventing attacks that attempt to act on all the available data in quick succession. For example, a read limit would thwart an attack that attempted to scan through all the user's data looking for credit card numbers while still allowing a user to read a moderate number of their own files. It would not, however, stop malware that introduced deliberate delays in order to read slowly through a data store. Nevertheless, it would be difficult for attackers to predict the required delay. It also increases the time required for a successful attack allowing more time to detect it through other means.

These read and write limits are just one example of a richer set of mount point permissions that can be used to help protect against attack. Append-only permissions (i.e. the ability to add new files but not modify or delete existing files) could be used to prevent removal or corruption of existing data. (SELinux has support for append-only file systems of this type [LOSM01].) For example, a directory containing photos could be mounted append-only in one virtual machine appliance allowing it to add photos, but not to delete existing photos. Another example would be restricting the size or file extension of files that are created (e.g. no ".exe" files).

2.3. Virtual Machine Appliances

Virtual machine appliances house system state much like the virtual private file server houses personal data. Each virtual machine appliance contains a base OS and any number of user level applications from desktop productivity applications to server software. They can have network interfaces on the physical network allowing communication with the outside world. They can also have network interfaces on the local virtual network over which they can mount subsets of personal data from the file server virtual machine.

There can be multiple mount points from the file system virtual machine into a client. Each mount point can have different permissions to allow finer grain control over the allowable access patterns. For example, in a single virtual machine, you might mount your mp3 collection read-only but your documents folder read-write. Or you might map your email inbox directly in local storage in a virtual machine but then move only that email you want to save onto a read-write volume exported from the personal file server.

In our prototype, we save known-good checkpoints of each virtual machine appliance. One important use of a known-good checkpoint is restoring a compromised virtual machine appliance from a trusted snapshot. Any changes made within the virtual machine appliance since the checkpoint would be lost, but changes to personal data mounted from the file server machine would be preserved. In this way, personal data does not become an automatic casualty of the process of restoring a compromised system. The checkpoint image would provide an immediately functional computing platform with access to the user's data store from the file system virtual machine.

While the base machine and file system virtual machine are hardened against attack, virtual machine appliances will, in general, continue to run an unpredictable mix of user applications including some high-risk applications. As a result, they may be susceptible to attack through an open network port running a vulnerable service or through a user-initiated download such as email or web content.

Compromised virtual machine appliances can often be automatically detected by the intrusion detection system running on the base machine. In our prototype, when the intrusion detection system detects an attack, we stop and checkpoint the compromised virtual machine, restart a known-good checkpoint of the same machine and notify the user of these actions. This process is nearly instantaneous – requiring only sufficient time to move the failed system image to a well-known location and move a copy of a trusted snapshot into place. It is worth noting that users can also trigger the restoration process manually if they suspect a compromise.

To facilitate automated attack detection and recovery of the virtual machine appliance, we use a combination of snort rules, log watchers and configurable shell scripts. Snort is an intrusion detection system that works by monitoring incoming and outgoing network traffic and can log (via Snort's Barnyard extension [Barnyard]) malicious network activity. A very simple real time log monitoring utility called Logsurfer [Logsurfer] is then used to execute pre-defined actions when it detects that a snort rule was triggered. Specifically, Logsurfer is configured to run a set of parameterized shell scripts that manipulate the virtual machine appliances (shutdown, checkpoint, and restart or reconfigure such that their network access is immediately revoked). We choose what action to take based on which virtual machine caused the fault and the severity of the Snort rule that caused the action.

The system once restarted would still have the same vulnerability that was originally attacked. To prevent future attacks, the trusted image should also be updated to patch the exploited vulnerability. The corrupted image can be saved or shipped to a system administrator for analysis and even possible recovery of data stored inside. Analysis of the corrupted image and/or secure logs collected by the virtual machine monitor [King03b] could provide clues to what needs to be modified. During this analysis and recovery process, the user would still have a functional computing platform with access to the majority of their data. This is a significant improvement over the extended down time that is often required when restoring a compromised system today.

We also limit the number of automatic restarts. For example, after three restarts of a given image, any further compromise will result in stopping the virtual machine and checkpointing, but not in restarting the "trusted" snapshot.

Users can also use the restoration process to rollback a virtual machine appliance for any other reason (e.g. they installed a piece of software and simply don't want to keep it in the system). Similarly, the restoration process can be used to recover from accidental system corruption, e.g. from a routine patch or upgrade that introduced instability into the system. Many users do not regularly apply patches and system upgrades because of the risk of instability. Stable checkpoints would encourage users to be compliant with upgrade requests by allowing them to easily experiment with the upgraded image. Reducing the risk of regular upgrades and patches is another subtle way in which virtual machine appliances enhance system security.

The number and type of applications in each virtual machine appliance can be tailored to the usage requirements and desired level of security. At one end of the spectrum, there could be only one virtual machine appliance containing all the software normally installed on a user's base machine. However, there could also be many virtual machine appliances each with a subset of the user's software.

Multiple virtual machine appliances allow finer grained control over resources required, expected behavior and the subset of personal data accessed. For example, a web server virtual machine appliance may be given read only access to the content it is serving and may be prevented from establishing outgoing network connections. Thus even if the web server is attacked, the damage done to the user's system is minimized. The attacker would also be prevented from harvesting information from the rest of the user's data store and their ability to use the system as a launching pad for other attacks would be diminished.

