Book Chapter to be published in
*ADVANCES IN COMPUTER SCIENCE AND ENGINEERING*

# Virtual Machine Security Systems

Xin Zhao, Kevin Borders, Atul Prakash
Department of EECS, University of Michigan
Ann Arbor, MI, 48109-2121, USA
{zhaoxin,kborders,aprakash}@eecs.umich.edu

**Abstract**

Current operating systems provide the process abstraction to achieve resource sharing and isolation. From a security perspective, however, an attacker who has compromised one process can usually gain control of the entire machine. This makes security systems running on the same computer, such as anti-virus programs or intrusion detection systems, also vulnerable to attack. In response to the imperfect isolation between processes in modern operating systems, security researchers have begun to use virtual machine technology when designing security systems. A virtual machine makes raw device requests to a set of devices that are emulated by underlying software. So, software running in a VM has the appearance of its own dedicated hardware, but is actually controlled and managed by a software layer on the physical computer. With reasonable assumptions, the level of isolation between virtual machines is nearly equivalent to having two separate machines. At the same time, the underlying software has full access to each virtual machine's state. These properties make virtual machines very attractive for designers of security systems. In this chapter we explore a wide variety of security applications that utilize virtual machine technology, including intrusion detection and introspection, honeyfarms, logging and replaying, secure file systems, and even malicious software.

# 1 Overview of Virtual Machine Technology

Virtual machines have been in existence since the mid 1970's [Cre81, Gum83]. A virtual machine (VM) is a logical process (most often an operating system) that interfaces with emulated hardware and is managed by an underlying control program. Originally, virtual machines were run on mainframes to provide resource multiplexing and isolation [Cre81, Gum83]. Recently, virtual machines have been growing in popularity. Some virtual machine systems, such as VMWare [SVL01] and Xen [BDF+03], have seen widespread deployment at many large corporations [Asl].

Most modern virtual machine systems use the virtual machine monitor (VMM) model for managing and controlling individual virtual machines. The VMM is a thin software layer that runs directly on a physical machine's hardware. On top of the virtual machine monitor, there can be one or more virtual machines. The VMM provides each virtual machine with a set of virtual interfaces that resemble direct interfaces to the underlying hardware. Applications on a virtual machine can run without modification as if they were on running on a dedicated physical machine. The VMM allows multiple virtual machines to be running at the same time and transparently multiplexes resources between them [Gol74]. The VMM also isolates the virtual machines from one another, preventing them from accessing each other's memory or disk space. The operating system that runs inside of a virtual machine is traditionally referred to as the *guest OS*, and applications running on the guest OS are referred to as *guest applications*.

# 2 Virtual Machine Types

All computer programs consist of instructions. When a program runs inside of a virtual machine, these instructions either execute directly on the processor, or are interpreted by the underlying software. The proportion of instructions that run directly on the processor will significantly impact a virtual machine's performance. In one extreme, a real machine only uses direct execution. Programs will run very fast on a real machine because their instructions do not require translation. Direct execution delivers the best performance, but has no support for checking machine state prior to executing privileged instructions. The other extreme is a complete software interpreter machine (CSIM). A CSIM translates and emulates every single processor instruction for a computer program. This requires a lot of extra work and is usually very slow. A virtual machine monitor uses a combination of these two methods. It executes a "statistically dominant subset" of program instructions (including all the basic arithmetic, memory, and branching operations) directly on the processor, while emulating privileged instructions such as system calls and device I/O requests [Gol72].

Virtual machine monitors can be categorized into two groups: Type I VMMs or Type II VMMs

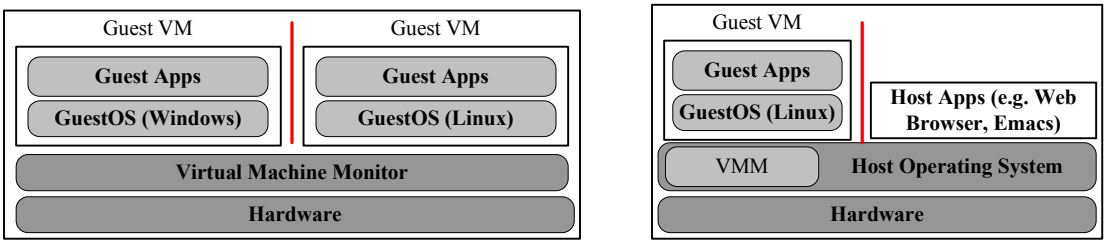


Figure 1: Architecture of Type I and II Virtual Machine Systems

[Gol72]. Figure 1 shows the architecture of each VMM type. A Type I VMM runs directly on the physical hardware. It does not have an operating system running below it; the Type I VMM is fully responsible for scheduling and allocating of the system's resources between virtual machines. Examples of Type-I VMM include VMWare ESX (enterprise) [Wal02], and Xen [BDF+03]. A Type II VMM runs as an application in a normal operating system. This operating system controls the real hardware resources, and is typically referred to as the "*Host OS*." The host OS has no knowledge of the Type II VMM, which is treated like any other process in the system. The operating system that runs inside of the Type II VMM is referred to as the "*Guest OS*." Examples of Type-II VMM include VMWare GSX (workstation) [SVL01], UML (User-Mode Linux) [Dik00], and FAUmachine [HWS04]. Because the Type II VMM is running inside of a standard operating system, any security vulnerabilities that lead to the compromise of the host OS will also give full control of the guest OS. Host operating systems for Type II VMMs are more heavyweight than Type I VMMs, and more prone to security vulnerabilities. For this reason, Type I VMMs are generally considered to be much more secure than Type II VMMs.

# 3 Overview of VM based Security Systems

## 3.1 Architecture of VM based Security Services

While VM-based security systems have different features, they usually share a similar architecture. As shown in Figure 2, security services can be deployed either as part of virtual machine monitor or in a dedicated virtual machine. Some security services may also run components in guest operating systems. However, components in guest OSes are often only responsible for making requests to security services on the VMM or a dedicated secure VM where policy enforcement takes place. Security policies are rarely enforced from within the guest OS because it is much more likely to be compromised.

The security of VM-based services rests on the assumption that the underlying trusted computing base (TCB) is also secure. If the TCB compromised, then all bets are off for the VM-based

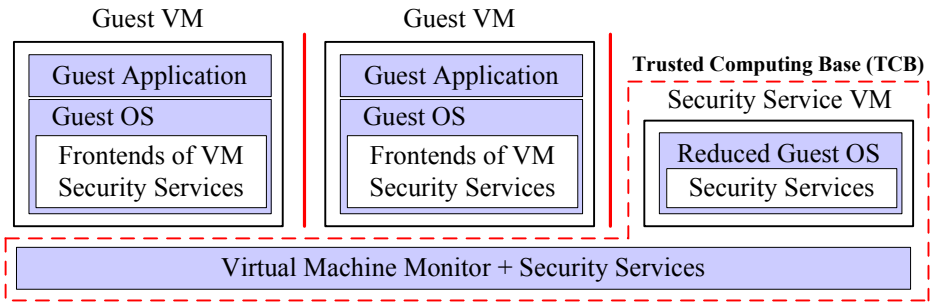


Figure 2: Architecture of VM-based Security Services

security service. In a Type I virtual machine, the trusted computing base is the virtual machine monitor. Some services also need to include the dedicated secure VM as part of TCB. The TCB is considered to be secure because "It is so simple that its implementation can be reasonably expected to be correct" [GR03]. Virtual machine monitors are only responsible for virtualizing the physical machine's hardware and partitioning it into logically separate virtual machines. Compared to a full operating system, which may have several million lines of code, the Disco [BDGR97, GTHR00] and Denali [WSG02] VMMs have around 30,000 lines of code [GR03], and Xen has approximately 60,000 lines of code. Also, the secure VM typically has a reduced mini-OS without any unneeded services or components.

