

PhD Proposal: August 9th 2007

System Support for Rapid Recovery and Attack Resistance

Todd Deshane

Advisor: Dr. Jeanna Matthews

1 Abstract

We propose a system to provide resistance to attack and rapid recovery from viruses, worms, problematic system updates, and other negative system changes. Our system uses two key techniques: isolation and intrusion detection. First, for isolation, we collect user data in a file system virtual machine (FS-VM) so that system corruption does not automatically compromise it. We also isolate groups of applications from each other by placing them into virtual machines, called virtual machine appliances (VMAs), so that we can place stronger limits on their behavior. User data is exported to the VMAs by the FS-VM as needed. Second, for intrusion detection, we incorporate a standard network intrusion detection system (NIDS) and firewall into a special network virtual machine (NET-VM) as well as integrate file system access controls into the FS-VM. To support both isolation and intrusion detection, we design a VMA contract system that is used to define the acceptable behavior of each VMA in terms of network and file system access requirements as well as any device access or system resource limits. The NET-VM enforces the network-based VMA contract rules and the FS-VM enforces the file system-based VMA contract rules. We add support for our system into a modern, low overhead, open source virtual machine monitor (VMM), namely the Xen hypervisor. We discuss the design, implementation, and evaluation of our proposed system. Evaluation of our system will be both in terms of performance costs and effectiveness against various real world attacks.

2 Motivation

Computers on the Internet are vulnerable to attacks from malware such as viruses, worms, and spyware. Many systems are vulnerable simply because users fail to install the latest security updates. Still other computers with the latest security updates have out-of-date virus definitions and are therefore vulnerable to attack. Even computers with the latest security updates and virus definitions are susceptible to brand new attacks, often referred to as zero day attacks.

Malware has two main negative effects for users: (1) User data is corrupted, deleted, or used for the purposes of the attacker. (2) The user's system is controlled by the malware and/or the user's system cannot be used by its owner for normal use. We propose a system that can both protect user data and also detect and recover quickly from attacks.

Fully restoring a compromised system can be an agonizing 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 (i.e. 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 challenging to restore, but it can be restored from public sources. Personal data, however, can only be restored 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 (i.e. rewriting a report) and in some cases can never be recovered (i.e. digital photos of a one time event).

Malware is not the only source of problems on a computer system. Software bugs and conflicting software packages are also sources of problems for users. Software bugs can cause system instability as well as data corruption. Conflicting software packages can cause applications not to work properly and can lead to system instability. Users may install a different combination of packages and apply patches in a different order making it nearly impossible to thoroughly validate all scenarios. The resulting software and patch combinations on a system can cause bugs. For these reasons, many users delay or refuse to install updates and patches. Our system can make applying updates and patches safer, since it allows the system to be restored in case of a problem.

3 Background and Related Work

This work builds on a significant amount of research in the areas of security, virtualization, and the combination of the two.

3.1 Virtualization

Virtual machine technology is not a new concept. The basic software, including the virtual machine monitor (VMM), was pioneered in the 1960s by IBM. Originally designed for IBM System/370 machines, the original IBM VMM has co-evolved with the IBM hardware that it runs on and is now known as the z/VM hypervisor and System z (zSeries) server hardware, respectively.

Virtual machine technology for mainstream PC platforms is a more recent development. One problem is that the x86 system, unlike IBM hardware, was not designed to be virtualizable [34]. This introduced additional overhead and complexity in virtualization. Also, until recently, personal computers were not typically configured with sufficient memory to support multiple, simultaneously running VMs. As PCs increased in power and memory prices fell, virtualization became more feasible for commodity platforms and a number of commercial and open source virtualization products were introduced. The Disco project [35]

was the first to create a VMM that ran on commodity NUMA hardware. Members of the Disco team later founded VMware [17], which is the commercial pioneer of virtual machine technology on x86 hardware.

We will focus on two types of virtual machine technology: (1) Full virtualization, which allows running unmodified operating systems on top of the VMM, and (2) Paravirtualization, which requires minor modifications to an operating system, making it aware that there is an underlying VMM. The first approach, full virtualization, which is used by VMware, used binary rewriting techniques to handle the non-virtualizable instructions on the fly. This allowed unmodified guest support, but with a fairly significant performance cost. Paravirtualization, first coined by Denali [16] and then popularized by Xen [8], brought with it evidence that virtualization benefits could be achieved with low overhead. Paravirtualization requires modification to the guest operating systems, thus avoiding the use of the non-virtualizable instructions. This approach has the advantage of better performance, but with some modification to the guest required, and thus it is ill-suited for use with closed source operating systems. When Xen released their performance numbers at SOSP 2003, a team of us at Clarkson University published independent verification of these results and extended the comparison to Linux running on an IBM zServer and also demonstrated that virtualization benefits could be realized on older hardware, with a low performance overhead [41].

In 2005, commodity hardware that supports virtualization was released by Intel, in the form of the VT-x virtualization extension, which was followed shortly after by AMD's virtualization extensions AMD-V. This marked the beginning of a new era of software and hardware virtualization co-evolution, this time instead of on IBM mainframe hardware, the cooperation is being done in the commodity market. The hardware virtualization extensions allows for unmodified guest operating systems to run on Xen and other virtualization systems. Second and third generation virtualization hardware extensions are already being worked on and planned for. These next generation hardware extensions will improve performance of virtual disk and network I/O and guest page table (also referred to as a shadow page table) management. This co-evolution process of commodity hardware, will make virtualization a ubiquitous part of commodity computing both in the server and desktop markets.

3.2 Security

Similar to virtualization, network security has been an area of concern since the early computers connected to the ARPANET. The Internet was founded on openness and trust and although this has provided a wealth of information and services that are available online, Internet security problems took many by surprise. Retrofitting protocols and applications for security has proven difficult.

