NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

THE IMPLICATIONS OF VIRTUAL MACHINE INTROSPECTION FOR DIGITAL FORENSICS ON NONQUIESCENT VIRTUAL MACHINES

by

Nathan Hirst

June 2011

Thesis Advisor:	Chris Eagle
Second Reader:	George Dinolt

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2011	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE The Implications of Virtual Machine Introspection for Digital Forensics on Nonquiescent Virtual Machines		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Nathan Hirst			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Science Foundation Grant No. DUE-0414012		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number N/A.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

The use of virtualized servers is on the rise, this results in a need for better forensic analysis capabilities for these virtualized environments. One of the answers to that has been the development of virtual machine introspection tools. Virtual machine introspection is a relatively new technique that has some important implications for digital forensics. Since it is performed outside of the virtual machine, it can help to alleviate the observer effect that is often encountered when performing a live analysis. This thesis tests how these tools can work in a nonquiescent environment and shows that the tools tested are able to produce reliable results.

14. SUBJECT TERMS Virtual Machine Introspection, VMI, Virtual Machine, Forensics, Xen			**15. NUMBER OF PAGES** 61
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

THE IMPLICATIONS OF VIRTUAL MACHINE INTROSPECTION FOR DIGITAL FORENSICS ON NONQUIESCENT VIRTUAL MACHINES

Nathan W. Hirst
Civilian, Naval Postgraduate School
B.S., Point Loma Nazarene University, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2011

Author: Nathan Hirst

Approved by: Chris Eagle
Thesis Advisor

George Dinolt
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The use of virtualized servers is on the rise. This results in a need for better forensic analysis capabilities for these virtualized environments. One of the answers to that has been the development of virtual machine introspection tools. Virtual machine introspection is a relatively new technique that has some important implications for digital forensics. Since it is performed outside of the virtual machine, it can help to alleviate the observer effect that is often encountered when performing a live analysis. This thesis tests how these tools can work in a nonquiescent environment and shows that the tools tested are able to produce reliable results.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
GB	Gigabyte
GUI	Graphical User Interface
OS	Operating System
PC	Personal Computer
RAM	Random Access Memory
VIX	Virtual Introspection for Xen
VM	Virtual Machine
VMI	Virtual Machine Introspection
VMM	Virtual Machine Monitor

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

A master's degree is a huge undertaking, and I could not have done it without the support and encouragement of countless people, from professors and classmates to family and friends. I especially want to thank my wife, Megan, for her patience and encouragement.

This material is based upon work supported by the National Science Foundation, under grant No. DUE-0414012. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. INTRODUCTION

The recent growth of virtualization and cloud computing has created the need for a new generation of incident response and forensics analysis tools capable of operating effectively in these new environments. Investigators require forensically sound tools and techniques that have been validated to yield correct results while minimizing their impact on the systems being investigated.

Many data centers are moving to virtualized servers and platforms. Since most incident response and investigations occur in data centers, it makes sense that with this trend, responders and digital forensic investigators will increasingly be required to work with virtual environments. In fact, many books and papers now cover the new challenges that virtualization presents to the forensics process [1], [2], [3]. Virtualization affords the opportunity for multiple virtualized instances to reside on a single physical machine. Each of these instances remains logically separated from the other but intermixed physically. One area of concern is the potential to escape from a virtual machine and break the logical separation between the machines allowing the possibility for breaking into other virtual machines residing on the same physical host. Even without the issue of breaking into another machine, performing forensics on a machine hosting multiple virtual machines raises a number of issues.

This thesis will investigate the utility of Virtual Machine Introspection (VMI), specifically if XenAccess, developed by Bryan Payne [4], will work on a nonquiescent virtual machine. By nonquiescent we mean an active virtual machine. This is as opposed to a paused virtual machine that much research so far has focused on. When forensic investigators are examining a new virtual machine where an incident may have taken place, it is important for them to know how reliable the tools that they are working with are. This thesis will perform some tests of selected virtual machine introspection tools on a nonquiescent virtual machine and compare the results with those obtained from a paused virtual machine to see how reliable the results they produce are.