When each virtual machine appliance has a small number of applications, it is easier to characterize expected behavior. This makes it easier for intrusion detection software running on the base machine to watch for signs of a compromised system. It also makes it easier to configure the virtual machine appliance with a tight upper bound on the set of rights to personal data that is necessary to accomplish the task.

However, each additional virtual machine appliance requires additional memory when executing and additional diskspace to store the operating system and other common files. Multiple virtual machine appliances also make it more difficult to share data between applications. For these reasons, it is best to group as many applications with similar requirements together as possible.

Taken to the extreme however this could mean a separate virtual machine for each application. We are **not** advocating this extreme. It is easier for users when they can exchange data between applications and many applications with similar resource, data and security requirements can and should be grouped together. In our experience with our prototype, we have found that a good strategy is to isolate those applications with special security needs. For example, applications that are commonly attacked (e.g. server software such as web servers or database servers) are good candidates for their own virtual machine appliance. Similarly, applications requiring access to sensitive personal data such as financial data are also good candidates for their own virtual machine appliance.

Another crucial benefit of virtual machine appliances is that in addition to rapid recovery from attack, they also provide rapid first time installation of software systems. Anyone who has struggled for hours to install and configure software that is already running on another machine will appreciate this benefit. Preconfigured virtual machines with fully functional, preconfigured web servers, database servers, etc. would save new users hours of headaches assembling and installing all the dependencies. This is similar to the benefits of LiveCDs that allow users to experiment with fully configured versions of software without the drawbacks of slow removable, unmodifiable media.

To quantify both the time saved for a system restoration as well as for initial software installation, Table 1 lists the time it took us to install a variety of software. (We measured the times in Table 1 locally, but clearly, individual experiences will vary). The measurements listed reflect local experiments installing software when the user had already successfully installed the software at least once before. The time it takes new users to install this software could be significantly higher as they frequently run into problems that can delay them for hours or even days; witness the many installation FAQs and installation questions posted to message boards across the Internet.

The times in Table 1 can also be considered a measure of the time saved whenever a checkpoint of a virtual machine appliance is used to recover from attack. Each time a virtual machine appliance is recovered from a known-good state, this is a lower bound on the time saved in reinstallation. If it has been some time since the user installed the software, the time savings are likely to be even higher as they must spend time gathering the installation materials and possibly stumbling into some of the same errors a new user would.

Table 1: Estimated software installation/ configuration/ recovery times for experienced users

Software	Time (Hours)
Base Windows desktop install	1
Windows desktop install with an array of user level software	3-5
Base Linux desktop install (RedHat)	0.75
Base Linux desktop install (Gentoo, binary packages)	3-5
Linux base installation (RedHat) with Apache Web Server and MySQL	1.5
Linux base installation (RedHat) with sendmail	3
Spyware removal (typical)	1-2

Checkpoints can also be used to transfer working system images from one physical host to another. In fact, we view these checkpointed virtual machine appliances as a new model for software distribution. A pre-configured virtual machine appliance could be delivered to a user with well-defined resource requirements and connections to the rest of the system including the characteristics of any mount points into the user's data store.

The term "appliance" implies a well-defined purpose, well-defined connections to the rest of the world and a minimum of unexpected side effects. It also implies that little setup is required to begin use and that use does not require extensive knowledge of the appliance's workings. Physical applications typically specify their resource requirements and can be replaced with an equivalent model if they malfunction. In the case of a virtual machine appliance, a user would load it on their system and plug it into their data store by mapping its defined mount points to the exports from the local file system virtual machine. If the virtual machine appliance is attacked or malfunctioned, it would be straightforward to replace it with a new functional equivalent without losing your personal data.

Virtual machine appliances provide a new platform for value added services including configuration, testing and characterization of virtual machine appliances. Those who produce virtual machine appliances could compete to produce appliances that have the right combinations of features, that are easy to "plug in", that have a good track record of being resistant to attacks, that use fewer system resources or that set and respect tight bounds on their expected behavior. Appliances that reliably provide the advertised service without violating their resource requirements would have value to users.

Virtual machine appliances are particularly attractive in the context of open source software because any number of applications could be distributed together in a virtual machine appliance without concern for licensing requirements of each individual software package. Similarly, developers of open source software could distribute virtual machine appliances with a complete development environment including source code with all the proper libraries required for compilation and software to support debugging and testing. For commercial software, this would be more difficult but not infeasible. OEMs like Dell, Gateway and Compaq already distribute physical machines with commercial software from multiple vendors.

2.4. Virtual Machine Contracts

The base machine creates a set of resource limits for each virtual machine appliance in several ways. First, the base machine can allocate a limited amount of system resources such as memory, disk space or even CPU time to each guest. Second, the base machine can restrict access to the local virtual network and/or the physical network connection. In either case, access can be denied completely or restricted through firewall rules. Third, the intrusion detection system running on the base machine monitors the behavior of the guest for both attack signatures and otherwise "innocent" looking traffic that is simply unexpected given the purpose of the virtual machine appliance.

These limits can be thought of as a contract with the virtual machine appliance. When a virtual machine appliance is loaded on the system, a contract is established that places limits on its expected set behavior. Accomplishing the required functionality under a more restricted contract would be an aspect of a high quality virtual machine appliance.

Contracts fix a fundamental problem with running new applications. Applications typically run with a user's full rights, but there is no method for holding them accountable for doing only what is advertised. This leads directly to Trojan horse exploits in which a piece of software claims to accomplish a particular desired task

when its real purpose is its malicious unadvertised effects. On some systems, there are tools like FreeBSD's jail or chroot that allow you to run software with a restricted set of access rights. Our system automatically provides that type of protection for all software run in a virtual machine appliance that is configured with a limited set of privileges.