The first major Internet worm appeared in 1988, but large scale computer virus and worm outbreaks have seen a surge since the Internet (in particular the world wide web) gained popularity in the late 1990s. Early viruses and worms were often merely somebody proving that they could break into something for the challenge and thrill of it, but attackers have continually increased the negative effects (often referred to as the payload) of their viruses and worms. We will refer to viruses, worms, trojan horses, spyware, and the like generically as malware.

Criminal malware networks are now a lucrative business. Organizations can hire these malware networks, or botnets, to perform distributed denial of service (DDOS) attacks against other companies or organizations that they don't like. Other known criminal attacks

include taking control over a user's data and demanding large sums of money to release it back to the user.

The vast majority of attacks exploit a vulnerable application such as one that does not safely handle its own program buffers and is susceptible to a buffer overflow attack. The most common entry point for an attack is over the Internet, either exploiting a common service or tricking a user or application to run malware. The attack can either have an immediate payload or open a door for the attacker to exploit at a later time.

To defend against these attacks, antivirus, firewall, and intrusion detection systems have been developed. To be effective, antivirus systems need to periodically scan all files and memory for things that match attack signatures. These operations are expensive and slow down overall system performance. Firewalls need to be configured to allow and deny the proper access to particular applications. This process is usually quite complicated for the average user, given the fast-changing ways that applications are being used and developed. Intrusion detection systems also rely on either attack signatures or anomalous activity to recognize the existence of possible malware on a user's system.

The main drawback to existing software is that it often requires users to accept or deny actions of software packages. It does not typically detect malware based on classifying acceptable behavior. It may have the limited ability to open ports for particular applications, but does not offer a fine granularity of control in an easy-to-use fashion. Users typically only have a choice to allow or deny an application the access it needs to make a connection or to modify files. Often, users don't understand the details of what applications should be allowed to do. They may understand at a high level that a particular application should or should not have access to their email or photo collection, but may not be aware of the details of how to adequately protect against data corruption or break-ins.

The general problem of malware is currently one of the biggest IT problems that exists. It is not uncommon for users to throw out a malware infested computer and buy a new computer. Malware writers and networks continue to grow and improve in complexity, and the current defense strategies that are in place do not give the user the right tools to defend themselves against possible attacks. With our solution, we aim to help improve this bleak situation.

There has been a lot research in the area of Internet security. The malware problem is not likely to be solved simply by known security mechanisms. New and innovative attacks are found all of the time. The process of capturing and characterizing malware in order to develop signatures to recognize it is a never-ending battle, since attackers only need to make small variations to an attack and it may get through unnoticed by up-to-date security systems. It is clearly a game of escalation. It is best to be able to recognize an attack before it happens (attack prevention), but second best is to recognize that an attack has occurred and be able to recover from it. Virtual machines offer a place to carefully study and characterize malware in an isolated environment.

3.3 Combining Virtualization and Security

The combination of virtualization and security has been an attractive option, since the beginning of virtual machine technology. Much like virtual memory protects the memory of one process from modification by another process, virtual machine technology protects the data in one VM from modification or compromise by other VMs.

It is advantageous to be able to treat two virtual machines as if they were two physical machines. For example, separate VMs can be created for testing or development. This “sandboxing” can isolate the production environment from errors even while still using the same physical system. The focus of this work, however, goes beyond simply sandboxing system components by systematically monitoring the interactions of the system components and the outside world.

Similar work in this area has dealt with using 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 [27]. Dunlap et al. used virtual machines to provide secure logging and replay [28]. King and Chen used virtual machine technology and secure logging to determine the cause of an attack after it has occurred [4]. Reed et al. used virtual machine technology to bring untrusted code safely into a shared computing environment [29]. Zhao et al. used virtual machines to provide protection against root kits [42].

3.4 Strategies for Providing Data Protection and Recovery from Attack

We focus on the problems 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 that has optimized the handling of each to provide rapid system restoration after an attack.

System reset facilities such as DeepFreeze [30] or Windows System Restore [31] tackle some aspects of these problems. 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.

4 Current Status: Prototype System

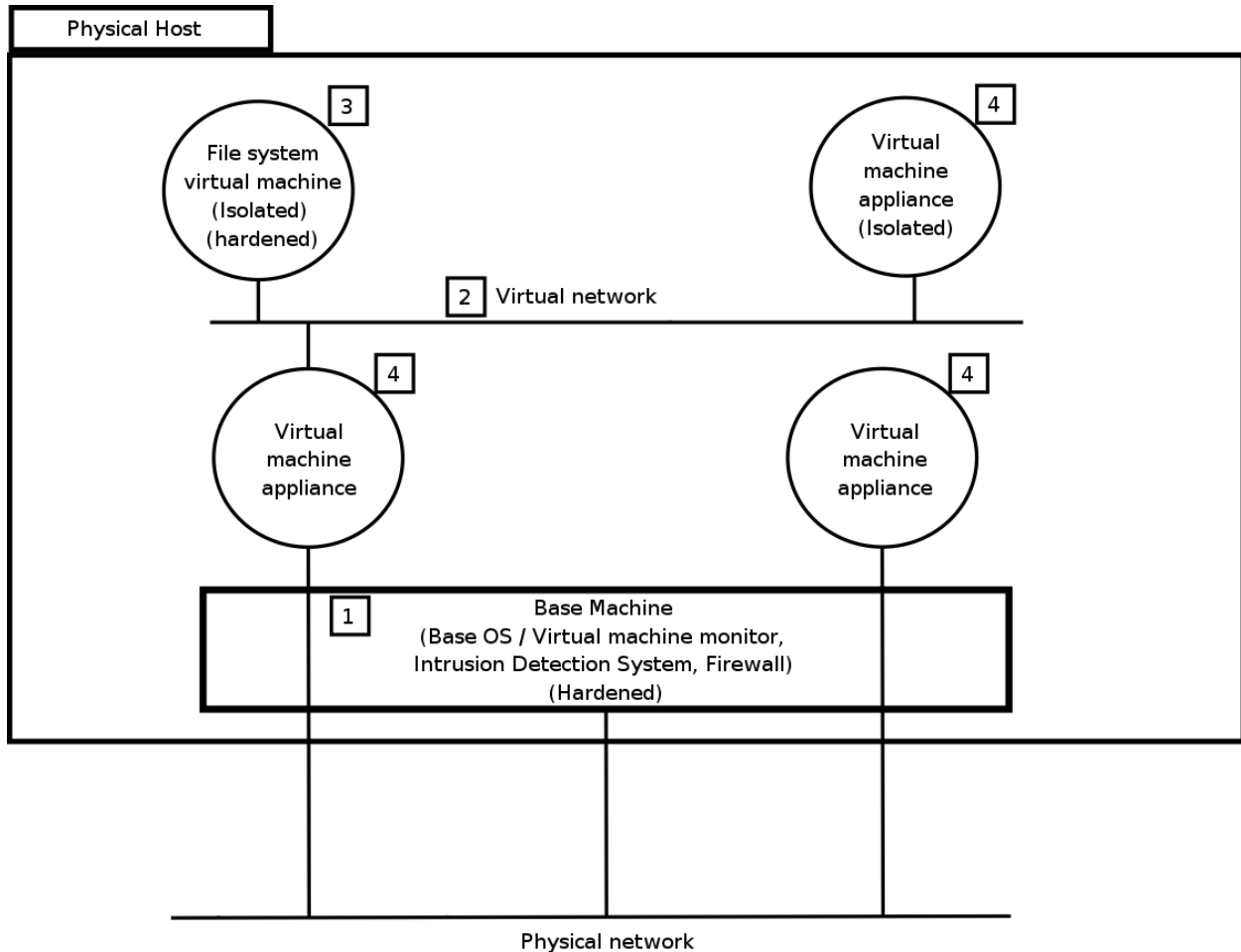
We published the paper “Data Protection and Rapid Recovery From Attack With A Virtual Private File Server and Virtual Machine Appliances” [36], in which we presented a working prototype of the system as well as an evaluation of the system. We found that the system both protected well against attack and performed well for a variety of workloads. We first present the prototype system, along with the evaluation of it. We will then discuss the enhancements that we would like to make to it and a detailed plan for accomplishing them.