Virtual Machine Introspection is an important new area of research. It can help to solve the problem of the observer effect in digital forensics. When an investigator examines a running machine, they will change the state of the machine simply because they are loading tools into that machine's memory in order to examine it, but with VMI the tools are executed outside of the virtual machine and therefore do not impact the memory of the virtual machine being investigated. With this development, the state of the art with virtual machine forensics is advancing.

B. ORGANIZATION OF THESIS

This thesis is organized into five chapters. The first is an introduction to the subject. The second chapter explores the background information necessary to understand the digital forensics process. The third explains the experimental environment and the methodology used in evaluating virtual machine introspection tools. The fourth details the results from the experiments and gives a conclusion for the thesis. Finally the fifth chapter highlights some related and future work.

II. BACKGROUND INFORMATION

This thesis will explore areas that require some background before we begin. The areas that will be covered briefly are: digital forensics, virtualization and virtual machine introspection.

A. DIGITAL FORENSICS

Digital forensics may be defined in many different ways, often depending on the source of the definition and the context in which forensics is being performed [5], [6]. Some definitions focus on observing state information of a system to gain a better picture of what is happening on that system. More traditional definitions will say that forensics deals with the handling of legal evidence; although, the techniques and applications from digital forensics are quite often applied outside of the legal setting. Digital forensics emerged from the field of forensic science, which has to do with secure handling of evidence, usually for a legal setting. The field of forensics needs to be very exact, since the evidence that is collected may be used to prosecute and possibly incriminate someone. Digital forensics focuses on obtaining and analyzing digital information for criminal, civil, and administrative cases [6].

Digital forensic investigators focus their efforts on the collection and analysis of electronic evidence. Often, this involves the collection and analysis of drive images (bit for bit copies of the contents of a storage device), as well as the collection and analysis of data collected from a running computer. There are many challenges that this presents, one of the biggest is to preserve the integrity of the data on the machine and collecting that data in a secure and reliable way. Information in a computer system is composed of binary data and the interpretation of that data may change based on the context in which it is interpreted. The forensic examiner needs to provide assurance that the data is interpreted in the same way as it would be interpreted on the machine being investigated. Examiners also need to provide some assurance that the data presented are the same as the data that were originally collected from the machine, which is often performed by computing a cryptographic hash of the data, both as the data is originally collected, and

again after analyzing the data. This allows for detection of any changes that may have occurred within the data. The forensic examiner should strive to capture the state of the system as close to the time of the incident of interest as possible. The state of the machine is defined as everything that is happening in the machine, from the registers to the memory to the hard drive [7]. The examiner needs to collect as much of this state as possible. Ideally, the examiner wants to build a timeline of events that could have created that state, and then find the evidence that supports or refutes that timeline. Once a timeline is established, the examiner may use the evidence to demonstrate what is likely to have caused those events. This thesis is concerned with the forensically sound collection of evidence that accurately reflects the state of the system being investigated.

There are two schools of thought when it comes to performing forensics on a machine. One is to perform what is referred to as live analysis, examining running processes, open files, network connections, and the contents of RAM to obtain as much pertinent data as possible, including volatile information, on what is currently happening on the system, preferably as close to the time of the incident as possible. However, this method generally changes the state of the machine that is being analyzed because each action carried out by the forensic examiner typically requires the execution of new programs that often modify (generally appending to) log data, create and delete temporary files, etc. [1]. This is known as the observer effect, as the examiner observes the data they are also changing the state of the system. The observer effect can be worsened by the presence of malicious software, in which case the compromised machine cannot even be trusted to accurately reveal its state. This concept is similar to one from the field of quantum mechanics that is known as the Heisenberg uncertainty principle; where the act of measuring a particle changes the state of the particle and introduces uncertainty [8].

The other school of thought is that to preserve the data as close to the original state as possible, the machine should be shut off immediately and the examiner should only use available nonvolatile data during the analysis of the system. Many times this nonvolatile data can be enough to show what has happened on the machine. However, it will never give a complete picture of the machine state. When forensics is being

performed on a virtualized computer system, virtual machine introspection (VMI) can help to eliminate the observer effect by leveraging the power of the virtual machine monitor (virtual machine host) to gather required information from outside the virtual machine instance.