In our prototype, these contracts are expressed through a combination of Snort rules and limits imposed by the virtual machine monitor and by the file server virtual machine. In the future, we would love to see a unified contract language that could be used to express all aspects of the contract. Such a contract could be inspected by the user and then loaded with the virtual machine appliance onto the user's system. Tools that make the creation, inspection and validation of these contracts easier for users and developers would be a helpful addition to such a system.

3. Comparison to Full Backups and Other Strategies for Providing Data Protection and Recovery from Attack

In this section, we compare our architecture to other strategies for providing data protection and recovery from attack. One common approach to providing data protection and recovery from attack is making full backups of all data on the physical machine – both personal and system data. There are several ways to backup a system including copying all files to alternate media that can be mounted as a separate file system (e.g. a data DVD) or making an exact bootable image of the drive with a utility such as Ghost [NortonGhost].

Burning data to DVD or other removable media creates a portable backup that is well suited to restoring personal data and transporting it to other systems. Mounting the backup is also an easy way to verify its correctness and completeness. However, backups of this type are rarely bootable and typically require system state to be restored via reinstallation of the operating system and applications. For example, even if all the files associated with a program are backed-up, the program may still not run correctly from the backup (e.g. if it requires registry changes, specific shared libraries, kernel support, etc.).

Making an exact image of the drive with a utility such as Ghost is a better way to backup system data. It maintains all dependencies between executables and the operating system. Images such as this can typically be either booted directly or used to re-image the damaged system to a bootable state. However, images such as this are rarely portable to other systems as they contain dependencies on the hardware configuration (CPU architecture, devices, etc.) They are also not as convenient for mounting on other systems to extract individual files and/or to verify the completeness of the backup.

Despite the limitations of backup facilities, our system is designed to compliment rather than replace backup. Backup is still required in the case of hardware failure etc. One goal of our system is to avoid the need for restoration from backup by preventing damage to personal data and providing rapid recovery of system data from known-good checkpoints. Restoring a system from backups is often a cumbersome and manual process – not to mention an error prone one. Given the small percentage of users that regularly backup their system (and the even smaller percentage that test the correctness of their backups!), avoiding the need for restoration is a huge advantage.

Our virtual machine appliances also make backups of system data portable to other machines. System data is made portable by checkpoints of the virtual machine appliances. The virtualization system handles abstracting details of the underlying hardware platform so that guests will run on any machine. In the case of VMware, they even allow the same guests to be used on both Windows and Linux base systems.

When restoring a traditional system from a backup, users are typically forced to choose between returning their system to a usable state immediately or preserving the corrupted system for analysis of the failure or attack and possible recovery of data. With our architecture, users can save the corrupted system image while still immediately restoring a functional image. These images are also much smaller than full backups because they contain only system data not personal data such as a user's MP3 collection.

Our system also helps streamline the backup process by allowing efforts to focus on the irreplaceable personal data rather than on the recoverable system data. It also allows backup efforts to be customized to the differing needs of system data and personal data. Specifically, there is a mismatch between the overall rate of change in system data and the user visible rate of change.

System data changes at clearly predictable points (e.g. when a new application is installed or a patch is applied). Between these points, new system data may be written (e.g. logs of system activity or writes to the page file), but often this activity is of little interest to users as long as the system continues to function. For example, if a month's worth of system logs were lost, most users would be perfectly happy as long as the system was returned to an internally consistent and functioning state. Therefore, there is little need to protect this new system data between change points.

With user data, however, even small changes are important. For example, a user may only add 1 page of text to a report in an 8 hour workday but the loss of that one day of data would be immediately visible. This means

that efforts to protect user data can be effective even if targeted at a small percentage of overall data. Users also tend to retain a large body of personal data that is not actively being changed. Incremental backups can be kept much smaller when focused on changes to user data rather than system data.

Category	#	Examples	Defenses
Backdoor attacks that initiate/listen for connections to send and receive data	12	W32.Sober W32.MyDoom W32.Bagle Sasser Phatbot Backdoor.Dextenea Trojan.Mochi Backdoor.Fuwudoor PWSteal.Ldpinch.E W32.Mugly Backdoor.Nibu.J Serbian.Trojan	Block unused ports or catch unexpected behavior and revert to trusted image.
Attacks that copy infected exe's to shared folders or destroy data.	3	W32.Zafi.D W32.Netsky W32.Netad	Write restrictions to personal data and restart of compromised VM to trusted image.
Attacks that harvests email addresses and other data.	5	W32.Zafi.D W32.Sober PWSteal.Ldpinch.E Backdoor.Nibu.J W32.MyDoom	Read restrictions, detection of unexpected behavior and restart of compromised VM.
Exploit weaknesses in specific server software.	6	Santy MySQL UDF W32.Korgo Blaster Slammer Witty Worm	Block unused ports if not running this software. If running the software, catch unexpected behavior and revert to trusted image.

Table 2: Attack Classification and Defenses

Finally, a key advantage of our system relative to backups is that our architecture allows compromised virtual machines to be restarted automatically and almost instantaneously. From the time the intrusion detection system detects symptoms of an attack, the system can be restored to an uncompromised, fully functional system in minutes! Similar advantages can be achieved with network booting facilities such as Stateless Linux or system reset facilities like DeepFreeze especially if used in conjunction with personal data mounted from a separate physical file server. However, these solutions require access to server machines –the fileserver, the boot server that supplies new system images, the firewall, etc. In many ways, our architecture can be viewed as bringing these advantages of a managed LAN architecture with multiple machines to a single PC environment.