4.1 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 that 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) that has only one network interface on the local virtual network. This FS-VM 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 FS-VM. 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.

Figure 1: Architecture of Prototype System



4.2 Hardened 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 FS-VM, 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. (In this case, the NET-VM was in Dom0. We will discuss making the construction and integration of a separate NET-VM component in the Plan of Work section).

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. In particular, we ran no server applications, so there were no open 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. We also installed no client software such as web browsers and email clients that are common entry points for attack. All normal user activity takes place in the virtual machine appliances. A machine with no open ports and few applications running is significantly harder to attack than a general purpose machine with many open ports and user applications that are interacting with the outside world.

4.3 Configuration of the File System Virtual Machine

Personal data is housed in the file system virtual machine (FS-VM) 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 writable 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.

Our design also supports a richer set of mount point permissions that allow "write-rarely" or "read-some" semantics. Specifically, we will add read and write rate-limiting capability to each mount point in addition to full read or write privileges. Using this design, 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. 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 [1]). 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 (i.e. no ".exe" or files marked as executable files).

4.4 Configuration of Virtual Machine Appliances

Virtual machine appliances house system state much like the virtual machine file system 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 FS-VM.

There can be multiple mount points from the FS-VM into a client. Each mount point can have different permissions to allow finer-grained 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 FS-VM.

While the base machine and FS-VM 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.

4.5 Checkpoint and Restart of Virtual Machines

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 FS-VM 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 FS-VM.

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.

Once restarted, the system would still have the same vulnerability that was originally exploited. 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 the data stored inside could possibly be recovered. Analysis of the corrupted image and/or secure logs collected by the virtual machine monitor [4] 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 (i.e. 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 (i.e. from a routine patch or upgrade that introduced an 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.

4.5.1 Application Mix in a Virtual Machine Appliance

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 the 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. Attackers 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 disk space 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 (i.e. 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.

4.5.2 Rapid First Time Installation

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. Pre-configured virtual machines with fully functional, pre-configured web servers, database servers, etc. would save new users hours of headaches assembling and installing all of 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 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 re-installation. 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

4.5.3 Model for Software Distribution and Value-Added Services

Checkpoints can also be used to transfer working system images from one physical host to another, allowing a user to take a working system on an old PC and move it painlessly to a new machine. This same ability to move working systems' images between machines could also be viewed as a new model for software distribution and/or value-added services. 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, users 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 FS-VM. If the virtual machine appliance is attacked or malfunctioned, it would be straightforward to replace it with a new functional equivalent without losing 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 the 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 of the proper libraries required for compilation and software to support debugging and testing. For commercial software, this would be more difficult, but not impossible. OEMs like Dell, Gateway, and HP already distribute physical machines with commercial software from multiple vendors.

4.6 Comparison to Full Backups

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 (i.e. a data DVD) or making an exact bootable image of the drive with a utility such as Norton Ghost [5].

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 re-installation of the operating system and applications. For example, even if all of the files associated with a program are backed up, the program may still not run correctly from the backup (i.e. if it requires registry changes, specific shared libraries, kernel support, etc.).

Making an exact image of the drive with a utility such as Norton 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), it is important to reduce the number of situations in which restoring from backup is required.

Our virtual machine appliances also make backups of system data that are portable to other machines. System data is made portable by the 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 to possibly recover 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 (i.e. when a new application is installed or a patch is applied). Between these points, new system data may be written (i.e. 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.

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 file server, the boot server that supplies new system images, the firewall, etc. In many ways, our architecture can be viewed as bringing the advantages of a managed LAN architecture with multiple machines to a single PC environment.

5 Evaluation

We evaluate the system in two ways: (1) in terms of how well it protects against attack, and (2) overall performance and overhead.

5.1 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 [6] from April 2004 to March 2005, the 6 most recent Symantec Security Response Latest Virus Threats [7], 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 our 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 6 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 writable from the private file server. Our architecture would not defend against this attack.