The value of live analysis is that the investigator can gain an insight into what is actually taking place in the system. They can run tools like ps, which provides a list of all currently running processes, or lsof, which provides a list of all files currently in use on the system. This and other data may provide important clues to the investigator and may possibly reveal the network connection an attacker has used to exploit the system or which files a user was viewing which could point to the motivation for the incident. Collecting an image of RAM will provide a more complete vision of what is going on in the computer. There are many tools that are designed to facilitate the analysis of RAM images such as Volatility [9] and IDETECT [10]. Given a RAM image alone, it is often possible to obtain a list of running processes, logged in users, as well as additional information about the running system.

B. VIRTUALIZATION

Virtualization is not new, but it has recently become more popular. In the late 1960s, mainframe computers were virtualized and rented out to smaller companies that could not afford an actual mainframe as a cost effective solution. It was impractical for many small companies to buy an expensive mainframe computer, but they still needed to use them, so other companies would create a virtual mainframe that they could rent out to the smaller companies for much cheaper than the cost of a full mainframe. In the 1980s and 1990s, with the advent of the personal computer (PC) and the drop in hardware costs, virtualization lost steam [11]. Recently however, virtualization has grown in popularity again, and desktop and server virtualization are common to see in organizations. Companies, such as VMware, have made virtualization a mainstream capability. Since VMware figured out how to do direct binary translation for x86 instructions which was previously thought to be impossible, virtual machines (VM) have appeared in more and

more places [12]. Virtualization has its beginnings in multiprocessing; one computer, one operating system, and many applications made way for one computer, many operating systems, and many applications [13].

There are two main types of virtual machine monitors (also called hypervisors): traditional sometimes called type one and, hosted or type two. The traditional type has a virtual machine monitor running directly on the hardware, and the virtual machines run on top of that, as shown in Figure 1. The hosted virtual machine runs on top of a virtual machine monitor (VMM) sometimes called a hypervisor installed in a host operating system [11]. This is illustrated in Figure 2.

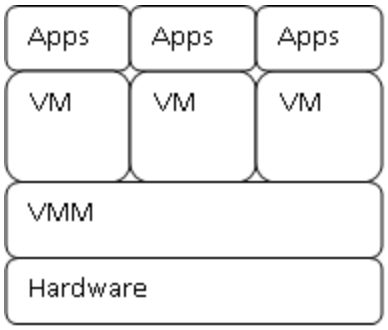

Figure 1. Type 1 VMM

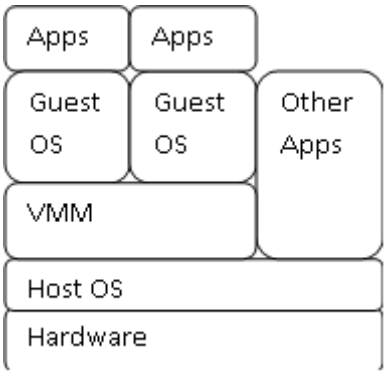

Figure 2. Type 2 VMM

The VMM is tasked with providing the interface between the virtual machine and either the host operating system or the hardware. It is also responsible for making sure the individual virtual machines remain logically separated from one another.

There are two main strategies when it comes to virtualization: full virtualization, in which the guest operating system is unmodified and the VMM takes care of the virtualization, and paravirtualization, in which the guest operating system has been modified to make some calls to the VMM instead of running directly on the hardware [12]. VMware started with using the full virtualization by doing binary translation of privileged instructions on the fly [12]. Xen started with making use of paravirtualization to get the operating system and the VMM to work together to provide the virtual interface [14]. Both companies are looking at the benefits of both techniques and are working to incorporate them into future products. Full virtualization allows easier portability and modularity by allowing any OS to be virtualized. Paravirtualization can offer significant performance improvements since it can eliminate some of the context switching between the VMM and the hardware or guest OS [11].