4. Protection Against Attacks

To assess how well our architecture prevents and helps recover from attacks, we began by examining several prominent lists of the most recent, most frequent and highest impact attacks. In particular, we examined the 17 US-CERT Current Activity reports [USCERT] from April 2004 to March 2005, the 6 most recent Symantec Security Response Latest Virus Threats (posted 3/24/2005) [Symantec] and a collection of 5 other well-known attacks including the Blaster and Slammer worms. For each of the attacks, we analyzed whether the architecture presented in the paper would effectively mitigate the risk of infection and/or reduce the resulting damage and data loss.

In total, we examined 22 attacks – 11 from US-CERT, 6 from Symantec and 5 others. (Note: Of the 17 US-CERT Current Activity reports, only 11 are descriptions of viruses; the remaining six are descriptions of vulnerabilities.) In Table 2, we group the majority of these attacks into 4 major categories. Of these 22, 12 use some sort of backdoor program, 3 write data in an attempt to either destroy existing data or to spread themselves by masquerading as legitimate executables in shared folders, 5 read through data in an attempt to harvest email addresses or other information and 6 exploit weaknesses in specific server software. In total, 21 out of the 22 viruses display one or more of these 4 categories of behavior. The numbers reported in Table 2 sum to more than 21 because some viruses display more than one of these behavior patterns. The remaining uncategorized attack is an Excel macro virus that can corrupt personal data stored in Excel spreadsheets if that data was mounted writeable from the private file server. Our architecture would not defend against this attack.

The architecture we propose has the potential to protect against all four categories of attacks. Whether a specific system would be protected against a given attack depends the limits placed on the virtual machine appliances in the system (e.g. limits on their access to the personal data store, rules monitoring their network activity, etc.). Most important are the restrictions on personal data mounted from the file server machine to prevent data loss. The tighter the bounds that can be placed on the data access needs of a virtual machine appliance the better.

For all categories of attack, if the attack can be limited to a single virtual machine appliance, then the worst case outcome is that this virtual machine must be

rolled back to a trusted checkpoint. This is quick relative to traditional reinstallation or restoration. For each category of attack, there are additional levels of protection.

We can defend against backdoor programs and programs that exploit specific server software by blocking all unneeded ports using firewall software at the base OS or virtual machine monitor level. On a base operating system, this may not always be possible because some ports exploited by viruses must be left open for legitimate reasons. For example, the blaster worm infects systems via the Microsoft Windows DCOM RPC service that listens on TCP port 135. Most VM's will not need access to this port so it will be blocked by default which completely removes any threat that the Blaster virus will infect those VM's. Some VM's may need access to TCP port 135 and on these systems you would not block it. In this case, an intrusion detection system on the base machine could monitor for and recover from many of these attacks.

Viruses that spread by writing legitimately named executable files to share and user data areas can be stopped with a simple file server rule that prevents executable files from being written to certain locations.

Viruses that harvest user data for email addresses and other information like credit card numbers and passwords can also be defended against by file server rules. Most user's email archives do not often need to be traversed in full. This implies that any attempt to read every single piece of an email archive could be an unauthorized attempt to harvest data. A file server rule that limits the amount of data that can be read in a given time interval can be used to thwart such an attack. The effects of viruses that destroy massive amount of user data, like W32.Netad, can be minimized by using limited writing file system rules similar to those discussed previously.

One key benefit of our architecture is that it does let people experiment without worry in a virtual machine appliance with no access to the file server VM. In the worst cast, the user may shutdown and restore that specific VM to the last known-good state and any problems are corrected. Today, many users are hesitant to download email attachments or click on certain web links for fear of getting a virus. Some email clients (example: certain versions of Microsoft Outlook Express) even completely block all incoming executable attachments. While this solution does keep the user safe, it forces them to be extremely conservative in their online behavior. Our solution allows the users to download even "risky" files from inside a VM that mounts no data from the private file server. If, for example, an attachment does contain a virus the user can easily restore the virtual machine and no permanent damage is done. In this way, we provide a safe "playpen" in which users can test uncertain actions without fear of the consequences.

5. Overhead of a Virtual Private File Server and Virtual Machine Appliances

In the first four sections, we have presented the benefits of our architecture. However, both running programs in a virtual machine environment and mounting data from a file server virtual machine will clearly introduce overhead and reduce performance. The crucial question is how much must we pay in terms of overhead for the benefits of data protection and rapid recovery.

To answer this question, we constructed two prototype systems. One using Xen on Linux to host Linux guest virtual machines and one using VMware on Windows to host both Linux and Windows guest virtual machines. We constructed both prototypes as described in Section 2 with a file server virtual machine running a modified version of the NFSv3 file server and a virtual network segment that isolated the file server from the outside world.

There are several other VM monitors available besides Xen and VMware, but a direct comparison of each system is beyond the scope of this paper. We chose to use Xen and VMware for our testing for several reasons. First, they are virtualization environments rather than simply emulators and therefore they provide the needed isolation between guests. They can also limit the resources consumed by each guest. They both offer flexible tools for creating virtual networks inside the machine. They both support Linux and a variety of other UNIX style operating systems.

A key advantage of VMware is that it supports Windows guests and also runs on Windows as a base operating system. This was important given that the majority of viruses target Windows. There are plans to implement XenoWindows or Windows guests for Xen, but this is not currently available [Xen03].