In addition to having a small code base, the interfaces to VMM and the dedicated security VM are much simpler, more constrained, and better specified than a standard operating system. This helps reduce the risk of security vulnerabilities. For example, the Xen virtual machine monitor can only be accessed through 28 predefined hypervisor calls (privileged functions used by guest OSes to the VMM). In contrast, a standard Linux operating system exposes upwards of a few hundred system calls (Linux kernel 2.6.11 has 289 system calls), special devices such as `/dev/kmem`, kernel modules, and a variety of privileged programs, such as sendmail and sshd, some of which are packaged with the OS and others are provided by third parties.

## 3.2 Advantages

Virtual machines have several advantages that make them better suited for providing security services than a standard physical machine. First, guest virtual machines run on emulated hardware, which means that the virtual machine monitor has access to all of a guest virtual machine's state. This includes registers, memory, disk contents, and other I/O device state (such as the video or sound buffer). By dealing with the VMM, security services can monitor all activities on a guest VM from outside.

Virtual machine technology also provides stronger isolation between virtual machines than is available between processes in conventional multiprogramming environments [MD73]. This

makes it possible to run security services on a VM that is isolated from attack by other compromised virtual machines. In contrast, any security services running on a normal machine can be tampered with or disabled once it has been compromised by an attacker.

Another advantage is that it is much easier to manipulate the state of a virtual machine than the state of a physical machine. The state of the virtual machine can be saved, cloned, encrypted, moved, and restored, none of which is easy to do with physical machines [CN01]. Saved VM states, or "snapshots," have been used to replay past activities on a guest OS for the purpose of studying intrusion techniques and collecting forensic evidence. Also, snapshots provide a convenient way of rolling back to a previous state after an attack has taken place and corrupted the system.

Two virtual machines can also communicate faster than two separate physical computer systems. Physical machines are separated by physical networks, which are slower than a memory bus. Fast inter-VM communication makes it possible for two machines to cooperate with each other for security purposes without impacting performance. For example, SVFS [ZBP05b] moves the guest virtual machine's file system to a dedicated data VM so that access control policies can be enforced outside of the guest VM. The data VM serves file access requests through fast inter-VM communication. If SVFS were to use a traditional networked file system, I/O performance would degrade severely, which would not affect security, but would make the system much less usable.

Finally, virtual machine technology makes it possible to recognize an malicious application by testing it on a "cloned" machine and see whether this application behave normally [CCC+05, CN01]. In contrast, it is hard to conduct this test on a real system because running suspicious applications risks compromising the system. Virtual machine make it easy to clone a running system. The cloned virtual machine can then be used to examine the effects of running suspicious applications rather than looking only at the applications. Because clones are isolated from the real system, all operations done in the clones will be contained and will not affect the real system, making the application testing safer.

## 3.3 Challenges

While virtual machine technology provides security benefits, it also presents some unique challenges. The first is performance. Running security services hurts system performance due to virtualization overhead and inter-VM communication. For example, inspecting a virtual machine's system calls requires that the VMM trap every system call, check its parameters, and then forward it to the guest operating system. Performing all these operations adds a significant amount of overhead. Furthermore, the VMM needs to trap all access requests to physical devices from the guest virtual machines. The VMM must then translate and reissue each device I/O command to the

physical hardware, rather than allowing the guest OS to directly manipulate those devices. In some virtual machine systems, like Xen, device access requests and results are exchanged via cross-VM communication, which requires extra context switching and further increases the system overhead. Virtualizing an x86 series processor also incurs an additional performance penalty because x86 processors do not trap all instructions that need to be virtualized [RI00]. Instead, the VMM must modify the privileged instructions to emulate correct behavior. Overall, virtual machines do not run as efficiently as normal computers, which can make them an unattractive option for real-world deployment, even if they do provide a higher level of security.

The second challenge is hardware cost. Currently, virtual machines require a good deal of physical resources. Some disk and memory space must be pre-allocated for each virtual machine. Users often face a configuration dilemma when setting up virtual machines. If a lot of resources are assigned to each VM, then they will have better performance, but fewer will be able to run on the same hardware. On the other hand, if each VM runs with a minimal amount of resources, then a lot of them will be able to run on one physical machine, but application performance will suffer. This conflict is caused by a combination of isolation and static resource allocation: virtual machines cannot share or swap resources, even if one VM is under-utilized. This can cause a problem for VM-based security services because they will take up a fixed amount of system resources, even if they are idle most of the time. If a user has an older machine or limited memory then it may not be feasible to run a virtual machine system.

The third challenge of virtual machine security services is the semantic gap between the guest operating system and the underlying virtual machine monitor [CN01]. The VMM can only see the raw state of the guest virtual machine (i.e. disk blocks rather than files [SVL01]), while security services usually need to reason about guest VM state on a higher level. Without knowledge of on-disk structures, the security service running on the VMM will be unable to view information at the appropriate level of abstraction. What further complicates this problem is the ability of a hacker to corrupt low-level structures residing in the guest VM's memory and on disk in an attempt to confuse and disrupt security services that try to view the data at a high level.

Another challenge for VM-based security services is that malicious software can fingerprint virtual machines and try to avoid them entirely. Several techniques have been published for fingerprinting a virtual machine [Cor03, Cor04, RK03], some of which are already employed by the Agobot malware program [F-S]. Although beneficial in many circumstances, having attackers avoid machines with VM security services on them may lead to focused attacks on more vulnerable systems. VM fingerprinting can also cause a problem for virtual machine honeypots, which are designed to lure attackers in order to study their techniques. If attackers are able to avoid honeypots, then the honeypots will be ineffective. Overall, this is a challenge that is likely to go away in the future as more computers run virtual machines, and chip manufacturers like Intel and AMD

add support for full virtualization to their microprocessors [Int, AMD].

Finally, virtual machines that run security services could potentially hurt system security themselves. For example, the VM-based logging tool Revirt [DKC+02] can log system activities and rollback or replay them at a later time. It records the previous states of the guest VM on disk. If this log is not carefully protected, an adversary can recover sensitive data (e.g. cryptographic keys, medical documents) from the log, even if the data has already been removed from the system [GPCR04, GR05]. Furthermore, these logs could be subpoenaed by a court of law at a later time. This is a serious concern for many companies, some of whom even go so far as to destroy old logs of e-mail between employees.

# 4 VM based Security Projects

During the past several years, researchers have developed a number of security systems that rely on the properties of virtual machines to provide a higher level of security than would be possible on a normal physical machine. In this section, we describe some recent advances in virtual machine-based security systems.

## 4.1 VM-Based Intrusion Detection Systems

Three capabilities of virtual machines make them particularly attractive for building an intrusion detection system. The first capability is *isolation*. Software running in a virtual machine cannot access or modify the software running in the VMM or in a different VM. Isolation ensures that even if an intruder has completely subverted a guest virtual machine, he or she still cannot tamper with the IDS.