One of the unique aspects of Xen is that, strictly speaking, Xen is a type one virtualization, however Xen makes use of a single privileged virtual machine to manage additional, nonprivileged virtual machines [14]. Xen is unique in that one of the VMs is closely tied to the VMM in a manner similar to the way that a type two virtualized platform places a VMM above the host operating system in order to control nonprivileged virtual machines. Xen breaks VMs up into domains and calls the privileged VM Domain 0 or Dom0. Additional virtual machines are the user domains or DomUs. This privileged machine is unique to Xen's architecture. VMware does not implement its management from a privileged virtual machine.

Virtualization has many benefits. A company may run many virtualized servers on a single physical machine, and still have the benefit of keeping the services logically separated. With the continued growth of computing power, it is not uncommon for a server to have unused processing capacity available. In many situations, a server may only be using a small percentage of its available processing power at any given time; and by running multiple virtual machines a server can make more efficient use of its available processing power. Virtualization may also save companies space in their data centers by

reducing the number of physical boxes required to implement all of the company's computing needs. This is a major selling point in the push toward cloud computing and use of virtualization to provide infrastructure as a service [15].

The wide spread use of virtualization in data centers has created many concerns on the security front. Collocating many virtual machines on a single physical computer changes the security paradigm. In a nonvirtualized environment, if one server went down, then presumably one service went down. With the move towards highly virtualized operations, the loss of a single machine that is hosting multiple virtual servers could have a much greater impact [3]. One of the dangers associated with virtualization is that an exploit might make it possible for an attacker to escape from one VM and into the VMM itself. Once an attacker has control of the VMM, he effectively has control of every virtual machine being controlled by that VMM. A proof of concept for such a VM escape was demonstrated at Black Hat by Kostya Kortchinsky [16]. It may be worth noting that the techniques used in this paper may even open a virtual machine up to more risk of this type; this remains an open research question [17].

C. VIRTUAL MACHINE INTROSPECTION

The use of virtual machine monitors and hypervisors leads directly into virtual machine introspection, a term first coined by Garfinkel and Rosenblum [18]. VMI leverages the VMM's unique position as an entity external to the VM and yet able to observe all of the VM at a very low level. Garfinkel and Rosenblum utilized this fact to position an intrusion detection system (IDS) at the VMM. This positioning allowed the IDS to be in a place that it would not be as susceptible to an attack as many host based IDSs are, since it would be outside the host. It would also allow better visibility into what was actually happening on the host than would be available if the IDS was just on the network. This technique has since been extended to perform digital forensics on virtual machines [1]. Brian Hay et al. have developed a toolkit called Virtual Introspection for Xen (VIX). They have shown that this toolkit can be used to perform live analysis and collect volatile data from a virtual machine. Their technique involves pausing the virtual machine, collecting required data and then resuming the virtual

machine. That research showed that even a compromised machine with a rootkit that was hiding it's presence from traditional user space forensics tools running within the virtual machine could still be detected via the use of VMI techniques. This is because the tools that they developed look at the memory on the computer system and don't rely on system calls within the guest operating system to do the analysis. Since Garfinkel and Rosenblum's initial work, this topic has seen a lot of growth and research. Other researchers [1] have successfully used VMI to collect volatile data off a virtual machine. The tools they developed allow an investigator to run the tools from within a Dom0 host to perform observations of a DomU host. Since the Dom0 is running from a privileged domain, it can access all of the DomU's memory, this is what allows the virtual introspection to take place. The Dom0 can map the DomU's memory into its own memory space and then analyze that memory. This strategy works well when a DomU virtual machine that has been compromised. The investigator can examine it from the Dom0 machine and see all of its memory, including things that might be trying to hide themselves, such as a kernel level rootkit in the virtual machine. This is possible because the Dom0 can leverage the VMM to obtain a view of the virtual machine that is below the kernel running within the virtual machine. This key piece of research showed how virtual machine introspection can give a view that is incredibly helpful to forensic examiners, what remained to be shown was further explored in a paper written by Kara Nance et al. [17]. They looked at the implications of virtual machine introspection for digital forensics, and outlined some of the areas that need to be examined for the potential to become fully realized. The authors outlined four main areas of research to be done: the first is VMI tool development, the second is application to nonquiescent machines, the third is covert operations using virtual machine introspection, and the last is detection of virtual machine introspection from within a virtual machine. These areas will be expanded on in the future work section.