Another advantage of using Xen and VMware is that there are already published results that quantify their overhead relative to base Linux on a variety of workloads including SpecInt, SpecWeb, Dbench and OLTP [Xen03, CDD+04]. These results show little degradation on Xen guests for all workloads and substantial degradation for VMware guests on some workloads like OSDB and SpecWeb.

To these previously published results, we added two classes of measurements. First, we ran IOzone [IOzone] to quantify the impact on I/O intensive workloads for which we expected to see the most degradation. Second, we ran Freebench [Freebench] to quantify impact on a range of benchmarks that stress CPU and memory performance.

We primarily ran our tests on a Pentium 3, 1-Ghz machine with 512 MB of memory. We deliberately chose an older machine with a moderate amount of

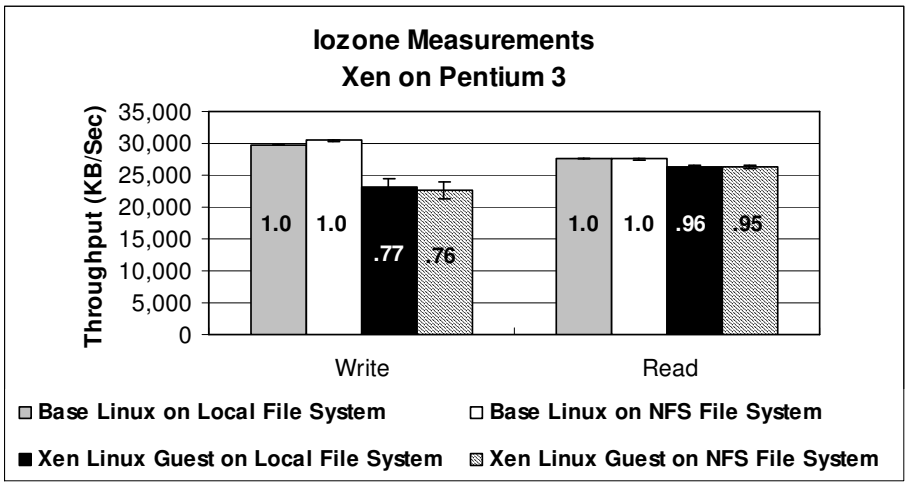


Figure 3: IOzone Read and Write tests, 4 KB accesses to a 2 GB file

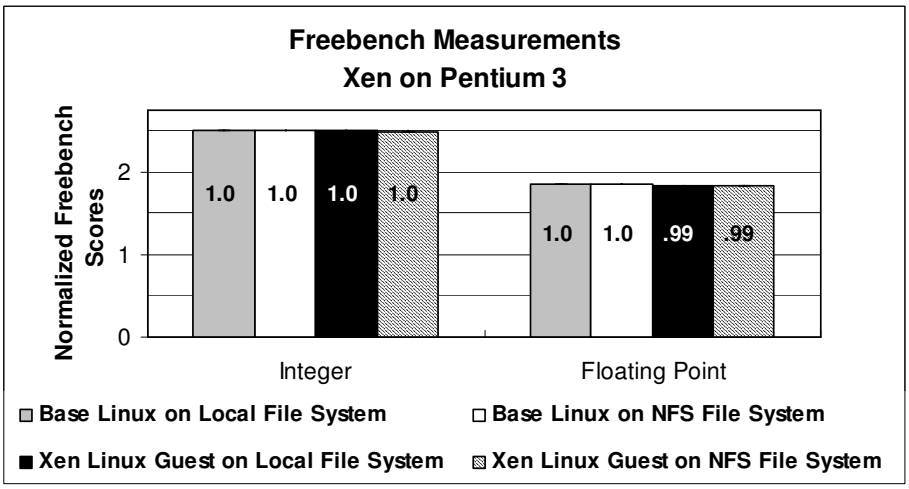


Figure 4: Freebench Tests

memory to evaluate the feasibility of our approach for average users. We also ran the same measurements on a Pentium 4, 2.8 Ghz machine with 1.5 GB of memory. As expected, they showed the same or lower overheads than measurements on the Pentium 3. (The raw scores were of course higher).

IOzone consists of 16 different I/O intensive access patterns including read, write, random read, random write, random mixture and others. For each test, IOzone specifies the size of the access and the total size of data being touched. We focused on 4 KB reads and writes to a file larger than physical memory (2 GB). Each measurement reported is the average of 3 runs and standard deviation is indicated with error bars. For some measurements, the standard deviation is so low that the error bars are not visible.

Figure 3 shows the results of running IOzone under four configurations. First, we configured Linux as the base operating system and accessed files on a local file system. This represents the traditional configuration with no virtual machine appliances and no personal file server. Second, we ran an NFS file server virtual machine and ran IOzone on the Linux base machine (not in a virtual machine appliance) and accessed files mounted from the NFS file server virtual machine. Third, we ran IOzone in a XenoLinux guest and accessed files that were local to that guest. Finally, we ran IOzone in a XenoLinux guest and accessed files mounted from the NFS file server virtual machine. This final configuration represents our proposed architecture while the second and third configurations represent intermediate points that help isolate the overhead due to different components of the system.

Figure 3 shows that the full cost of our architecture on I/O intensive workloads is 24% overhead for writes and 5% overhead for reads. The majority of that overhead is due to running the benchmark in the Linux guest. The cost of using a local NFS server is low.