The second capability is *inspection*. In a virtual machine system, guest VMs run on emulated hardware and the virtual machine monitor has access to the entire state of each guest VM. Being able to directly inspect the virtual machine makes it particularly difficult to evade a VM-based IDS because there is no state in the VM that the IDS cannot see.

The third capability is *interposition*. The presence of privileged instructions forces the VMM to trap and emulate these instructions, which incur extra overhead that would not exist in a conventional system. However, these privileged instructions also provide hooks to allow a VM-based IDS to record or modify privileged instruction parameters and other virtual machine state [RG05, WCSG05, Whi].

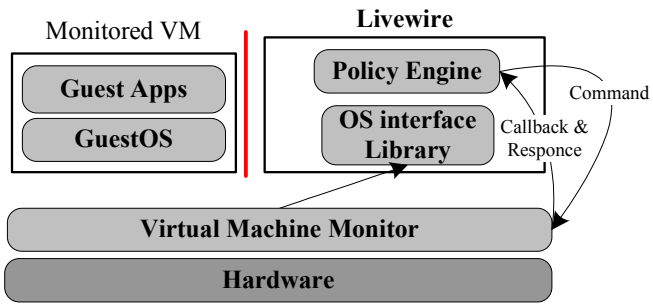


Figure 3: Architecture of Livewire

### 4.1.1 Livewire

Livewire [GR03] takes advantage of virtual machine technology to enforce security policies on guest virtual machines using the VMM model. As shown in Figure 3, Livewire consists of two major components: the *OS Interface Library*, and the *Policy Engine*.

The *OS Interface Library* provides an OS-level view of the target virtual machine by interpreting the hardware state on the VMM. This component is important because VMMs manage state strictly at the hardware level (i.e. the VMM can see disk blocks, but not files). An IDS, however, prefers to analyze actions and events using OS-level semantics such as files, sockets, and processes. The implementation of the OS Interface Library requires knowledge of the guest OS in order to properly interpret the VM's machine state. Different guest OSes will need different OS Interface Libraries. For example, the VFS [BC00] super block structure in the Linux 2.4 kernel is different from that of the Linux 2.6 kernel. The OS Interface Library must match the guest OS semantics in order for the state interpretation to be correct. The diversity of OS kernels makes developing a common OS Interface Library a formidable challenge. This is one major limitation of the Livewire intrusion detection system.

The *policy engine* is the heart of Livewire. This component obtains events from the VMM interface and the OS Interface Library, and decides whether or not the system has been compromised. If Livewire believes that the system has been compromised, the policy engine is also responsible for taking appropriate action, such as notifying an administrator or blocking the machine from accessing the disk or network. Livewire can run multiple policy modules that implement specific heuristics in order to detect a wide variety of intrusions.

### 4.1.2 Siren

Siren [BZP05] is another VM-based intrusion detection system. Siren is designed to detect malicious software operating within a guest virtual machine that attempts to send out information over the network. Siren operates on the principle that workstation machines rarely need to communicate

over the network in the absence of human input, with the exception of traffic from a few applications like automated update programs. Siren takes advantage of both isolation and interposition to correlate human input from the mouse and keyboard with resulting network traffic. If it sees traffic when no human input has occurred for over five minutes, it flags the traffic as potentially malicious (unless the traffic is deemed to be safe according a white list of trusted programs such as Windows Update).

A simple counter-measure to basic human input correlation would be for malware to wait for human input before sending out data over the network. In fact, many spyware programs operate as browser add-ons and only communicate with their home servers while the user is actively browsing the web. To deal with this type of malware, Siren also injects streams of specially crafted human input that are designed to mimic normal user activity. Siren determines ahead of time the exact sequence of network messages it expects to see as a result of the crafted human input. Finally, it compares the actual network traffic to the expected network traffic, flagging unexpected requests that are likely to have come from malware programs.

A major factor in the success of Siren's input injection technique is how closely it resembles normal user activity. A very clever malicious program that knows about Siren could use complex natural language and behavioral analysis techniques to identify real human input, and thus determine when it is safe to send out network traffic. Therefore if such malware is part of the threat model, it is critical to ensure that the input generation algorithm utilizes equally complex techniques to create input that is indistinguishable from real user activity to another program. This leads to an "arms race" situation, as commonly occurs with many intrusion detection techniques. Siren puts no theoretical restriction on whether a malware program will be able to escape detection. It does, however place a practical requirement on the amount of sophistication required to avoid detection. Automatically scrutinizing all input to identify operations that must have been executed by the user is no small task, and significantly raises the bar for today's malware.

To evaluate Siren, the authors created a sequence of web requests and manually injected them into a guest virtual machine. This was done on a clean Windows XP installation to determine the set of expected network traffic, and then repeated on machines with various types of spyware installed on them. For the sake of comparison, the authors also tested the spyware programs against a traditional behavioral analysis tool called Web Tap [BP04] that looks solely at network requests to pick out spyware. Without injecting human input, Siren was able to detect three of the ten spyware programs tested. These three programs were also detected by Web Tap. The other seven programs, however, required injection of human input to identify because they only sent traffic while the user was actively browsing. Immediately after injecting the crafted human input, however, all of the remaining spyware programs made requests to their home servers. These requests did not show up in the original network trace from the clean machine and were correctly flagged by Siren as

coming from malicious software.

Siren's approach to intrusion detection takes full advantage of the virtual machine monitor's interposition capability. It injects human input into the guest virtual machine in an attempt to trick malware into sending data out over the network. Siren's use of deception for intrusion detection is quite unique, and it could not be done without the help of virtual machine technology. The isolation between the VMM and the guest VM ensures that a compromised guest will be unable to recognize that Siren is running and also unable to gain any information about Siren's configuration to identify the false human input. The Siren technique also raises an interesting question: will it be more difficult in the future to generate realistic human input or to write a program that is able to identify real human input? The authors hope that the latter is the case and that the use of deception will help to identify previously hard-to-detect malicious software.

## 4.2 VM-Based Intrusion Prevention Systems

It is not only helpful to detect attacks as they happen, but also to harden systems ahead of time so that they are more resilient to intrusions. Intrusion prevention systems do just this. With the aid of virtual machine technology, the systems described in this section enforce policies that help protect critical resources so that they are safe from potential attackers.

### 4.2.1 SVFS

SVFS is a secure virtual file system that is designed to protect sensitive files, even in the event of a compromised operating system [ZBP05b]. When a hacker successfully gains control of a machine, he or she often tries to steal confidential information (e.g. documents, passwords), taint system logs to avoid detection, and modify critical system files to create a backdoor and retain persistent access to the machine. Although traditional file protection mechanisms attempt to block malicious file access, they can usually be bypassed or disabled once a system has been compromised [ZQ, A N, Bea03, bre]. In contrast, SVFS uses virtual machine technology to moderate access at the VMM layer, better protecting sensitive files from malicious access by a compromised OS.

In the SVFS architecture, which can be seen in Figure 4, multiple virtual machines run on a single physical machine. One of these virtual machines, which is referred to as the *Data Virtual Machine (DVM)*, mediates file access requests from all of the other VMs. The user's standard applications reside on the other guest virtual machine(s). All access to sensitive files by these applications must first be approved by the DVM. The DVM moderates legitimate access transparently, only returning access denied if the virtual machine does not have permission to access a file.