Another toolset that is similar to VIX is XenAccess developed and maintained by Bryan Payne [4]. XenAccess is designed to provide a library of functions to build a monitoring architecture. This incorporates virtual memory introspection to monitor

applications and access the memory state of the virtual machine. This architecture has been used to implement VMI, and with that, we can perform forensics on virtual machines.

III. DESIGN AND METHODOLOGY

A. ENVIRONMENT

Our experiments were performed using a Dell Optiplex 755 system with a core 2 duo processor and four GB of RAM. We installed CentOS 5.6 x86_64 with Linux kernel version number 2.6.18-238.9.1.el5 on the base machine [19]. We then replaced the kernel with the Xen kernel version 3.0.3-120.el5_6.2.x86_64. This was done by using the package manager in CentOS, the install required a reboot. We also edited the /boot/menu.lst to boot into the xen kernel by default. We then used the graphical virtual machine editor interface and installed a hardware assisted virtual machine with CentOS 5.6 x86 32 bit. The VM was a fully virtualized machine, we did not use the paravirtualization option, and this allowed us to install an unmodified operating system within a virtual machine. We kept both the host machine and the guest VM up to date with all recommended automatic patches as of June 2011.

We set up the virtual machine using the Virtual Machine Manager that was installed on CentOS. It allowed us to specify in a GUI what options we wanted to include in the virtual machine. We configured the DomU VM to have one GB of RAM and to only have one CPU allocated, while the Dom0 machine was allocated both CPUs. This is common among virtualization environments. Since the Dom0 machine could use an extra CPU, it could be looking into memory of the DomU while the DomU was currently running. Both machines could be running on separate cores depending on the scheduling.

The XenAccess package depends on the Xen development library; this was also downloaded and installed through the CentOS package manager by installing the package *xen-devel*. This package includes the header files that XenAccess depends on and uses to map memory and copy it from DomU to Dom0. We downloaded XenAccess version 0.5 by using a svn checkout of the code hosted at [20]. When that was done we ran the

autogen.sh script that comes with XenAccess and ran configure and make. This built, and compiled XenAccess on the Dom0. We then ran a make install as root to install the tools on the Dom0 by the tools are installed to the /usr/local/bin/ directory.

One of the key components to getting XenAccess to work is creating a configuration file for the DomU that is going to be examined. This file will provide the information that is necessary for XenAccess to examine the DomU's memory. According to the XenAccess documentation, "This file has a set of entries for each domain that XenAccess will access. These entries specify things such as the OS type (e.g., Linux or Windows), the location of symbolic information, and offsets used to access data within the domain" [21]. We obtained these values using a find offsets tool that is included in the XenAccess package. This tool is a loadable kernel module, *findoffsets.ko* that needs to be run from inside the DomU. Once loaded, the module determines the required offset information and logs the values via the syslog facility. The module is loaded using the *insmod* command after which the necessary offset information may be retrieved from the system log file. We then copied the values and unloaded the module. For our setup, the configuration file looked like this:

```
testhvm1 {
        ostype = "Linux";
        sysmap = "/boot/System.map-testhvm1";
#        linux_name = 0x194;
        linux_tasks = 0x7c;
        linux_mm = 0x84;
        linux_pid = 0xa8;
        linux_pgd = 0x28;
        linux_addr = 0x84;
}
```

In order for this configuration file to work, we had to copy the System.map-2.6.18-238.9.1.el5 from the DomU and place it in the Dom0 as /boot/System.map-testhvm1. The system map is a look-up table that maps kernel symbols to their respective locations in memory.