Figure 4 shows both Integer and Floating point results from running Freebench in the same four

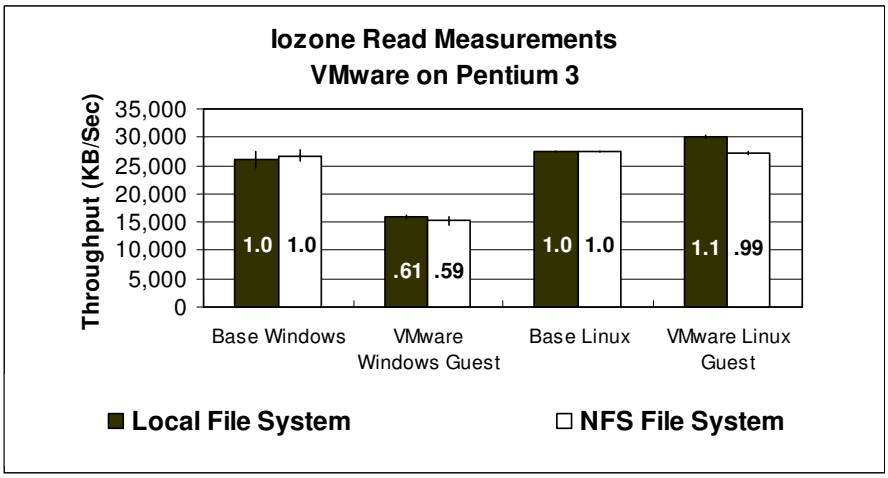


Figure 5: IOzone Read tests, 4 KB accesses to a 2 GB file

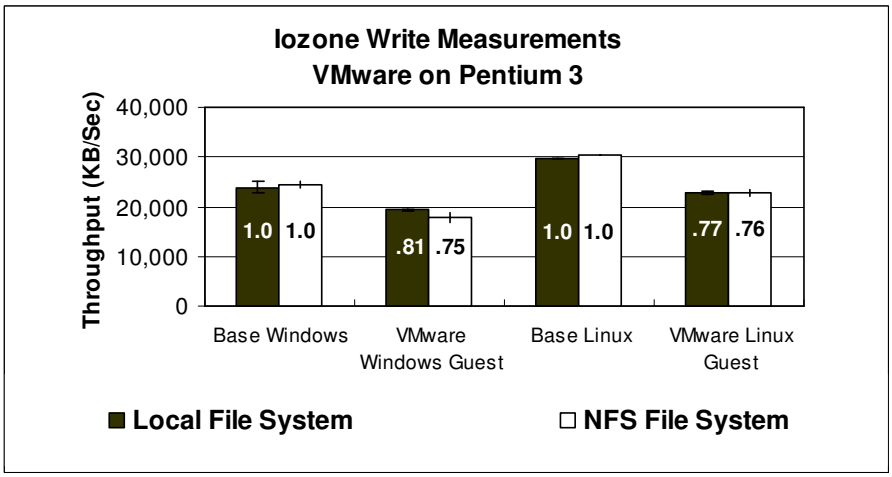


Figure 6: Iozone Write tests, 4 KB accesses to a 2 GB file

configurations. Again each measurement represents the average of 3 runs. Standard deviation is so low that the error bars are not visible. Here the overhead of our architecture is at most 1%.

Figures 5 and 6 show the results of running IOzone read and write tests under 8 configurations. These configurations are 1) base Windows with a local file system, 2) base Windows with data mounted from the NFS virtual file server machine, 3) Windows running in a VMware guest with a file system local to the guest, 4) Windows running in a VMware guest with data mounted from the NFS virtual file server machine, 5) base Linux with a local file system, 6) base Linux with data mounted from the NFS virtual file server machine, 7) Linux running in a VMware guest with a file system local to the guest, and finally, 8) Linux running in a VMware guest with data mounted from the NFS virtual file server machine.

Configurations 1 and 5 (base Windows and base Linux to local file systems) represent traditional configuration with no virtual machine appliances and no personal file server. Configurations 4 and 8 represent our proposed architecture. As in Figures 3 and 4, the other 4 configurations represent intermediate points that help isolate the overhead due to different components of the system.

The results in Figures 5 and 6 show that the overhead of running a Windows guest is substantially higher than for Linux guests. The Windows guests using files mounted by NFS experience 25% degradation for reads and 41% degradation for writes. Interestingly, the Linux VMware guests experience nearly the same degradation as the Linux guests under Xen. We do not show the Freebench scores for these configurations, but they also show no significant degradation.

From these results, we conclude that the overhead of running a private NFS file server to house data is small in all configurations. Similarly, the overhead of running CPU and memory intensive workloads in a virtual machine appliance is also small in all

configurations. I/O intensive workloads experience a significant degradation – 1% to 25% for Linux guests under Xen or VMware and 25% to 41% for Windows guests under VMware. The average system workload would be a mix of CPU, memory and I/O activity and therefore in general the overhead would be lower even for Windows guests. Finally, we did see even lower overhead in tests on a Pentium 4, 2.8 Ghz machine with 1.5 GB of memory.

Given the power of modern computers, however, users often have resources to spare and we suspect that many users would be willing to incur these overheads to gain the added security and predictability of the virtual machine appliances we have described. Studies of user satisfaction with file servers have shown that users prefer a system with lower yet predictable performance to a system with higher average performance with unexpected periods of poor performance [RG96]. This is an even more extreme example. We imagine many users would be happy to incur a 25% reduction in speed to be able to download email attachments and surf the web secure in the knowledge that if attacked they can simply restart the compromised virtual machine appliance.

