Undetectable Monitoring in a Fully-Virtualized Environment –
A Continuation of the HAL Keystroke Logger

Michael Kranch  Roy Ragsdale
Department of Electrical Engineering and Computer Science,
United States Military Academy, West Point, NY 10996
Michael.Kranch@us.army.mil, Roy.Ragsdale@usma.edu

Abstract

Virtualization is ever an expanding research field and, as many predict, the way of the future for large scale business and server solutions. Originally designed as a method of centralizing physical resources and maintenance, recent research has developed methods of also utilizing virtualization for centralizing machine monitoring. Recently, there have been substantial advances in centralized monitoring in a virtualized environment[1]. Specifically, researchers at the Georgia Tech have developed XenAccess, a system for monitoring memory in a paravirtualized environment [2]. This paper highlights the differences between two popular virtualization methods, paravirtualization and full-system virtualization. A comparison between techniques used by XenAccess to those implemented in our undetectable Hardware Abstraction Layer (HAL) Keystroke Logger is then presented before expanding the original HAL template and finally discussing in detail methods to monitor disk access and memory management.*

1. Introduction

In terms of computing, virtualization is a rather broad term that simply refers to the abstraction of hardware. Although virtualization was originally developed in the 1960’s, significant advances have been made in both the technology and its application in the earlier part of this decade. These advances have moved virtualization back to the forefront of computer software and engineering research, specifically in the field of platform virtualization [3]. Traditionally, a computer has one set of hardware that runs one operating system that then runs numerous applications. Platform virtualization is a type of virtualization that uses software to allow a user to run multiple simulated computer environments, or virtual machines (VMs), on one set of hardware, similar to how a traditional operating system runs applications.

Platform virtualization software has increased in popularity because of advancements in computer hardware. Current hardware, such as multi-processor systems and terabyte hard drives, are too large for an average user to fully utilize. As such, large businesses have turned to virtualization technology to achieve maximum efficiency in the use of their servers and user workstations.

![Figure 1. How virtualization works](image)

* The views expressed are those of the author and do not reflect the official position of the United States Military Academy, the US Army, or the US Department of Defense.
paravirtualization or full-virtualization system, such as VMware.

The following sections provides a background on how VMs are implemented and why they need to be monitored for security reasons before outlining novel approaches to monitoring in paravirtualization and fully-virtualized environments. How this has been achieved already by this research effort through the creation of the HAL keystroke logger, parser and viewer is then presented. The results of this work as well as how to monitor other devices in the QEMU Project are outlined before concluding on the potential significance of this endeavor.

2. Background

In modern day computing, there are numerous situations that involve the monitoring of computer systems. These monitoring schemes vary from the good, such as a virus scanner constantly monitoring a system with the intent of catching or alerting on any malicious activity, to the bad, such as a rootkit installed by a malicious user with the intent of stealing private user passwords and accounts. In addition, there are knowledge focused monitoring schemes like the Honeynet Project that utilizes advanced monitoring techniques to record the activities of hackers [6]. Although differing in design, all of these monitoring schemes have a similar implementation in that they involve modifying the host, or monitored machine, with software that interacts with low-level functions of the operating system to capture and view selected system events.

The problem with these monitoring systems is two-fold. First, although the modification of the host system is necessary, this modification also allows outside applications to detect the monitoring software. Because outside application can detect the monitoring software, they can also implement measures to change or subvert the monitoring software. The Ramex.d worm (also known as Pyskpa.d by Symantec) launched within Skype during late 2007 is a prime example of this problem [7]. Like maybe new worms and virus, Ramex.d corrupts explorer.exe and modifies the Windows host file in order to prevent updates and to prevent access to anti-virus software sites. This modification prevents the anti-virus software from ever detecting the worm even after the worm’s signature has been identified by the anti-virus software distributors. Because the monitoring software resides within the host operating system, the monitoring software remains vulnerable to new malicious software.

Secondly, these current monitoring schemes are designed for single computer monitoring. In a virtualized system, one set of hardware can support multiple virtual machines. Using current monitoring methods, each of these virtual machines would need its own monitoring software. As such, each machine would also need to be maintained and updated as new virus definition become available. This individual monitoring method in this case is cumbersome and does not utilize the advantages provided by operating in a virtualized environment.