The architecture we propose has the potential to protect against all 4 categories of attacks. Whether a specific system would be protected against a given attack depends on the limits placed on the virtual machine appliances in the system (i.e. 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 FS-VM 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 re-installation 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 VMs will not need access to this port so it will be blocked by default, which completely removes any threat that the Blaster worm will infect those VMs. Some VMs 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 lets people experiment without worry in a virtual machine appliance with no access to the FS-VM. In the worst case, 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 (i.e. certain versions of Microsoft Outlook Express) even completely block all incoming executable attachments. While this solution keeps 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.

Table 2: Attack Classification and Defenses

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.
---	---	---	--

5.2 Performance of the Prototype Systems

Running programs in a virtual machine environment and mounting data from a FS-VM 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 4 with a FS-VM running a modified version of the NFSv3 file server and a virtual network segment that isolated the file server from the outside world.

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.

At the time of the prototype, a key advantage of VMware was that it supports Windows guests. This was important given that the majority of viruses target Windows. Since the prototype, Xen has added support for Windows guests too, with the help of hardware virtualization support (recall that this technology was released in 2005 first by Intel (VT), followed shortly after by AMD (AMD-V)).

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 [8] [9] [10]. 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 [11] to quantify the impact on I/O intensive workloads for which we expected to see the most degradation. Second, we ran Freebench [12] to quantify impact on a range of benchmarks that stress CPU and memory performance.

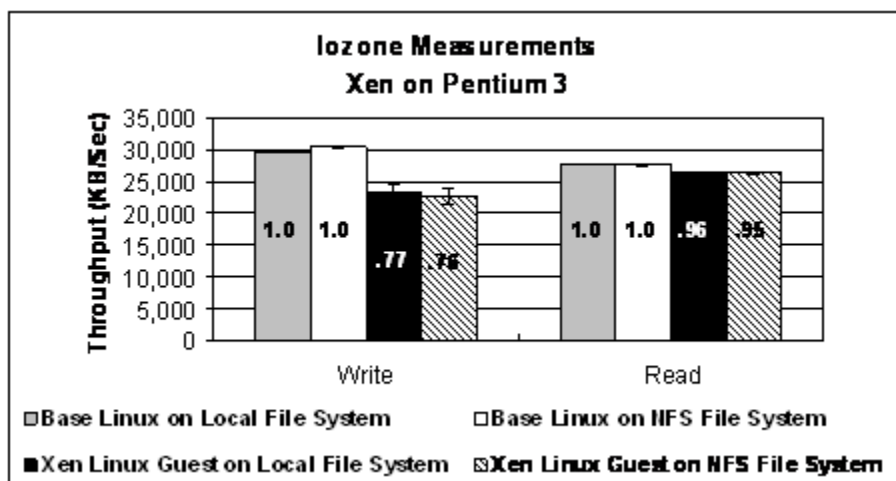
We primarily ran our tests on a Pentium 3, 1- GHz machine with 512 MB of memory. We deliberately chose an older machine with a moderate amount of memory to evaluate the feasibility of our approach for average users. We also ran the same measurements on a Pentium 4, 2.8 GHz machine with 1.5 GB of memory. As expected, they showed the same or lower overhead 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 2 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 XenLinux guest and accessed files that were local to that guest. Finally, we ran IOzone in a XenLinux 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 2 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. We ran the Integer and Floating point tests from Freebench in the same four configurations. We did not include these in the graph, but the overhead of our architecture was at most 1% for these tests.

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



Figures 3 and 4 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.

Figure 3: Iozone Read tests, 4 KB accesses to a 2 GB file

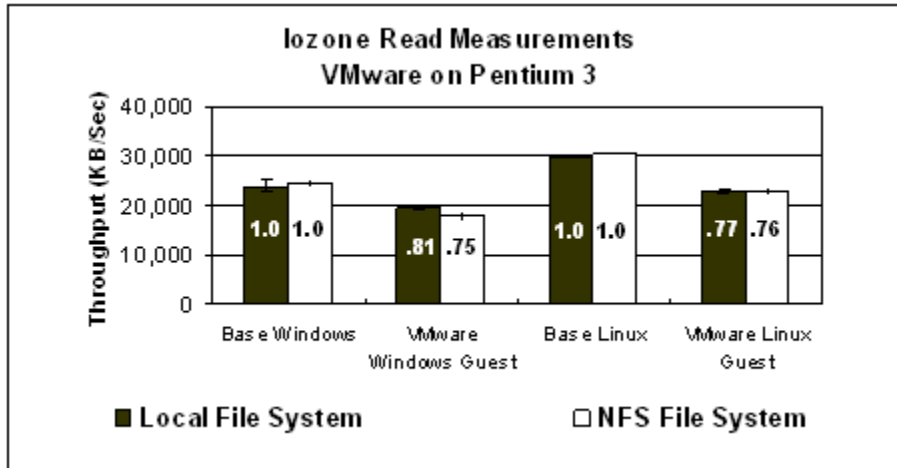
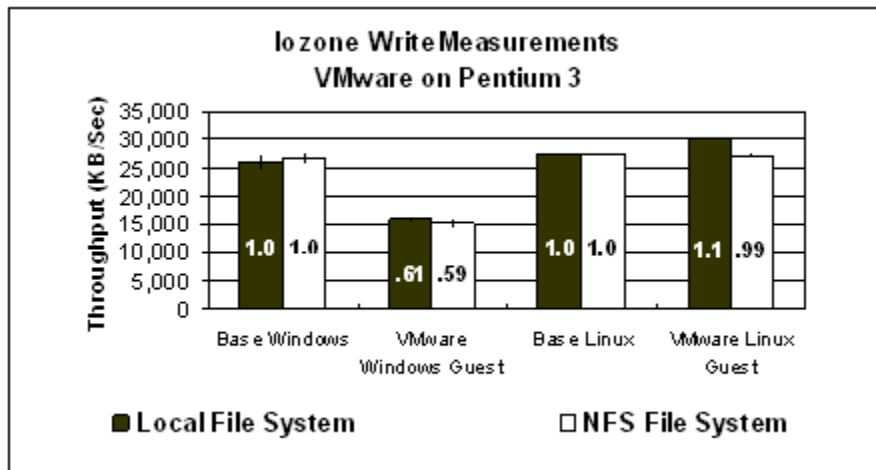


Figure 4: Iozone Write tests, 4 KB accesses to a 2 GB file



Configurations 1 and 5 (base Windows and base Linux to local file systems) represent a traditional configuration with no virtual machine appliances and no personal file server. Configurations 4 and 8 represent our proposed architecture. As in Figure 2, the other 4 configurations represent intermediate points that help isolate the overhead due to different components of the system.