The SVFS file system is composed of two parts: the *SVFS server*, which runs in the DVM, and

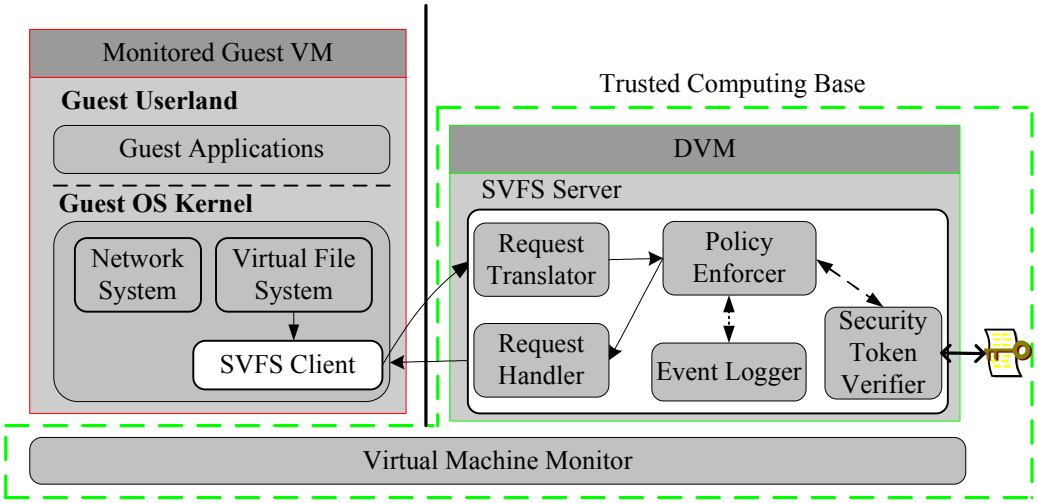


Figure 4: Architecture of SVFS

*SVFS clients*, which run in each guest VM. The SVFS clients operate inside of the kernel of each guest virtual machine. Each client registers with the SVFS server during boot time and forwards access requests to the SVFS server by hooking calls at the virtual file system layer [BC00]. Guest application can then access sensitive files using standard system calls, such as `open`, `read` and `write`. When the system calls get passed down to the kernel, the SVFS client determines if the file resides within SVFS and forwards the request to the SVFS server if necessary. Requests from a SVFS client to the SVFS server are made using virtual remote procedure calls (VRPCs). VRPCs are much faster than normal RPCs because they do not go through a network stack. When the client receives the result, it returns the data transparently to the application that made the system call, giving the illusion that the request was processed locally.

The SVFS server runs inside of the data virtual machine and uses the Ext3 filesystem [TT02] to store files for the guest virtual machines. It is responsible for serving access requests and checking security policies. Unlike traditional user-based access control systems, the SVFS security policies specify which guest virtual machines are allowed to access which files. SVFS policies allow an administrator to make critical operating system files read-only. Guest operating systems will still be able to boot, but hackers who compromise a guest OS will not be able to install a persistent root kit by modifying system files. It is still possible to update the OS by running a trusted administrative guest VM. The idea behind an administrative VM is that it will only be used to modify critical system files, and not to perform activities that may lead to a security breach such as reading e-mail, browsing the web, and downloading software.

Because sensitive files reside in DVM, guest virtual machines that wish to access them must use inter-VM communication. Because file system performance is critical for many applications, inter-VM communication must be fast. Although TCP connections are an option for inter-VM

communication, the additional overhead of going through two network stacks, one to send and one to receive, makes them very inefficient. To speed up inter-VM communication, SVFS introduces the *virtual remote procedure call(VRPC)*. A VRPC has the same format as a normal remote procedure call. However, it uses memory sharing to achieve fast data exchange between virtual machines running on the same hardware, completely bypassing the network stack.

VRPCs take advantage of Xen's memory re-mapping support to improve data exchange performance. In Xen, a virtual machine can allocate a memory region and report the starting address and size of this region to another virtual machine. The other machine can then map that memory region into its own address space and access data in that region directly. VRPC uses this mechanism to establish a shared memory region between the DVM and each guest VM. The two virtual machines can then exchange data using shared memory. This avoids the overhead of network connections, and eliminates the need to copy memory because each VM can access it directly. In contrast, a network-based RPC requires at least two memory copies: the sender copying data from the application to the network stack, and the receiver copying data from network stack back to the application. In addition, network RPCs require translation of memory to network byte order. VRPCs do not need to worry about byte ordering because both virtual machines are running on the same physical host, which further reduces overhead.

In addition to securing individual machines, the secure virtual file system plays a key role in the secure grid computing environment SVGrid [ZBP05a]. SVGrid allows untrusted grid computing tasks to share the base software environment, which includes the operating system and common programs. File access requests made by the untrusted tasks are mediated by the SVGrid policy enforcer, which resides in the data virtual machine and enforces file access policies. If a request is made to modify a shared file, then SVGrid will make a private copy of that file for the guest virtual machine. This enhances SVGrid's flexibility when running untrusted programs, while still ensuring that the programs are unable to disrupt other activities on the shared physical computer.

### 4.2.2 NetTop

Many organizations have information with varying levels of sensitivity. In the government, sensitivity is denoted using a classification system. In order to keep classified digital information from falling into the wrong hands, government agencies will have different networks for different classification levels. However, an employee may need to access both classified information and browse the internet throughout the day. To deal with this dilemma, a user will typically have a separate physical computer for each classification level.

While having one computer for each level of sensitivity solves the problem, it is very inefficient. First, it clutters workspaces with computer equipment. These extra computers can also be quite expensive. Another problem with having multiple isolated networks is that sometimes workers

need to transfer data between them. Although it would be bad to allow data to flow from a high-sensitivity network to a low-sensitivity network, users often need to transfer information, such as program updates, from the internet to a classified network.

It would be best if each user had one computer that could handle all classifications of data without compromising security. To achieve this goal, the US National Security Administration developed the NetTop architecture [MS00]. NetTop uses VMware's virtual machine monitor to run multiple isolated VMs, each of which has access to data with a different level security classification. The isolation property of virtual machines ensures that no data flows from a classified to an unclassified VM. NetTop also runs two dedicated virtual machines, one to perform encryption using IPSec, and one filtering router machine. These two VMs enforce strict security policies that prevent network traffic from flowing between networks of different classifications. The NetTop architecture enables users to run tasks of different security levels simultaneously on a single desktop machine using different virtual machines. This significantly reduces overhead in terms of number of physical computers, and also increases productivity. An important question raised by the NetTop architecture is how safe is the VMM from attack by a virtual machine? Co-locating a classified and unclassified VM on the same hardware is risky, especially due to the very high value of classified information and incentive for an attacker to steal it. In order for NetTop to see wide-scale adoption by conservative government organizations, security experts will have to guarantee that the level of isolation between virtual machines is the same as isolation between separate physical machines.

### 4.2.3 IntroVirt

The most effective way of eliminating security vulnerabilities is to patch vulnerable software. However, administrators are often hesitant to apply patches right after they are released because a vendor's rush to release a patch often precludes thorough testing and leads to buggy patches. Instead, administrators may wish to install a patch after it has proved to be stable. However, this delay in applying patches leaves vulnerable applications open to attack.