2.1. Anti-Virus Software

In order to understand how to develop an undetectable monitoring scheme within Xen, one must first understand how a traditional monitoring scheme works, such as anti-virus software. There are four major methods of virus detection widely in use today: scanning, integrity checking, heuristics, and interception [8]. However, most of these methods work in generally the same way. They scan all files on the hard drive and memory in order to perform an analysis on these files to determine if viruses exist. This analysis is done by comparing these files to a list of known viruses (in traditional scanning) by comparing each to a previously calculated checksum of that file (integrity checking) or by performing some internal analysis of code of the file to predict if it might be a virus (heuristics). Interception virus scanning is the only method that differs from this general scheme. Instead of actually scanning system files, interception detectors wait for a file to do something unusual or characteristic of a virus, such as relocating itself in memory or installing itself into an currently operating system file. When this happens, the monitoring scheme intercepts the action and then warns the user. Since interception detectors are rare, we focus on the other three, more traditional virus scanners.

Although differing in how they analyze a file to detect a virus, the other three scanners operate in essentially the same manner. Anti-virus scanners function in essentially two modes: on-demand and on-access [9]. On-demand scanning takes place when the system user commands -- either directly or through the use of a schedule -- a scan by the anti-virus software of a machine. On-demand scanning scans the entire machine including main memory, the boot sector, disk memory as well as all internal and external drives. On-demand scanning is fairly system intensive and is used
to detect viruses already on the host machine. The second type of virus scanning is on-access scanning. This type of scanning is far less system intensive. On-access scanning occurs when a file is first loaded into memory. If the scanner determines the file is a virus, it prevents the file from being loaded into memory. Since memory plays such an important role in the monitoring of a system, the next thing one needs to understand in order to create an undetectable virtualized monitoring scheme is how a virtualized system emulates memory.

2.2. Related Work

As the field of virtualization grows, research related to it continues to increase. First, because Xen is an open-source project, there is a copious amount of knowledge available on the Xen support forum. Our project grew significantly due to the knowledge sharing available on the internet. In addition, this project is specifically related to two recently virtualization research projects. One is Blue Pill, a recent project by Joanna Rutkowska’s that was supposedly an undetectable method of infecting a system with malware through the use of virtualization [10]. In Blue Pill, Rutkowska explored the use of a very thin Virtual Machine Monitor as a way of tricking the operating system into thinking it was running on normal hardware when, in fact, it was virtualized. Rutkowska was then able to run malware in this thin virtualization layer. Blue Pill’s goal was to hide from the operating system the fact that it was in a virtualized environment, thereby making the malware hidden in the virtualization layer undetectable. While the malware within this layer is undetectable, the fact that the system was virtualized was not [11]. Researchers have shown that system virtualization is detectable due to the fundamental unavoidability of virtualization’s processor and memory footprint. Unlike Blue Pill, our project is running on an already known and preexisting virtualized environment; therefore, the ability to detect a virtualized environment is irrelevant.

The second research project related to our project is XenAccess. XenAccess is a similar project that monitors multiple guest domains (DomU’s) through the use of a specially designed monitoring DomU. Specifically, XenAccess provides virtual memory introspection and virtual disk monitoring capabilities. However, XenAccess works specifically with Xen in paravirtualized mode. Also, XenAccess monitors only virtual memory and not all memory.

3. Methodology

This paper introduces multiple undetectable monitoring techniques based on both of Xen’s device management methods. The discussion will begin with an explanation of how both paravirtualization and fully-virtualized systems work. This section will then explain the methodology behind the HAL Keystroke logger, (an undetectable keystroke logger utilizing a fully virtualized environment).

3.1. Paravirtualization

Paravirtualization is a method of virtualization that required modification of the virtualized operating systems. In paravirtualization, a virtualization engine, commonly referred to as virtual machine monitor or virtual machine manager (VMM), runs directly on the hardware. This VMM acts like a special operating system and has direct control over all the system resources [12]. Instead of applications, the VMM runs other operating systems (VMs). The VMM is responsible for allocating systems resources to these operating systems as needed. In some instances, the VMM assigns exclusive access to a resource to a virtualized operating system (also called a guest domain, or DomU), while in others it does a form of time-based sharing, such as with processor access. The VMM also manages resource in a third, more complicated way. In some cases, such as with network access, the VMM actually splits the I/O stream, tricking each DomU into believing they have direct access to the device. The VMM also assigns separate virtual MAC access to each VM so they can talk to each other over the same I/O stream [13].