The results in Figures 3 and 4 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 ran the Integer and Floating point tests from Freebench scores in these configurations as well and again, there is 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 this overhead 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 [13]. 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.

In our initial prototype, we presented an architecture in which personal data is protected in a FS-VM and in which trusted checkpoints of virtual machine appliances house system data and enable rapid recovery from attack. We 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 FS-VM that has a richer set of mount point semantics and that is not even directly accessible from remote machines. We also showed 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 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 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.

6 Plan of Work

We propose to complete additional enhancements to the initial prototype in four primary areas:

- (1)** Construction and integration of a separate NET-VM component.
- (2)** A comprehensive virtual machine appliance contract system including new file system and network rules.
- (3)** Tight integration of the NET-VM and FS-VM into the trusted virtual machine software support layer of Xen.
- (4)** Evaluation of the system including performance, resistance to attack, and constructing example appliances.

We will discuss each of these in detail, but first we begin with an overview of the modified system architecture.

Figure 5 shows the new architecture of the system. We based the new implementation exclusively on the Xen hypervisor and not VMware. In our first prototype of the system, we built two versions, one using VMware as the VMM and another using Xen. At the time, VMware was the only one of the two that supported Windows guests. The majority of attacks against desktop machines target the Windows platform as it is the most popular desktop platform and we wanted to be able to evaluate these attacks. However, Xen can now support Windows guests also with hardware support available in most Intel and AMD chips released after 2005. In Xen, this type of guest is called a hardware virtual machine (HVM) guest. In addition to supporting Windows guests, Xen is also open source, which allows us to easily integrate our enhancements into the system.

Figure 5 represents a single physical host, which contains multiple virtual machines. The system is built on top of the Xen hypervisor (VMM). The privileged management VM is called Dom0 in Xen terminology (labeled with a 1 in Figure 5). Dom0 administers the virtual machines, for example granting resources, and performing administrative tasks such as start, stop, restart, and shutdown of the guests. It can also be used to save or restore checkpoints of virtual machine images.

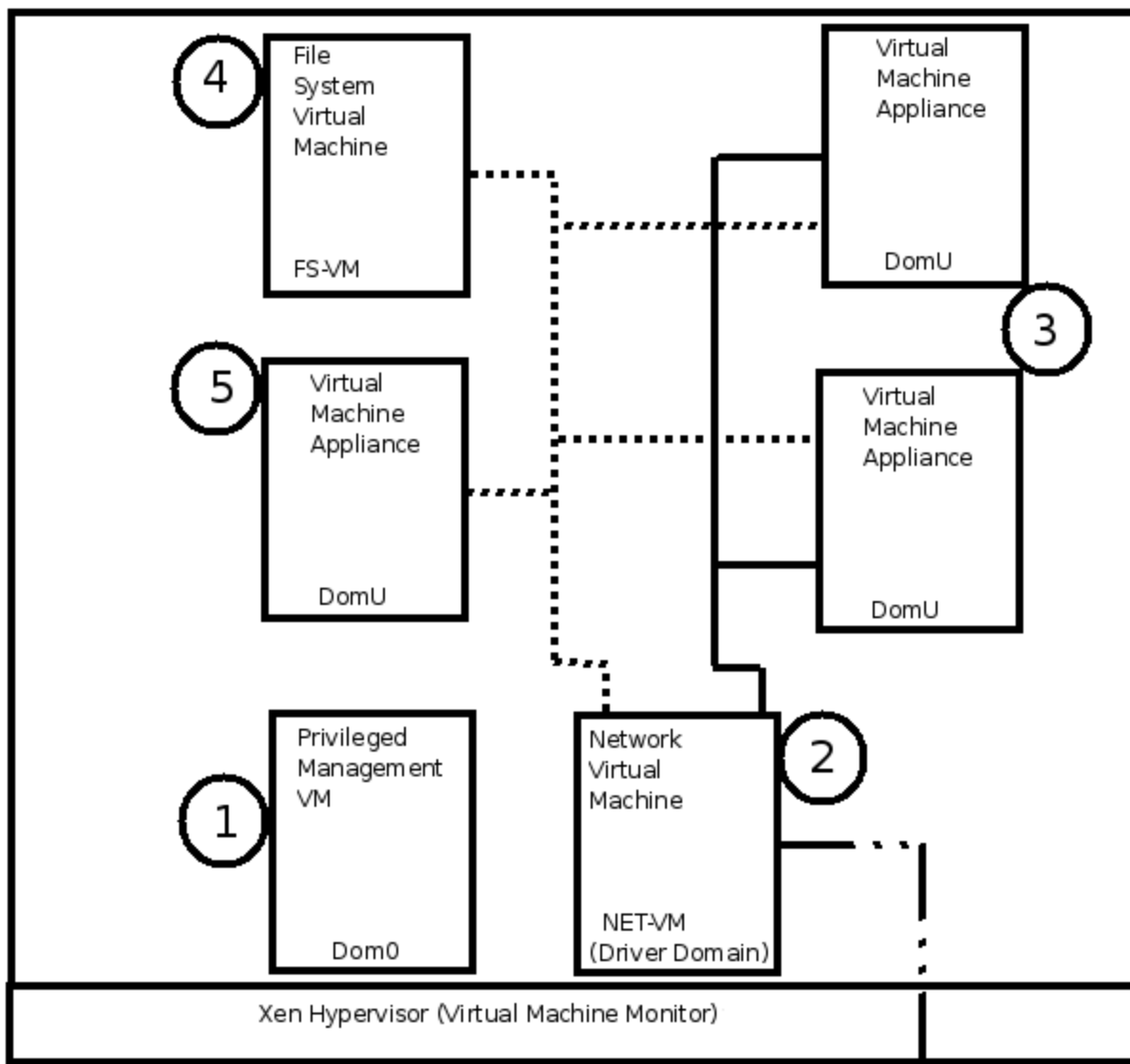
Figure 5: Improved Architecture (Based on Xen)