6. Related Work

Virtual machine technology has been available for over 30 years on mainframes [VM370, Goldberg74], but it is relatively new for commodity personal computers [Denali02, VMware, Xen03].

Today, virtual machine technology on commodity hardware has many applications including providing multiple operating systems platforms on the same physical machine, building efficient honeypot machines and providing a stable platform for OS development. As the computing power and storage capacities of commodity platforms increase, additional applications of virtual machine technology are explored.

We used VMware and Xen in our prototype, but there are many other virtualization or emulation systems available like Qemu, Bochs, VirtualPC, Win4Lin, UserModeLinux, FAUmachine and others. These all have some limitations for our purposes including the degree of resource isolation among guests, the expected performance and the number of supported platforms. Our goal in this work was the construction of a prototype to illustrate our architecture, rather than a thorough comparison of virtualization/emulation systems.

Several systems have used virtual machine technology to enhance system security and fault tolerance. Bressoud and Schneider developed fault-tolerant systems using virtual machine technology to replicate the state of a primary system to a backup system [Bressoud96]. Dunlap et al used virtual machines to provide secure logging and replay [Dunlap02]. King and Chen used virtual machine technology and secure logging to determine the cause of an attack after it has occurred. Reed et al used virtual machine technology to bring untrusted code safely into a shared computing environment.

We focus on the related problem of rapid system restoration and protection of user data. We are unaware of another system that has separated user data and system data in the way we are proposing and optimized the handling of each to provide rapid system restoration after an attack. There are system reset facilities such as DeepFreeze or Windows System Restore. DeepFreeze restores a system to a trusted point each time the system is rebooted. Any and all changes made while the system is running will be lost on reboot. This is similar to the concept of a machine appliance. However, DeepFreeze does not facilitate moving appliances between physical machines and therefore loses many of the benefits of our solution (a new model for system distribution, a way to transfer corrupted images for analysis and recovery, etc.). Windows System Restore is another system restore utility that monitors and records changes to system data on a Windows system (registry files, installed programs, etc.) It supports rolling back to a previously identified snapshot. It is limited to Windows and only supports rollback of changes to specific system files not generic recovery from attacks that could compromise other parts of the system. Neither DeepFreeze nor Windows System Restore attempt to protect user data from damage or loss.

7. Future Work

We would like to pursue improvements to this architecture in two primary categories – improvements to the base machine and improvements to the file server virtual machine.

We would like to design a unified system and language for expressing all aspects of the contract between the virtual machine and the system (base machine and file server virtual machine). Tools could be written to write, compare and validate the contracts for each virtual machine. We would also like to see richer contract semantics including richer mount point semantics as we described. We would also like to see rules that link multiple resource types. For example, for an email virtual machine appliance, it would be great to specify that the amount of data written to the personal data store should be no more than that received via POP or IMAP. This would require linking of network monitoring and file system monitoring which may be challenging, but it could prevent email viruses from writing unexpected data or overwriting/damaging a large portion of the user's data store.

We would also like to see improvements to the user interface of the base machine to support use by novice users. Right now, it would be difficult for novice

users to manage our prototype, but we don't believe this is fundamental. In fact, if presented correctly, we think novice users may really appreciate the model of computing appliances that can be started, stopped, restarted and even replaced with a functionally equivalent appliance that uses fewer resources. VMware, for example, has an easy-to-use GUI, but we would prefer to have such a GUI for a hardened virtual machine monitor than for an application running on top of a commodity OS like Windows or Linux.

Finally, we would like to investigate enhancements to the file server appliance. We have done some limited experiments with AFS as well as NFS.

Ideally, we would like to implement a file server that logs data modifications in the personal data store system on a per client basis. Logging data modifications would allow the file server virtual machine to roll back changes that were made by a compromised virtual machine even before suspicious behavior is detected. The length of the log could be based on an estimate of the time required to detect an attack. Such a facility would give the opportunity to defend against the Excel macro attack described in Section 4. Such a delay would also provide an undo facility to recover from accidental destruction of personal data.

8. Conclusions

We have presented an architecture in which personal data is protected in a file server virtual machine and in which trusted checkpoints of virtual machine appliances house system data and enable rapid recovery from attack.

We have described numerous benefits of this architecture including automatic restart of virtual machine appliances when signs of an attack are recognized by an intrusion detection system and the ability to protect data on the file server virtual machine that has a richer set of mount point semantics and that is not even directly accessible from remote machines. We have also shown how this architecture reduces the risk of regular patches and upgrades, facilitates efficient incremental backups that focus on unrecoverable personal data, and the ability to send checkpoints of compromised machines for analysis and recovery. We have discussed how virtual machine appliances can be used not only to provide rapid recovery from attack, but also to make first time installation and configuration of software easier.

Finally, we have quantified the overhead of two prototype implementations of our system and found the overhead to be negligible in many configurations. Specifically, we find that for Xen, the overhead of read intensive workloads is at most 5% and for write intensive workloads the overhead is at most 24%. For system benchmarks that stress CPU and memory performance, we see no noticeable degradation. For Windows guests in VMware, we see no noticeable degradation on system benchmarks and between 25% and 41% degradation for I/O intensive workloads.

It is our sincere hope that systems like the ones described in this paper can be used to free average users from constant fear of attack.

9. References

[Barnyard] Snort, Barnyard, http://www.snort.org/dl/barnyard, Accessed March 2005.