In a paravirtualized mode, one of the most difficult resources for the Virtual Machine Monitor to manage is the system’s memory. To achieve this task, Xen separates memory in three separate addressing levels: machine, physical, and virtual. The machine addresses are the actual addresses used by the hardware and managed by the VMM. These machine addresses are utilized by the VMM in the same way that a single operating system uses physical addresses. Each virtual machine accesses memory through the use of physical addresses. To the individual virtual machine, these physical addresses are accesses in the same way that a single machine system utilizes physical address. In reality however, these physical addresses are translated by a look-up table in the VMM to machine addresses [14]. This method works because the VMM tricks the guest operating systems into believing they are
communicating with the hardware memory. The third level of addresses, virtual addresses, are used in a paravirtualized system in much the same way as a single machine system, with each VM receiving its own distinct virtualized memory location.

In reality, this “tricking” of the virtualized operating systems into thinking they are operating directly on the machine’s hardware is a very difficult to achieve because the system’s hardware was originally designed to run with only one operating system. A traditional operating system maintains control over its resources through the use of privilege modes designated within the processor chip. In a traditional x86 processor, there are two modes: Ring 3, or “user mode”, and Ring 0, or “privileged mode”. The computer’s operating system uses privileged instructions for its system kernel, device drivers, memory mapping, interrupt handling and other vital operating system instructions [15]. The operating system then uses Ring 3 instructions for everything else, primarily for the user applications running on the system. The operating system separates these instructions so it can manage the system resources appropriately. If an application is requesting system resource to run, the kernel can manage those instructions and interrupt them with Ring 0 instructions as needed.

This two mode system causes a problem when you introduce VMMs. There is no “-1” ring for the VMM to use in order to interrupt the Ring 0 instructions from the various VMs requesting system resources. In order to solve this, slight modifications to each operating systems’ device drivers were required in order for the original operating systems to be virtualized. These changes allowed the VMM to manage the virtualized operating systems. Instead of talking directly to the hardware, these modifications enable the modified “frontend” drivers in each DomU to talk to “backend” drivers provided by the VMM. These backend drivers then interacted directly with the hardware for each virtualized domain.

3.2. Fully-Virtualized Environments

Fully-Virtualized Environments are a recent advancement in virtualization that do not require modification of the guest operating systems. These environments utilize recent additions to the computer’s processor to operate. In a fully-virtualized environment, the various DomUs communicate directly with the hardware instead of operating through backend drivers provided by the VMM [16].

Figure 2. Ex. of para-virtualization vs. HVM in Xen

In order for this action to work, the entire virtual system memory space is pre-allocated. “A system BIOS boots inside this space, much like a full PC’s BIOS would boot, providing a real-mode int13 interface to emulated chipsets inside the virtual machine. The guest operating systems then boots and loads device drivers that directly interface with these emulated chipsets” [17].

Xen utilizes the additional functionality of the AMD SVM (Pacifica) and Intel VTX (Vanderpool) chipsets to create a fully-virtualized environment. The AMD SVM (Pacifica) chip has two additional modes, Host Mode and Guest Mode, and an additional instruction set, VMRUN, designed specifically for virtualization. When these newly design chips boot, they start in guest mode until a capability VMM activates Host mode. In Host mode, a VMM can define in hardware the characteristics of each DomU. The VMM is able to do this action through the use of virtual machine control blocks (VMCBs). In these VMCBs, the VMM specifics the details of each DomU, including processor access, IO access and memory allocation [18]. Once defined, the VMM switches into guest mode and gives control of the system to each of the guest operating systems through the use of the VMRUN instruction. For the majority of DomU instructions, the microprocessor simply refers to the VMCBs for information on how to handle the instruction. In the case of more complicated instructions, the microprocessor passes control back to the VMM. The VMM then evaluates the instruction, makes a determination passed on the instruction, (possibly) updates the VMCBs and goes back into Guest mode. While not exactly the same, the Intel VTX works in a very similar manner.
3.3. The QEMU Project

For each VM in HVM, Xen provides an abstracted view of the pc hardware. In order to reduce the amount of device emulation necessary, Xen decided to utilize the emulated IO devices from the preexisting QEMU Project. The QEMU Project simulates a set list of software devices. The list of peripherals Xen 3.2 uses the QEMU Project to simulate includes the following [19]:

- i440FX host PCI bridge and PIIX3 PCI to ISA bridge
- Cirrus CLGD 5446 PCI VGA card
- PS/2 mouse and keyboard
- 2 PCI IDE interfaces with hard disk and CD-ROM support
- Floppy disk
- PCI/ISA PCI network adapters
- Serial ports
- Creative SoundBlaster 16 sound card,
- PCI UHCI USB controller and a virtual USB hub

The code for all the hardware devices within the QEMU Project is located within the Xen 3.1 source code at /tools/ioemu/hw/.