IntroVirt [JKDC05] provides a way to delaying security patches while still preventing exploitation of recently released vulnerabilities. It uses virtual-machine introspection to monitor application and operating system execution in a guest virtual machine. IntroVirt also allows administrators to define predicates for known vulnerabilities. IntroVirt can then run these predicates on previously recorded execution paths using a combination of virtual-machine introspection replay technology. Running predicates on past execution will help determine whether the system has been compromised by exploitation of the newly discovered vulnerability. This is all done using virtual machine snapshots so as to not disturb the current state of the target system. In addition to replaying old execution, IntroVirt monitors current system execution to look for activity matching the vulnerability predicate. If it is discovers an attempt to exploit the vulnerability, it can respond by halting

execution, notifying a system administrator, or blocking malicious activity.

One limitation of IntroVirt is that the vulnerability predicates need to be written by hand. The amount of time and effort required to write a vulnerability predicate is not entirely clear. Furthermore, writing a correct vulnerability predicate may require access to the program source code, requiring vendor cooperation for closed-source software. In spite of this fact, IntroVirt is a promising and useful system, especially for assessing whether a vulnerability has been exploited in the past before it was publicly known.

### 4.2.4 sHype

The security systems we have discussed so far have only dealt with losing sensitive data or having important files corrupted by an attacker. They have not, however, addressed the problem of a denial of service attack. In a traditional virtual machine architecture, the VMM is only responsible for binding resources to each guest VM, not regulating their use of resources that have been allocated to them. This reduces overhead, but does so at the cost of granularity of control. One could imagine a misbehaving guest virtual machine saturating the disk, network, and memory bandwidth or utilizing 100% of the CPU in order to disrupt other virtual machines. SHype [SVJ+05], developed at IBM, attempts to deal with this problem by giving an administrator fine-grained control over system resource allocation. SHype mediates access to hardware resources at a low level (i.e. disk block writes instead of file writes), eliminating the need to have multiple implementations for different operating systems. The downside of the SHype solution is that it cannot do anything to prevent resource starvation within a virtual machine. SHype cannot stop an intruder from modifying sensitive files or saturating the CPU on a compromised VM because its policies are only enforced at the virtual machine level.

## 4.3 VM-Based Honeypots

Honeypots have recently become a popular tool for identifying wide-scale attacks on the internet. According to Lance Spitzner, the founder of the Honeynet project, "*A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource.* [Spi03]" To put it more concretely, a honeypot is a computer system that is set up with the sole intention of luring attackers. Security experts analyze activity on honeypot machines to get a better idea of threats that face their networks and the internet as a whole.

In general, honeypots fall into two categories: *low-interaction* and *high-interaction*. A low-interaction honeypot will accept network messages, but only give a minimal response, if any. For example, a honeypot that accepts TCP connections on port 80 (the standard port for HTTP requests), receives all data sent by the client, but does not respond would be a low-interaction

honeypot. A high-interaction honeypot, on the other hand, behaves more like a normal computer. In the previous example, a high-interaction honeypot would service the HTTP request just like a normal web server. In general, high-interaction honeypots provide much better information about attacks, but require a lot more resources to run. Some security researchers have opted to use low-interaction honeypots in order to handle traffic to large IP address blocks because running high-interaction honeypots would be too costly.

With the growing popularity of honeypots, virtual machines have begun to play an important role. Because honeypots do not serve any legitimate clients, they only require a small amount of system resources. Virtual machines provide resource multiplexing, which allows more high-interaction honeypots to run on the same hardware. Virtual machine technology makes it feasible to deploy more high-interaction honeypots on the same hardware. Furthermore, virtual machine technology allows more in-depth monitoring of malicious activities on honeypot machines without attackers being able to detect or disable monitoring software.

Using virtual machine honeypots, however, also has a disadvantage. As of the writing of this book, full hardware virtualization is not supported by Intel or AMD processors. A program can determine if it is running inside of a virtual machine by executing the x86 SIDT instruction at an unprivileged level. The *redpill* [Rut] program does just that, examining the result to determine if it is running on a virtual machine. Hackers can use *redpill* to avoid honeypots that run on virtual machines. Mitigating this attack on current processors would require binary translation or instruction-level emulation, which would have a negative impact on performance. In the future, both Intel and AMD plan to release processors that support full virtualization, making it easier for honeypots to avoid fingerprinting by attackers.

### 4.3.1 The Potemkin Virtual Honeyfarm

Standard virtual machines support multiplexing of processor and device resources, but require dedicated physical memory and disk space. Because the utility of honeypots is based entirely on their external appearance, they will rarely need the disk space and memory allocated to them. The Potemkin Virtual Honeyfarm [VMC+05] takes advantage of this fact to allow hundreds of high-interaction honeypots to run on a single computer. It utilizes copy-on-write to eliminate much of the redundancy in memory and on disk between honeypots. Copy-on-write also enables the Potemkin to rapidly create new virtual machines when it receives network traffic to a particular IP address. Waiting for network traffic to create a honeypot enables Potemkin to cover a much larger address space more efficiently because individual IP addresses do not receive traffic very often.

One challenge in implementing any high-interaction honeypot system is containing malware once it has infected the honeypots. For legal reasons, anyone deploying honeypots is responsible for ensuring that compromised hosts aren't able to launch attacks on other computers on the inter-

net. Potemkin leverages its ability to rapidly create new honeypots to provide a unique solution to this problem. It uses reflection to prevent infected hosts from attacking the outside network. When a compromised VM attempts to contact an outside IP address, a gateway reflects the network traffic to a new virtual honeypot that emulates the outside host. This not only protects Potemkin from legal liability, but also allows it to better monitor malware propagation.

The creators of the Potemkin Virtual Honeyfarm ran a number of experiments in a test bed environment to determine Potemkin's scalability. The test bed consisted of servers with 2.8 GHz Xeon processors, 2 GB of physical memory, and 1Gbps network connections. The servers received traffic to an entire /16 address block ($2^{16}$ or 65,536 IP addresses). In the experiments, the creators of Potemkin also added a "scan filter" to automatically reply to simple port scans without creating new virtual machines. With the scan filter, and with swapping out virtual machines that were inactive for greater than five minutes, the peak number of simultaneous machines required to handle traffic to the /16 address block was 1,745. Separate experiments to determine the maximum number of VMs on one physical computer showed that it could support 116, but improvements to the virtual machine monitor could increase that number to approximately 1,500.

Prior to the Potemkin Virtual Honeyfarm, scalability of high-interaction virtual machine honeypots was severely limited. With standard VM technology, a cluster of high-interaction honeypots could only cover a few hundred IP addresses. Potemkin optimizes memory sharing between virtual honeypots to make high-interaction honeyfarms covering an entire /16 address space a reality. This technology will aid future security professionals in getting a better idea of how hackers, worms, and other malware operate at an internet-wide level.