[Bressoud96] T. Bressoud and F. Schneider. Hypervisor-based fault tolerance. ACM Transactions on Computer Systems, 14(1):80-107, February 1996.

[Bochs] Kevin Lawton, Bochs, http://bochs.sourceforge.net/, Accessed March 2005.

[Chen01] P. Chen and B. Noble. When virtual is better than real. Proceedings of the 2001 Workshop on Hot Topics in Operating Systems (HotOS), p. 133-138, May 2001.

[CDD+04] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, J. Matthews. "Xen and the Art of Repeated Research", 2004 USENIX Annual Technical Conference FREENIX Track, June 2004.

[Dbench] dbench-3.03, http://samba.org/ftp/tridge/dbench, Accessed March 2005.

[Deepfreeze] Faronics, Deepfreeze, http://www.faronics.com/html/deepfreeze.asp, Accessed March 2005.

[Denali02] A. Whitaker, M. Shaw, S. Gribble. Scale and Performance in the Denali Isolation Kernel. Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), ACM Operating Systems Review, Winter 2002 Special Issue, pages 195-210, Boston, MA, USA, December 2002.

[Dike00] J. Dike. A User-mode Port of the Linux Kernel. Proceedings of the 4th Annual Linux Showcase & Conference (ALS 2000), page 63, 2000.

[Dike01] J. Dike. User-mode Linux. Proceedings of the 5th Annual Linux Showcase & Conference, Oakland CA (ALS 2001). pp 3-14, 2001.

[Dike02] J. Dike. Making Linux Safe for Virtual Machines. Proceedings of the 2002 Ottawa Linux Symposium (OLS), June 2002.

[Disco97] E. Bugnion, S. Devine, K. Govil, M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. ACM Transactions on Computer Systems, Vol. 15, No. 4, 1997, pp. 412-447.

[Dunlap02] G. Dunlap, S. King, S. Cinar, M. Basrai, P. Chen. ReVirt: Enabling Intrusion Detection Analysis through Virtual Machine Logging and Replay. Proceedings of the 2002 Symposium on Operating

Systems Design and Implementation (OSDI), p.211-224, December 2002.

[Freebench] Freebench v1.03, http://www.freebench.org, Accessed March 2005.

[Goldberg74] R. Goldberg. Survey of Virtual Machine Research. IEEE Computer, p. 34-35, June 1974.

[IOzone] IOzone 3.2.35, http://www.IOzone.org, Accessed March 2005.

[King03a] S. King, G. Dunlap, P. Chen. Operating System Support for Virtual Machines. Proceedings of the 2003 USENIX Technical Conference, June 2003.

[King03b] S. King and P. Chen. Backtracking Intrusions. Proceedings of the 19th ACM Symposium on Operating Systems, p. 223-236, December 2003.

[Labtam] Labtam Inc., ProNFS Version 2.4, http://www.labtam-inc.com, Accessed March 2005.

[Logsurfer] DFN-Cert, Logsurfer, http://www.cert.dfn.de/eng/logsurf/.

[Norm98] D. Norman. The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution. MIT Press, 1998.

[NortonGhost] Symantec, Norton Ghost, http://www.powerquest.com/sabu/ghost/ghost_personal, Accessed March 2005.

[Qemu] Fabrice Bellard, Qemu, http://fabrice.bellard.free.fr/qemu/, Accessed March 2005.

[RG96] Erik Riedel and Garth Gibson. Understanding Customer Dissatisfaction With Underutilized Distributed File Servers. Proceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, September 17-19, 1996

[Spafford89] E. Spafford. Crisis and Aftermath, Communications of the ACM, 37(6):678-687, June 1989.

[StatelessLinux] Redhat Inc, StatelessLinux, http://fedora.redhat.com/projects/stateless, Accessed March 2005.

[Symantec] Symantec, Symantec Security Response, http://securityresponse.symantec.com/, Accessed March 2005.

[Tripwire] Tripwire, http://www.tripwire.org, Accessed March 2005.

[USCERT] US-CERT Current Activity, http://www.us-cert.gov/current, Accessed March 24, 2005.

[UML] Jeff Dike, http://user-mode-linux.sourceforge.net, Accessed March 2005.

[VM370] R. Creasy. The Origin of the VM/370 Time-Sharing System. IBM Journal of Research and Development. Vol. 25, Number 5. Page 483. Published 1981.

[VirtualPC] Microsoft, Virtual PC, http://www.microsoft.com/windows/virtualpc/default.mspx, Accessed March 2005.

[VMWARE] VMware, URL http://www.VMware.com Accessed March 2005.

[Win4Lin] NeTraverse, Win4Lin, http://www.netraverse.com/, Accessed March 2005.

[Xen99] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accounted Execution of Untrusted Code. Proceedings of the 7th Workshop on Hot Topics in Operating Systems, 1999.

[Xen03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield. Xen and the Art of Virtualization. Proceedings of the Nineteenth ACM Symposium on Operating systems principles, pp 164-177, Bolton Landing, NY, USA, 2003

[Xen03a] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, E. Kotsovinos, A. Madhavapeddy, R. Neugebauer, I. Pratt and A. Warfield. Xen 2002. Technical Report UCAM-CL-TR-553, January 2003.

[Xen03b] K. Fraser, S. Hand, T. Harris, I. Leslie, and I. Pratt. The Xenoserver Computing Infrastructure. Technical Report UCAM-CL-TR-552, University of Cambridge, Computer Laboratory, Jan. 2003.

[Xen03c] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt. Controlling the XenoServer Open Platform, April 2003.