3.4. HAL Keystroke Logger

As discussed in Section 1, virtualization leverages the increasing performance of computers to host multiple guest operating systems on a single server. Each of these guest operating systems utilizes a hypervisor layer to interpret commands to be executed on the computer hardware. As a proof of concept, we first built a Hardware Abstraction Layer (HAL) keystroke logger that uses this interpretation layer to monitor activity on the guest operating system to invisibly log every keystroke from all guest operating systems.

As noted in Section 3.3, the QEMU Project emulates a PS2 keyboard and mouse. The actual code for the keyboard device emulation is located at /tools/ioemu/hw/ps2.c. This file works similarly to how a keyboard device would in a normal system. When then Xen host domain starts up a new DomU, the host domain creates and initializes a new device for each device associated with the new DomU. For the keyboard, Xen runs ps2.c’s ps2_kbd_init() procedure. This procedure updates the IRQ (interrupt request) for the keyboard as well as sets ps2_put_keycode as the keyboard event handler.

When the user in a DomU presses a key, this action generates a level 1 interrupt request (IRQ1).

This interrupt then passes the user generated raw scan code to the ps2_put_keycode procedure via the predefined event handler. Ps2_put_keycode() then calls ps2_raw_keycode(), another procedure that converts the raw PC scan code to a raw keycode. Ps2_put_keycode() then calls ps2_queue(), which eventually updates the IRQ and passes the translated keycode back to the DomU. These keycodes are translated based on a preset keymap dependent on the language of the user. The resulting value is passed back to the DomU for interpretation. In ps2.c, we inserted code within the ps2_put_keycode() procedure that appends this value to file before the procedure passes the keycode to the IRQ queue.

```
static void ps2_put_keycode(void *opaque, int keycode)
{
    PS2KbdState *s = opaque;
    if ((opaque & 0x80) == 0)
        ps2_queue((opaque & 0x80);
    else
        ps2_queue((opaque & 0x80);
    keycode = ps2_raw_keycode(keycode & 0x7F);
    ps2_queue((opaque & 0x80);

    outPut(keycode);
}
```

Figure 3. Modification to Ps2.c

We also uniquely named this file based on the Domain name associated with the emulated device. The variable domain name within ps2.c contains the name of the DomU. Since each DomU has a unique name and a unique set of emulated devices, this naming convention allows us to differentiate between the various domains.

Essentially what this is doing is the equivalent of in a traditional, non virtualized setup, placing a hardware based monitoring device within the physical keyboard. Such a device, like the commercially available KeyGhost Keylogger, offers many attractive features, such as being “[i]mpossible to detect and/or disable by using software scanners”, the ability to “record every keystroke, even those typed in the critical period between computer switch on and the operating system being loaded”, and working “with any PC operating system” [20]. In a traditional scheme, such a hardware device would only be detectable through visual inspection. Through our modification of the device emulation layer in a virtualized setup, we achieve all of these goals and then some. For one, it is completely undetectable because there is no way for the user to physically inspect the keyboard, it only exists in code. Additionally, where a traditional physical solution can only monitor a single machine at a time, our solution...
has the ability to monitor any number of Virtual Machine instances simultaneously. Finally each traditional hardware based monitoring solution would cost not only money but the time required to inspect many physically separated devices. With our methodology all that is required is a patch to the Virtual Machine Monitor and then all logs are consolidated in a single location. Thus in the emulation layer our solution provides the undetectable and unavoidable qualities of a traditional hardware based key logging scheme with the ease of a software based scheme.

3.5. Keycode parser and viewer

Since we capture only the raw keycodes passed from the various DomU to the emulated keyboard device, we first need to apply a keymap to translate these codes into actual text. Alone these keycodes are not necessarily the most useful, so we also designed a translator based on the common keymap that converts each keycode into a human readable log of exactly what was typed. By separating the parts of code that actually modify Xen to log the keycodes from the code that analyzes the logs, the overhead required in monitoring a user’s keystrokes is reduced. By separating the keycode logger and viewer, the analysis required to interpret the keystrokes is done only as needed, instead of adding additional overhead continually to Xen. This separation also allows the logs to be used for multiple purposes. Our viewer has two modes: readable text and total translation mode. An additional view could easily be created to search the logs for dirty words, such as “bomb” or “murder”. In addition, the modifications we made to Xen are minor and easily portable with newer versions of Xen.