### 4.3.2 The Collapsar Honeypot Center

One major challenge of deploying honeypots across a wide variety of networks is management overhead. Configuring and maintaining honeypots can be a lot of work, and also requires a good deal of expertise. Smaller enterprises may wish to deploy honeypots in their network, but do not have the resources to administer them. The Collapsar center [JX04] is a cluster of high-interaction honeypots that receive traffic from redirection devices spread out across other networks. A redirector acts as a local host in a production network by tunneling all LAN traffic back to a Collapsar honeypot. Once collected in a central location, it is much easier to analyze honeypot data from multiple source networks.

The Collapsar honeypots take advantage of the isolation provided by virtual machine technology to get a better picture of what hackers are doing without them being able to detect or disrupt the monitoring activity. Each honeypot is instrumented with secure logging and OS monitoring capabilities. Operating system information can be much more useful than network traces alone. An example of this is obtaining clear-text for encrypted communications by hooking system calls.

It would be very difficult to read encrypted traffic captured at the network level because the keys are kept private.

One disadvantage of the Collapsar system is it adds a new method for fingerprinting honeypots. One could imagine a network with a redirector that sends traffic to a Collapsar center far away. It would be trivially easy for another host in the local network to fingerprint all of the honeypot redirectors by simply running a ping sweep and comparing round-trip-times (RTTs). Computers on the same sub-network will often reply in less than a millisecond, while the RTT to a remote honeypot center may be anywhere from 20 to 500 milliseconds depending on how far away it is. The first way of countering this attack is to have the redirector respond immediately to pings. However, if an attacker performs a scan with more complicated application-layer requests, then the redirectors will have to forward the packets to Collapsar. Another counter-measure is modifying legitimate machines to delay their responses. There is a reasonable chance, however, that slowing down legitimate machines will have a significant impact on quality of service in the network. Furthermore, hackers who are clever enough to run a timing attack in the first place may avoid the network altogether after seeing unusually large response times.

The primary contribution of the work on Collapsar is introduction of a new management structure for honeypot deployment. Most of the VM-based monitoring technology used by Collapsar comes from other work on virtual machine intrusion detection systems, particularly that of Garfinkel and Rosenblum. The new method of redirecting traffic to a central location helps facilitate dynamic deployment of honeypots over a wide variety of networks. Unlike the Potemkin virtual honeyfarm, which is targeted more towards covering huge contiguous IP addess blocks, Collapsar does a better job of filling small holes in production networks to detect more focused attacks (i.e. individual hackers instead of worms). Because the two technologies are not mutually exclusive, however, a Collapsar-type redirection infrastructure with central honeypots running Potemkin may prove to be a very effective tool for detecting a wide variety of attacks across a large but diverse address space.

## 4.4 Terra: A VM-Based Trusted Computing Platform

Virtual machines monitors are particularly well suited for building trusted computing platforms. This is because VMMs are very light-weight and do not need to change often. The Terra system [GPC+03] uses a *trusted virtual machine monitor* (TVMM) to partition resources between isolated virtual machines (VM), thus providing the appearance of a dedicated physical machine for each VM. The TVMM can expose either an ”open box” hardware interface, or a ”closed box” interface. ”Open box” hardware is a general-purpose platform like a personal computer. ”Closed box” hardware protects the privacy and integrity of some of its contents, more like game consoles and

cellular phones.

The TVMM is capable of verifying and authenticating software running inside of the virtual machines to remote party over a network connection using a process called *attestation*. Attestation requires building a chain of cryptographic certificates. The trusted hardware contains an asymmetric private key that is signed by the vendor. The tamper-resistant hardware certifies the integrity of the system firmware (e.g. PC BIOS). The firmware, in turn, verifies the boot loader, which verifies the TVMM. Finally, the TVMM certifies that each of the virtual machines it loads have not been tampered with. This process of attestation allows an application running in a closed box environment to cryptographically identify itself to a remote party. This allows the remote party to trust the application to behave as expected. Attestation is particularly useful for digital rights management and copyright enforcement; manufacturers may not want give service to modified or pirated software.

## 4.5 ReVirt: A VM-Based Logging and Replaying System

Many administrators rely on system logs to analyze and understand security breaches, discover the vulnerability that led to the initial compromise, and to repair any damage that may have been caused by the intruder. Traditional logging systems, however, have a few shortcomings that prevent them from fully achieving this goal: lack of *integrity* and of *completeness*. These systems rely on the integrity of the operating system to protect them from attack. If a hacker is able to break in to the operating system, however, then he or she can tamper with or disable all logging capabilities, and even corrupt old logs [tam01]. Traditional logging systems also lack completeness. They usually do not record enough data to fully understand or recreate attacks. Typical loggers only save a few types of system events, which may not be sufficient to determine how the break-in occurred or the extent of the damage caused by the break-in. This often leaves administrators with an assessment of data loss, and no way of fixing the machine without completely reinstalling the operating system and all of the applications.

ReVirt is a VM-based logging and replaying system that attempts to address the lack of integrity and completeness provided by traditional loggers. ReVirt performs system logging at the VMM layer, thus eliminating the need to trust the operating system. The guest virtual machines are unable to access or tamper with ReVirt. ReVirt also records all of the operations necessary to recreate an entire attack, even if the intruder goes so far as to replace the compromised machine's operating system. Logging is done at an instruction-by-instruction level, giving administrators a complete picture of everything that has happened on the guest virtual machine, even in the presence of non-deterministic attacks and execution.

While ReVirt is a powerful logging tool, it may leak sensitive information if the logs are not

well protected. One fundamental principle of building secure systems is minimizing the amount of time that sensitive data remains in a system [GPCR04]. Many cautious applications destroy sensitive data (e.g. password, cryptographic keys) right after it is used. With ReVirt, however, this data is never really ”dead” because ReVirt records everything that happens in the system. If an attacker manages to get a hold of ReVirt logs, then he or she will have access to all of this sensitive information. Although it is very difficult to remotely compromise the ReVirt logging virtual machine, an insider could steal a hard drive or attempt to view the log by physically opening it on a machine. One way of partially mitigating this threat is to encrypt ReVirt's logs.

## 4.6 SubVirt: VM-Based Malicious Software

Virtual machines have a number of excellent security properties that help protection systems to run on a machine without being exposed to attack by malware running on the guest operating system. The isolation provided by virtual machines is great for security systems, but it can also be used by malicious software to avoid detection and prevent removal. This idea was first proposed by King et al. in research on SubVirt [KC06]. The authors outline a method for taking the operating system running on the target machine and hoisting it into a guest virtual machine. The malware can then run below it at the virtual machine monitor level. Once encapsulated in a virtual machine, the only way for the guest operating system to detect the malware, other than noticing the very small performance overhead, is to use a technique such as *redpill* to determine if it is running inside of a virtual machine. (*Redpill* was discussed earlier in section 4.3. It works by exploiting imperfect virtualization of the x86 SIDT instruction.) However, the authors implemented a basic countermeasure to the *redpill* technique that involves searching for *redpill* and emulating the SIDT instruction, making detection even more difficult. Now with malware installed below the below the guest operating system, it is free to perform any number of tasks without having to worry about hiding itself or being removed by the guest OS.