Key:

———— Internal Network A (NAT)

..... Internal Network B (Isolated)

- - - - Internet Connection



Xen also supports the concept of driver domains, in which a guest virtual machine (DomU in Xen terminology), is granted direct access to one or more physical devices, such as a network interface card. We use a driver domain for the network intrusion detection system (NIDS) and a firewall component of our architecture, calling it the NET-VM (labeled with a 2 in Figure 5). The purpose of the NET-VM is to monitor network activity, both by watching for known attacks using a database of attack signatures and also detecting anomalous network activity such as too many connections in a given period of time.

Finally, the storage of local user data in our system is protected and isolated by our FS-VM (labeled with a 4 in Figure 5), which will monitor access from the virtual machine appliances to the user data that it stores. The FS-VM is well-isolated from the external network, and will not be granted any access to incoming or outgoing Internet connections. In order for the FS-VM to be compromised, an attacker must first break through the defense setup by the NET-VM, and then also compromise a virtual machine appliance that has access to the data stored in the FS-VM.

Our current architecture supports two distinct types of virtual machine appliance guests (DomUs) - those that need access to the Internet (labeled with a 3 in Figure 5), such as browsers and email clients, and those that only need access to local files (labeled with a 5 in Figure 5), such as many office applications and other editing applications. The two distinct internal networks, A and B, denoted with solid and dotted lines in Figure 5, respectively, serve as the mechanism to isolate the virtual machine appliances appropriately.

6.1 Construction and Integration of a Separate NET-VM Component

To facilitate automated attack detection and recovery of the virtual machine appliances, we will construct a specialized network virtual machine (NET-VM). The NET-VM will use a combination of Snort rules, log watchers, and configurable scripts. Snort is an intrusion detection system that works by monitoring incoming and outgoing network traffic and can log (via Snort's Barnyard extension [2]) malicious network activity. A very simple real time log monitoring utility, called Logsurfer [3], can then be used to execute pre-defined actions when it detects that a Snort rule was triggered. Specifically, Logsurfer can be configured to run a set of parameterized 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 NET-VM also enforces limits on each VMA's network use as specified in its virtual machine contract.

6.2 A Comprehensive Virtual Machine Appliance Contract System

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 users to run software with a restricted set of access rights. Our system will

automatically provide that type of protection for all software that is run in a virtual machine appliance that is configured with a limited set of privileges.

In our prototype, these contracts were expressed through a combination of Snort rules and limits imposed by the virtual machine monitor and by the FS-VM. For this implementation, we propose a unified contract language that could be used to express all aspects of the contract. This contract can 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 will also be developed as needed.

In our proposed implementation, the Xen hypervisor, along with the privileged Dom0, FS-VM, and the NET-VM enforce a set of resource limits for each virtual machine appliance in several ways. The FS-VM restricts access to user data stored in the file system. The NET-VM can restrict access to the local virtual network and/or the physical network connection. Access can be denied completely or restricted through firewall rules.

The NET-VM monitors the behavior of the guest for both attack signatures and otherwise anomalous 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 behavior. Accomplishing the required functionality under a more restricted contract would be an aspect of a high quality virtual machine appliance. Producers of virtual machine appliances could charge more for their appliance if its behavior was well-characterized with a clear contract.

6.2.1 Overview of Xen's Current Guest Configuration Files

We will integrate our unified contract system into Xen's current guest configuration files. A Xen DomU guest configuration file is a file that contains the set of parameters that are passed to Xen that specify the resources that will be available to the DomU guest. The most direct comparison in a non-virtualized environment would be a list of specifications of the physical hardware. The configuration file specifies system resources and devices that are granted to the VM by the Xen hypervisor (i.e. 128 MB of memory and a disk image).

A simple example is shown in Listing 1. The kernel parameter specifies the system kernel the guest should use to boot from. The memory parameter specifies how much RAM the guest will get. The name parameter is used as another way to refer to a guest (guests are automatically given an ID upon creation). The vif parameter specifies the virtual network interface(s) that the guest will have. For each interface a number of options can be specified, all of which are optional, since Xen will give default values for the interface properties, such as MAC address (each interface specification is enclosed in single quotes and separated by commas). The disk parameter specifies the disks that the guest have.

Listing 1

```
kernel = "/boot/vmlinuz-2.6"  
memory = 128  
name = "email-client"  
vif = [ " ]  
disk = [ 'tap:aio:/images/email-client.img,hda,w' ]
```

6.2.2 Additions to the Xen Configuration file

The heart of the new contract system that we are proposing is additional contract rules in two main categories: file system rules and network rules.

6.2.2.1 File System Rule Language

The current Xen DomU guest configuration files have a disk parameter that specifies the physical and virtual disks to which the DomU is granted access. The general format for the disk parameter is:

```
disk = [ '<access method>:<path to the actual disk>,<device as DomU will see it
mounted>,<permissions>' ]
```

There can be one or more of these tuples (enclosed by single quotes and separated by commas), each representing a different DomU disk device.

We propose to add a new access method, named fsmv. The fsmv access method will indicate that the DomU is allowed to mount the specified area of the personal data store protected by the FS-VM. We then use the permissions field to pass our file system rules. More specifically, we allow the permissions field to be a list of identifiers for pre-defined file system rules.