4. Results

The creation of the HAL keystroke logger achieves exactly what we sought to do: monitor all of the keystrokes of each DomU without making any modifications to the guest domains. We verify our ability to monitoring a DomU without modification utilizing live CDs for our DomU. Live CDs are small operating systems (ISOs), such as the Helix Forensic ISO or Damn Small Linux (DSL), that boot direct from the original ISO every time the operating system is loaded. Because they boot directly from the original ISO, no one can modify the operating system. We also verified that the log files translated from our parser represent the actual input in the various DomUs. For example, we start a HVM from the DSL live CD. We then connect to this domain using VNC viewer from a secondary machine to get access to the domain’s graphical interface. Once connected, we start pressing keys and recording our values. We then verify the output is, in fact, the same keycode. For instance, if we pressed the “a” key, our log file writes “30 158”. The keycode for “a” is 30 and the keycode for the release of a key is that key’s keycode + 128, thus 158 (30 + 128) is derived for the release of the “a” key. This works for all the keys on the keyboard including special modifiers like the shift key. Then using a simple table lookup in our keycode parser, we are able to easily translate the codes into the human readable ASCII text, just as it was entered, with annotations for special key modifiers. We are also able to capture keystrokes in any other DomU’s running at the same time.

5. Monitoring Additional Devices

The HAL keystroke logger was intended to serve as a template for full system monitoring. Since the keystroke logger was a success (the only one of its kind in the literature), we began examining additional devices to see if the same methodology would work. Originally, we intended to monitor the network card, all system drives and the system memory in order to create our full system monitoring scheme. In this section, we discuss how to monitor some of these additional devices.
5.1. SCSI Disk

One of the devices emulated by the QEMU Project is a SCSI hard drive disk. This file used to emulate a SCSI disk is located at /tools/ioemu/hw/scsi-disk.c. As originally hypothesized, all QEMU devices follow the same general template. When spawning a new DomU configured to have a scsi disk, Xen calls scsi_disk_init to initialize the disk. Instead of initializing an event handler, this function returns a file of type SCSIDevice based upon the pre-configured size. In additional to scsi_disk_init(), there are essentially three other functions within scsi-disk.c: scsi_send_command(), scsi_write_data() and scsi_read_data().

Send command is the most important of these three. This function executes a scsi command and returns the length of data expected by the command. This value will be a positive int for disk reads and negative for disk writes. Based on the outcome of send_command, the code then executes either write or read data. In both read and write, the function iterates through the data based on a preset sector length of 512 byte. For every 512 bytes, the respective function then calls either bdrv_read or bdrv_write. Bdrv is the variable name for the Block Driver State. The QEMU Project created a single class to handle all hard drive access, regardless of the perceived emulation type of device. The generic QEMU Emulator block drive is located at /tools/ioemu/block.c. This device works in the same way as the scsi device. Block.c has an init file to register the various possible block devices. For instance, block.c registers block-vvfat.c, the QEMU Block Drive for virtual VFAT. Block.c also has a raw_read and raw_write. These two procedures are actually responsible for reading or writing the data from the disk one 512 byte sector at a time.

Once we determined how the system writes and reads to data, we focused our attention on how to monitor this data. While certainly theoretically possible, we are still in the final stage of determining a method of monitoring individual sectors of information because this achievement would be particularly useful to developing a monitoring scheme that currently does not exist. As explained in Section 2.1, a virus scanner works by scanning the hard drive on demand or the memory on access. Doing on access monitoring of the hard drive is redundant. The VMM already has access to the entire physical hard drive; a simple on-demand scan in the VMM would also scan the hard drives of every VM stored on the host machine. This virus scanner would have to have a larger backend database than a traditional scanner since it would potentially work with multiple operating systems; however, a single scan from the host domain would scan the guest domains as well. As such, this single scan eliminates the need for a separate monitoring scheme to provide on-demand scanning of the various DomUs.