SubVirt works by installing itself during system boot. Before this can happen, however, SubVirt needs a way of getting onto the target system's disk. This can be done by packaging SubVirt with other software as a Trojan, or by remotely exploiting the target machine. During installation, SubVirt is placed on the hard disk in specially allocated disk blocks using low-level drivers in order to bypass anti-malware protection mechanisms. The installer then modifies the boot sequence during the next shut down to run SubVirt. The next time that the machine boots up the user's operating system will be running in a virtual machine with SubVirt below it. SubVirt virtualizes the original disk, keeping most of the blocks in the same place and only using a small amount of space for its storage. SubVirt also virtualizes the small amount of extra physical memory it occupies by swapping extra pages out to disk. This way, the guest OS sees the same amount of

memory and the original disk contents without incurring a significant performance overhead.

One potential weakness of SubVirt is its liability to detection during system boot. Starting the virtual machine monitor and the guest operating system does add a significant amount of time to the normal boot process, going from a 23-second boot time to a 101-second boot time for a Windows XP target. To help mitigate this problem, SubVirt emulates restarts and shut-downs by going into power-saving mode without turning off the machine or restarting the virtual machine monitor. However, emulated restarts still take longer (54 seconds for a Windows XP target) than normal boots. Essentially, what these numbers mean is if somebody is suspicious and is looking for SubVirt, then that person will most likely be able to detect it by rebooting the machine. Typical users, however, do not closely monitor their boot times and will most likely not notice the difference, especially since boot times will vary naturally when new drivers and programs are installed on the system.

In summary, SubVirt poses significant challenges for designers of anti-malware protection programs. It exploits the isolation properties of virtual machines to achieve a new level of separation from the target operating system, making it very difficult to detect and remove. Perhaps the best method for dealing with this new breed of malware is to run VM-based intrusion detection systems on the host. This way, SubVirt may be able to move the guest OS into a virtual machine one level deeper, but any malicious programs run by SubVirt would still be subject to scrutiny by a VM-based IDS. Furthermore, VM introspection techniques may immediately detect the presence of SubVirt due to the addition of an intermediary operating system. It is always important to keep in mind how attackers might take advantage of state-of-the-art security mechanisms to protect their own malicious software.

# 5 Future VM Security System Research

While VM-based security systems have been a major focus of security researchers, there are still a number of challenging problems that have not been addressed. This section discusses some potential areas for future work on virtual machine security systems.

## 5.1 Secure User Interface

When a guest operating system is compromised, current virtual machine security systems do not prevent sensitive user input from going to the compromised machine. For example, if a hacker breaks into a guest OS and installs a key logger, then the hacker can obtain the user's password. Instead of sending the raw password input to the guest VM, it may be possible for the virtual machine monitor to expose a secure input field directly to the user, and instead send the password

hash to the guest VM. This way, the guest OS will never see the user's real password, keeping it safe from attackers who do not have access to the VMM layer.

There are a number of challenges with developing a secure input system. First, it is difficult to determine where passwords are checked in the guest OS and in its applications. Ideally, a secure input system should be able to automatically insert hooks in appropriate locations to support secure data entry without recompiling the target operating system or applications. Doing this, however, is a difficult problem. Another approach might be for the virtual machine monitor to export a secure input API that returns a password hash to the caller on the guest VM.

Another challenge to developing a secure input solution is making sure that an attacker cannot spoof the secure input interface on the user's video monitor. This can be achieved by providing a graphical display that cannot be forged or obstructed. There are various options available for addressing this problem. The first is to have the user input a specific key sequence, similar to ctrl+alt+delete, which will always trap to the VMM. The VMM can then display a secure input dialog box. Another option that is used in the NetTop architecture [MS00] is to reserve a small space at the top of the screen for exclusive use by the VMM. This space indicates which system has control of the display, and could be changed to say "VMM" during secure input entry. Implementing this type of secure display, however, would require modification to current commodity graphics hardware.

## 5.2 Secure Device Driver Domains

Device drivers are a critical part of every operating system. They mediate interaction between software and hardware, often running inside of the kernel at a high privilege level. Many device drivers are also very complicated, and some are even written by relatively unskilled programmers, making their quality highly suspect. These factors make driver code a serious security risk to the operating system. Empirically, driver code tends to be the worst quality code found in most kernels [CYC+01], as well as the greatest source of exploitable security vulnerabilities [AFS97].

Given that device drivers pose a serious security risk, it would be desirable to isolate them from the rest of the operating system. The resources available to each device driver should also be constrained to prevent them from affecting system stability. Virtual machine technology allows such isolation. If each driver runs in its own domain, then it will be unable to interfere with other parts of the operating system.

While having a domain for each driver would significantly improve system stability and security, there are a number of challenges that must addressed to achieve this goal. First, the device driver domain should be securely partitioned from the virtual machine monitor. The virtual machine monitor must run in an memory that the device domain cannot access directly. This would

require additional hardware support from the I/O MMU or a similar chip set. Second, this architecture requires secure communication between the guest VM and the device driver domain. No other virtual machines should be able to eavesdrop or tamper with this communication channel. Finally, the cost of partitioning device drivers into their own domains must be low. This is a very difficult problem because some devices have very high throughput and latency requirements. Switching to a separate domain for each driver call requires context switching, which can be very expensive for high-throughput devices. If driver domains slow down the system too much, then they are not likely to practical for real-world deployment

## 5.3 VM based Security Support for Portable Devices

With the increased number of attacks on portable devices such as cell phones and PDAs, security for portable devices has become a serious concern. Adapting virtual machine technology to function on portable devices would allow many of the security mechanisms discussed here to run on those devices as well, providing a higher level of security. There are several unique challenges, however, when running virtual machines on portable devices with limited resources.

One problem with running virtual machines on portable devices is that some virtual machine monitors, such as Xen, rely on hardware support for multiple privilege rings. While standard processors support a number of privilege rings, embedded processors rarely offer such capability. Getting around this limitation would require additional instruction emulation and translation, which will negatively impact performance. Another challenge that must be overcome to run virtual machines on portable devices is their limited amount of resources and computation ability. Things like battery life also start to become an issue when you are dealing with PDAs and cellular phones. It may also be costly to run multiple virtual machines due to memory requirements. Finally, portable devices will often have intermittent or no network connectivity. Security mechanisms such as trusted computing platforms may not be able to function properly without network-based verification. This makes digital rights management a difficult task for portable devices.

# References


[A N] A New Adore Rootkit. *http://lwn.net/Articles/75990*.

[AFS97] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 65. IEEE Computer Society, Washington, DC, USA, 1997.

[AMD] AMD Corp. AMD's Virtualization Solutions–Optimizing Enterprise Services. Http://enterprise.amd.com/en/Solutions/Consolidation/virtualization.aspx.

[Asl] M. Aslett. A virtual success. In *Computer Business Review Online*. *http://www.cbronline.com/content/COMP/magazine/Articles/Servers_Mainframes/AVirtual-Success.asp*.

[BC00] D. P. Bovet and M. Cassetti. *Understanding the Linux Kernel (Ed. A. Oram)*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000. ISBN 0596000022.

[BDF+03] 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. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM Press, New York, NY, USA, 2003.

[BDGR97] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447, 1997. URL `citeseer.ist.psu.edu/bugnion97disco.html`.

[Bea03] J. Beale. Detecting server compromises. In *Information Security Magazine*. TechTarget, Feb. 2003. *http://infosecuritymag.techtarget.com/2003/feb/linuxguru.shtml*.

[BP04] K. Borders and A. Prakash. Web tap: detecting covert web traffic. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 110–120. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-961-6.