A specific example of our additions is shown in Listing 2. We add an fs_rule parameter to define a rule and then use the permission field of the disk line to associate a particular rule or set of rules with an fsmv mount point. In this example, we mount the user's email stored in the fsmv to their home directory within their email virtual appliance (a DomU) and set a read limit of 1024 bytes in 5 seconds to that mount point.

Listing 2

```
#Example file system rule portion of a contract for an email client.
```

```
fs_rule = [ 'id=1, read,1024,5' ]
#read at most 1024 bytes of data in 5 seconds
disk = [ 'fsvm:/mnt/email, /home/user/mail, fs_rule=1' ]
```

In Listing 3, we build on Listing 2 to show a more complex example, with more mount points.

Listing 3

```
# A more complex example file system rule set for an email client.
```

```
fs_rule = [ 'id=1, read,1024,5' ]
# read at most 1024 bytes of data in 5 seconds

fs_rule = [ 'id=2, append,1024,4' ]
# append at most 1024 bytes of data in 4 seconds.

fs_rule = [ 'id=3, write,320,4' ]
# write at most 320 bytes in 4 seconds
```

```
# The email mount point is accessible to the email client, and fs_rules # with id=1 and
id=2 are applied
disk = [ 'fsvm:/mnt/email, /home/user/mail,fs_rule=1:2' ]

# The email mount point is accessible to the email client, and fs_rules # with id=1 and
id=3 are applied.
disk = [ 'fsvm:/mnt/email, /home/user/attachments,fs_rule=1:3' ]
```

6.2.2.2 Network Rule Language

We propose to add a permissions option to the vif parameter similar to our file system rules. However, in this case, we do not need to specify a new rules language since several effective rule languages already exist for this purpose. A specific example of using our additions is shown in Listing 4. For this example, we will use a iptables file for our rules. Similar to using file system rules, we add the network_rule option that would pass additional options to the network intrusion detection system (Snort).

The general format of a network_rule will be

```
[ 'id=#, <type of rule i.e. snort, iptables>, <full path to the file>' ]
```

We also use the rate option (which is already built into Xen) of the vif parameter to create a network choke point for observing and limiting network traffic, if desired. Adding a rate limit of 2 Mb/s means that this DomU guest can send and receive at most 2Mb/s of data across the network.

Listing 4

```
#Email client example continued

network_rule = [ 'id=1, iptables, file=/etc/iptables/email_client' ]

network_rule = [ 'id=2, snort, file=/etc/snort/rules/email_client' ]

vif = [ 'rate=2Mb/s, network_rule=1:2' ]
```

6.3 Tight Integration of the NET-VM and FS-VM into the Trusted Virtual Machine Software Support Layer

In our proposed design, the virtual machine appliance contracts are expressed in the Xen configuration file for the DomU guest, which already enforces memory limits, access to disk devices, and network interfaces that are imposed by the Xen hypervisor. We propose to enhance the configuration file syntax to support our new contract language. To support improved network and file system rules, we implement a tighter integration of the NET-VM and FS-VM into the Xen base architecture. Specifically, Dom0 communicates the rules from the configuration files to the Xen hypervisor when the domain is created. The hypervisor then communicates the rules to the NET-VM and FS-VM where they are associated with the proper domain identification. The NET-VM and FS-VM are then trusted to enforce these rules.

The mechanisms for giving control of network interfaces to the NET-VM already exists in Xen in the form of a driver domain, a guest domain that is granted direct access to devices at boot time. The mechanism for giving control of file system access to the FS-VM will need to be implemented. Enforcing the contracts, both from the FS-VM and NET-VM will also need to be added. Finally, support for parsing the newly added syntax in the DomU guest configuration file will need to be added to Xen, this can be modeled from existing mechanisms that parse the current configuration file syntax [32].

The mechanism for giving control of file system access to the FS-VM requires that Xen recognizes and registers the FS-VM with the privileges of being the FS-VM. It then requires that Xen provides the FS-VM with the file system specific rules so that when the DomU reads or writes data, the FS-VM can enforce those rules.

Enforcing the contracts requires the FS-VM and NET-VM to detect a violation of a rule and communicate that violation to Xen so that an appropriate action can then be taken. Possible actions include restoring the DomU to a known good state, simply restarting the DomU, shutting down the DomU, simply notifying the user of the system, or even preparing the DomU for forensic analysis.

6.4 Evaluation Plan

As with the initial prototype, we plan to evaluate in terms of performance and functionality.

In terms of performance, some degradation is acceptable to gain safety, but we want to show that the overall performance is acceptable for typical desktop use. We will follow a similar methodology to testing of the prototype. For example, we will run a variety of workloads that stress various parts of the system including I/O intensive workloads (both read and write), network intensive workloads (both send and receive), and CPU intensive workloads. We will test the performance on the Xen hypervisor running both Windows and Linux guests. We will quantify performance with and without file system and network rules to measure their overhead.

In terms of functionality, we look at several categories including:

(1) **Resistance to attack:** This includes techniques for preventing attacks from occurring at all such as blocking ports, rate-limiting, and forbidding the writing of executable files from some VMAs.

(2) **Recovery from attack:** This includes techniques for quickly detecting and removing the effects of successful attacks including watching for abnormal behavior and freezing misbehaving VMAs, facilitating forensic analysis of an attack, restoring compromised VMAs to a known good state, and rolling back modifications to user data.

To drive our evaluation (both performance and functionality), the construction of example virtual machine appliances is necessary. The careful construction of the virtual machine appliances is important in order to demonstrate that our system can be effectively used to real world applications. We will build on existing appliances where applicable, but will need to construct the virtual machine appliance contracts for each of the appliances that we test.

Since the publication of our original paper, virtual machine appliances have gained significant popularity thanks to companies like VMware[14] and rPath[37], both of which provide downloadable images on the Internet. There are also a growing number of other

sites that provide pre-built images for many different virtualization systems [38][39][40]. The proliferation of different options has created an abundance of choices, but also much confusion from the typical user's standpoint. We will look at the state of virtual appliances and solutions for interoperability and management.