5.2. HVM Memory Management

In HVM, or full virtualization mode, memory is handled quite differently than in paravirtualized mode. Since each VM has direct access to the hardware, the VMM does not handle the memory lookup table. Instead, the memory lookup is handled directly between the DomU and the hardware. As stated previous, Xen HVM provides an individual interface for each DomU to memory. Xen HVM also utilizes VMCBs to allow this interaction between the individual guest operating systems and the actual hardware.

The QEMU Project uses the KQEMU package for memory emulation. This package is located at /tools/ioemu/kqemu.c. Similar to the previous QEMU device, this file has a kqemu_init() procedure that initializes memory emulation. This procedure checks to confirm if a block of memory is already in use, flushes that memory and then sets the memory’s dirty byte (a unique byte used to determine if memory is currently in use). The KQEMU file has various helper procedures that help control the memory block, such as flush memory and set dirty. This file also has other procedures, such as restore_native_fp_frstor, save_native_fo_frstor, kqemu_record_pc, and kqemu_record_flush. Despite the kqemu.c file being extremely complicated, our investigation revealed that the VMM uses it to manage memory in event of a conflict within the VMCBs. With further research, modifications within this file and device driver should provide the user with the ability to monitor the system’s memory to create a full-system virus scanner.

6. Future Work

Given the success of our hypervisor keystroke logger, research will continue on other emulated devices that can be tapped as well as the actual DomU processor calls. With the appropriate taps, a full system-wide monitoring system would be possible. Some uses for such a system could be an anti-virus software that cannot be subverted or disabled. Xen HVM monitoring also seems particularly geared towards individual device monitoring. Since fully-virtualized environments provide individual emulate
devices to each DomU, monitoring these devices in a
dynamic matter is relatively easy. By modifying the
code that provides network emulation, one could easily
create a system that captures and logs all network
traffic from each DomU. By combining a network
monitoring scheme with the HAL keystroke logger,
one could create a monitoring scheme that would be
very useful to the Honeynet Project. This proposed
scheme would allow the Honeynet Project to
undetectably monitor both an attacker’s input
commands once on the system while capturing all
malicious files uploaded onto the system. While this
paper provides a template for monitoring individual
devices, additional work is needed to create a full-
system monitoring schemes. In order to create this
scheme, additional work is required to determine a way
of centralizing memory calls in Xen HVM mode and
developing a system that would integrate the multiple
DomU’s into a single monitoring scheme.

7. Conclusion

As computer virtualization spreads and becomes
more commonplace, more people will find themselves
on a virtualized machine and give it less and less of a
second thought. Thus the security advantages of an
undetectable monitoring scheme will become more
wide-spread and valuable. Our unique, undetectable
keystroke logger shows the viability of an undetectable
monitoring solution through the hypervisor layer for
virtual machines. The transmission of the actual
hardware results through the hypervisor to virtual
instances can be easily tapped with modifications to
the hypervisor, and the resulting logs can then be
monitored as needed.

Fully-virtualized systems are advantageous because
they can be utilized with unmodified guest domains.
They are able to work with unmodified guest domains
because they present unique emulated devices to each
new DomU. However, this process also helps to
decentralize the domains. While this decentralization
makes simple, individual device monitoring schemes,
like a keystroke logger, easier to implement on a fully-
virtualized system, it also makes monitoring more
complicated systems, like memory, more difficult to
implement because the management of these systems
are not done in a single place or by a single device.
Nevertheless, this work strongly suggests that a full
system monitoring scheme is viable for virtual
environments and, as such, warrants the attention of
the research community to make this a reality.

8. Acknowledgement

We wish to extend special thanks to LTC Ron
Dodge and Colonel Grant A. Jacoby from the United
States Military Academy’s Information and
Educational Technology Division and the Department
of Electrical Engineering and Computer Science
respectively for their help and guidance in conducting
this research.

9. References

Monitoring through a Virtualized Environment – HAL
Keystroke Logger.”

Lee, “Secure and Flexible Monitoring of Virtual Machines.”

Available at <http://www.kernelthread.com/publications/virtualization/>.

Available at <http://www.kernelthread.com/publications/virtualization/>.

[5] “VMware Customers.” Available from

<http://www.honeynet.org/misc/project.html>.

[7] “Ramex.a Worm is Bad News for Skype Users.”
Available from <http://www.uberreview.com/2007/09/ramexa-worm-is-bad-
news-for-skype-users.htm>.

<http://cse.stanford.edu/class/cs201/Projects/viruses/anti-
virus.html>.

<http://cse.stanford.edu/class/cs201/Projects/viruses/anti-
virus.html>.