[bre] How to break out of a chroot() jail. Http://www.bpfh.net/simes/computing/chroot-break.html.

[BZP05] K. Borders, X. Zhao, and A. Prakash. Sting: Detecting evasive malware. In *IEEE Symposium on Security and Privacy*. 2005.

[CCC+05] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: end-to-end containment of internet worms. *SIGOPS Oper. Syst. Rev.*, 39(5):133–147, 2005. ISSN 0163-5980.

[CN01] P. M. Chen and B. D. Noble. When virtual is better than real. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 133. IEEE Computer Society, Washington, DC, USA, 2001.

[Cor03] J. Corey. Local honeypot identification, September 2003. URL `http://www.phrack.org/fakes/p62/p62-0x07.txt`, http://www.phrack.org/fakes/p62/p62-0x07.txt.

[Cor04] J. Corey. Advanced honey pot identification, January 2004. URL `http://www.phrack.org/fakes/p63/p63-0x09.txt`, http://www.phrack.org/fakes/p63/p63-0x09.txt.

[Cre81] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490, 1981.

[CYC+01] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An empirical study of operating systems errors. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 73–88. ACM Press, New York, NY, USA, 2001. ISBN 1-58113-389-8.

[Dik00] J. Dike. A user-mode port of the Linux kernel. In R. Spenneberg, editor, *Proc. of the 4th Annual Linux Showcase & Conference*. Oct. 2000. URL `http://www.linuxshowcase.org/2000/2000papers/papers/dike/dike.pdf`.

[DKC+02] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.*, 36(SI):211–224, 2002. ISSN 0163-5980. URL `http://dx.doi.org/10.1145/844128.844148`.

[F-S] F-Secure Corp. F-Secure Virus Descriptions : Agobot. Http://www.f-secure.com/v-descs/agobot.shtml.

[Gol72] R. Goldberg. Architectural principles for virtual computer systems, 1972. URL `citeseer.ist.psu.edu/mazieres97security.html`, ph.D thesis, Div. of Eng. and Applied Physics, Harvard U., Cambridge, Mass.

[Gol74] R. Goldberg. Survey of virtual machine research. *IEEE Computer Magazine*, 7:34–45, June 1974.

[GPC+03] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP*, pages 193–206. 2003.

[GPCR04] T. Garfinkel, B. Pfaff, J. Chow, and M. Rosenblum. Data lifetime is a systems problem. In *Proc. 11th ACM SIGOPS European Workshop*. September 2004.

[GR03] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, pages 191–206. February 2003. URL `citeseer.ist.psu.edu/garfinkel03virtual.html`.

[GR05] T. Garfinkel and M. Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS-X)*. May 2005.

[GTHR00] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 18(3):229–262, 2000. URL `citeseer.ist.psu.edu/govil99cellular.html`.

[Gum83] P. H. Gum. System/370 extended architecture: Facilities for virtual machines. *IBM Journal of Research and Development*, 27(6):530–544, 1983.

[HWS04] H. Höxer, M. Waitz, and V. Sieh. Advanced virtualization techniques for FAUmachine. In R. Spenneberg, editor, *11th International Linux System Technology Conference, Erlangen, Germany, September 7-10, 2004*, pages 1–12. 2004.

[Int] Intel Corp. Intel Delivers New Era For Virtualization. Http://www.intel.com/pressroom/archive/releases/20051114comp.htm.

[JKDC05] A. Joshi, S. T. King, G. W. Dunlap, and P. M. Chen. Detecting past and present intrusions through vulnerability-specific predicates. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 91–104. ACM Press, New York, NY, USA, 2005. ISBN 1-59593-079-5.

[JX04] X. Jiang and D. Xu. Collapsar: A vm-based architecture for network attack detention center. In *USENIX Security Symposium*, pages 15–28. 2004.

[KC06] S. King and P. Chen. Subvirt: Implementing malware with virtual machines. In *IEEE Symposium on Security and Privacy*. Oakland, California, May 2006. URL citeseer.ist.psu.edu/666505.html.

[MD73] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the workshop on virtual computer systems*, pages 210–224. 1973. URL http://portal.acm.org/citation.cfm?id=803961.

[MS00] R. Meushaw and D. Simard. Nettop: Commercial technology in high assurance applications. *Tech Trend Notes: Preview of Tomorrow's Information Technologies*, 9(4), September 2000.

[RG05] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005. ISSN 0018-9162.

[RI00] J. S. Robin and C. E. Irvine. Analysis of the intel pentium's ability to support a secure virtual machine monitor. In *Proceedings of the 9th USENIX Security Symposium*, pages 129–144. Aug 2000.

[RK03] L. H. J. Rick Kennell. Establishing the genuinity of remote computer systems. In *Preceedings of USENIX security Symposium*, pages 295–308. August 2003.

[Rut] J. Rutkowska. Red Pill... Or How To Detect VMM Using (Almost) One CPU Instruction. Http://invisiblethings.org/papers/redpill.html.

[Spi03] L. Spitzner. Honeytokens: The other honeypot, July 2003. URL http://www.securityfocus.com/infocus/1713, http://www.securityfocus.com/infocus/1713.

[SVJ+05] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. van Doorn, J. L. Griffin, and S. Berger. sHype: Secure hypervisor approach to trusted virtualized systems. Technical Report RC23511, IBM T.J. Watson Research Center, Feb,2005.

[SVL01] J. Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14. USENIX Association, Berkeley, CA, USA, 2001.

[tam01] CERT/CC Security Improvement Modules: Analyze all available information to characterize an intrusion. Technical report, CERT Coordination Center, May 2001.

[TT02] T. Ts'o and S. Tweedie. Planned extensions to the Linux Ext2/Ext3 filesystem. In *Proc. of the FREENIX Track: 2002 USENIX Annual Technical Conference*, pages 235–244. 2002.

[VMC+05] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 148–162. ACM Press, New York, NY, USA, 2005. ISBN 1-59593-079-5.

[Wal02] C. A. Waldspurger. Memory resource management in vmware esx server. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 181–194. ACM Press, New York, NY, USA, 2002.

[WCSG05] A. Whitaker, R. S. Cox, M. Shaw, and S. D. Gribble. Rethinking the design of virtual machine monitors. *Computer*, 38(5):57–62, 2005. ISSN 0018-9162.

[Whi] A. Whitaker. Building Robust Systems With Virtual Machine Monitors. University of Washington Generals Examination, 8-17-04, http://www.cs.washington.edu/homes/andrew/papers/generals.pdf.

[WSG02] A. Whitaker, M. Shaw, and S. Gribble. Scale and performance in the Denali isolation kernel. *SIGOPS Oper. Syst. Rev.*, 36(SI):195–209, 2002. ISSN 0163-5980.

[ZBP05a] X. Zhao, K. Borders, and A. Prakash. Svgrid: A secure virtual environment for untrusted grid applications. In *CM/IFIP/USENIX 6th International Middleware Conference, Grenoble, France*. 2005.

[ZBP05b] X. Zhao, K. Borders, and A. Prakash. Towards protecting sensitive files in a compromised system. In *3rd International IEEE Security in Storage Workshop*. 2005.

[ZQ] J. Zhou and L. Qiao. Backdoor and Linux LKM rootkit - smashing the kernel at your own risk. Project report: *http://citeseer.ist.psu.edu/giffin04efficient.html*.