We will carefully choose an application or groups of applications that require the same access to be stored in a virtual machines appliance. Some examples of virtual machine appliances include Internet browser appliances, Email appliances, Office appliances, and Photo editing appliances. Finally, we ensure that all accesses to user data are controlled by virtual machine appliance-specific contracts that define the acceptable access patterns, in terms of file and network access, that the virtual machine appliance needs.

We note that there are more advantages if you split up applications into these virtual machine appliances, but there are still advantages even if all users applications are in one appliance. By decomposing applications into multiple virtual machine appliances, we can tighten security by providing more specific contracts. Taken to the other extreme, one application per virtual machine appliance would provide the most specific contracts, but would cost the most in terms of performance (i.e. each application would require its own underlying operating system, as well as its individual disk, network, and memory allocation). In our evaluation, we will explore this tradeoff.

7 Relationship to Related Projects at Clarkson

There are a few ongoing/proposed projects at Clarkson that are closely related to this one. One project involves using the Log-structured File System (LFS) inside the FS-VM to enable rollback of writes to user data in case of an attack. Another project involves the design and application of more advanced file system rules. Still another project will look into the results of an attack by saving copies of the corrupted VM and developing tools for forensic analysis to be used on the compromised image. The project's goal will be to provide better recommendations for future attack prevention and detection. Finally, there is a proposed project to create tools to help users inspect and assess the risks of contracts, in particular by having a way to visualize what is being granted by the contract, and understand the contract better.

8 References and Further Reading

[1] P. Loscocco and S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. Proceedings of the FREENIX Track, 2001 USENIX Annual Technical p. 29-42, 2001.

[2] Snort, Barnyard, <http://www.snort.org/dl/barnyard>.

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

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

[5] Symantec, Norton Ghost, http://www.powerquest.com/sabu/ghost/ghost_personal.

[6] US-CERT Current Activity, <http://www.us-cert.gov/current>.

- [7] Symantec, Symantec Security Response, <http://securityresponse.symantec.com/>.
- [8] 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
- [9] 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.
- [10] dbench-3.03, <http://samba.org/ftp/tridge/dbench>.
- [11] IOzone 3.2.35, <http://www.IOzone.org>.
- [12] Freebench v1.03, <http://www.freebench.org>.
- [13] 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
- [14] 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.
- [15] R. Goldberg. Survey of Virtual Machine Research. IEEE Computer, p. 34-35, June 1974.
- [16] 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.
- [17] VMware, URL <http://www.VMware.com>.
- [18] Fabrice Bellard, Qemu, <http://fabrice.bellard.free.fr/qemu/>.
- [19] Kevin Lawton, Bochs, <http://bochs.sourceforge.net/>.
- [20] Microsoft, Virtual PC, <http://www.microsoft.com/windows/virtualpc/default.mspcx>.
- [21] NeTraverse, Win4Lin, <http://www.netraverse.com/>.
- [22] J. Dike. A User-mode Port of the Linux Kernel. Proceedings of the 4th Annual Linux Showcase & Conference (ALS 2000), page 63, 2000.
- [23] J. Dike. User-mode Linux. Proceedings of the 5th Annual Linux Showcase & Conference, Oakland CA (ALS 2001). pp 3-14, 2001.
- [24] J. Dike. Making Linux Safe for Virtual Machines. Proceedings of the 2002 Ottawa Linux Symposium (OLS), June 2002.
- [25] Jeff Dike, <http://user-mode-linux.sourceforge.net>.
- [26] Faux Machine, <http://www.jsequeira.com/cgi-bin/virtualization/FauxMachine>.

- [27] T. Bressoud and F. Schneider. Hypervisor-based fault tolerance. *ACM Transactions on Computer Systems*, 14(1):80-107, February 1996.
- [28] 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.
- [29] 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.
- [30] Faronics, Deepfreeze, <http://www.faronics.com/html/deepfreeze.asp>.
- [31] Microsoft, Windows System Restore, <http://support.microsoft.com/default.aspx?scid=kb;%5BLN%5D;267951>.
- [32] Xen Source code that parses Xen DomU configuration file and creates DomU guest <http://lxr.xensource.com/lxr/source/tools/python/xen/xm/create.py>.
- [33] Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor John Scott Robin, U.S. Air Force; Cynthia E. Irvine, Naval Postgraduate School Security Symposium - 2000 <http://www.usenix.org/events/sec2000/robin.html>
- [34] Popek, G. J. and Goldberg, R. P. 1974. Formal requirements for virtualizable third generation architectures. *Commun. ACM* 17, 7 (Jul. 1974), 412-421. DOI= <http://doi.acm.org/10.1145/361011.361073>
- [35] Bugnion, E., Devine, S., and Rosenblum, M. 1997. Disco: running commodity operating systems on scalable multiprocessors. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (Saint Malo, France, October 05 - 08, 1997)*. W. M. Waite, Ed. SOSP '97. ACM Press, New York, NY, 143-156. DOI= <http://doi.acm.org/10.1145/268998.266672>
- [36] J. Matthews, J. Herne, T. Deshane, P. Jablonski, L. Cherian, M McCabe Data Protection and Rapid Recovery From Attack With A Virtual Private File Server and Virtual Machine Appliances *Proceedings of the IASTED International Conference on Communication, Network and Information Security (CNIS 2005)*, p. 170-181, November 2005.
- [37] rPath's rBuilder Online <http://www.rpath.com/rbuilder/>.
- [38] <http://jailtime.org>.
- [39] <http://virtualappliances.net/>.
- [40] <http://www.jumpbox.com/>.
- [42] Zhao, X., Borders, K., and Prakash, A. 2005. Towards Protecting Sensitive Files in a Compromised System. In *Proceedings of the Third IEEE international Security in Storage Workshop (Sisw'05) - Volume 00* (December 13 - 13, 2005). SISW. IEEE Computer Society, Washington, DC, 21-28. DOI= <http://dx.doi.org/10.1109/SISW.2005.17>