XenAccess includes six tools that can be used and may be applicable for forensics. The first is *dump-memory*, this tool maps out each page in the DomU's

memory space and copies it into a specified file. The second is *process-list*, this tool mimics the Unix *ps* utility and provides the rough equivalent of running *ps* –A within the VM. The third is *module-list*, this tool mimics the lsmod utility on the VM. The fourth is *map-addr*, this maps a specified address into the Dom0's address space to examine. The fifth is *map-symbol*, this provides a simple view of a specified kernel symbol. The final tool is *process-data*, this shows the userspace data of a specified process pid. [21] We used the first three tools, as we felt that these were representative of tasks that can be accomplished by using VMI, and Unix counterparts of these utilities are often useful in conducting live system analysis. The remaining three tools were more specific and we felt that the three higher level tools would better show the utility for forensics.

We logged into both the Dom0 and the DomU machines as root using ssh to run our experiments. The XenAccess toolkit requires root access to run.

B. METHODOLOGY

The questions that we wanted to answer were as follows: do the tools work in a nonquiescent environment? If so, can they give reliable results in that environment? What kind of performance do the tools give under differing loads on the VM? Do the tools cause any noticeable degradation in the processing on the VM?

To answer each of these questions, the following tests were designed and performed. To test if the tools work in a nonquiescent environment, the tools were first run on a paused machine and then run again on a nonquiescent machine in order to compare the results. To test how reliable they were in each environment, we compared the output from the VMI tools with their Unix counterparts that were run within the DomU machine. To test the performance of the tools, we ran tests under different loads on the VM and compared the average time taken to run over ten runs. To measure the degradation in performance happening on the VM, we timed the compiling of the kernel without running the tools from the Dom0 and then timed the compiling of the kernel while the tools were run from the Dom0. We also measured the average CPU load across the compile for both sets of tests.

To help minimize the variables with the experiments, the same saved state was loaded before each test. We ran each test ten times to validate and collect average times for each run. The results of these tests will be covered in the next chapter, and the results of each individual test will be available in the appendix. The tests were run over an ssh connection to the Dom0 machine and an ssh connection to the DomU machine for the nonquiescent tests against the DomU. On both machines, we logged in as root to run the commands.

Let us look a little more in depth at the tools that we chose to run for our experiments. The *memory-dump* tool is used to copy the entire memory space of the VM. It copies a page at a time from the DomU's memory space into the Dom0's memory space. This results in a snapshot of the memory of the DomU. *Process-list* makes use of the linux_pid value from the system map to walk through each process in the DomU's process list. *Module-list* performs in a manner similar to process list but iterates over the loaded Linux kernel modules instead of the process list. Linux kernel modules are additional components of the operating system that are dynamically loaded into the running kernel.

We tested each of these tools to observe how much having the VM in a paused state would affect the reliability of the tools. Specifically, we were interested in determining whether the tools would only be able to collect useable data when the VM was in a quiescent state, or whether the same data be obtained while in a nonquiescent state? Following the procedure outlined earlier, the tests that we performed were to pause the VM, and then run the VMI process list, module list, and memory dump. This would give us a baseline with which to compare the other results. We would also test the tools while in the nonquiescent state. Since the performance of the tools could vary based on the loading of the system being examined, we tested the tools while the machine was in a state close to idle (i.e., just background processes running), and while the machine was heavily loaded (i.e., compiling the Linux kernel). The tools provided us with a dump of memory, a list of processes, and a list of loaded kernel modules. Additional processing of the memory dump for forensic purposes was out of scope for

this research. A number of programs are capable of analyzing memory dumps, such as Volatility [9] or IDETECT [10]. These tools, and other methods, have been documented in other papers [22], [23].

We established a baseline snapshot of the DomU virtual machine that was loaded prior to each test. We booted up the system and then used the save state menu option to save this "start" state. We then rolled back the DomU state using this saved snapshot for each subsequent test. This allowed us to have some assurance that each test was being performed on a nearly identical system. This will also allow for repeatability of the experiments.

Before testing the tools in a nonquiescent environment, we ran the tools to make sure they worked while the machine was paused to obtain a baseline dataset to compare to our results. We started up the machine using the saved snapshot then logged in and launched the same processes that we would be using for the nonquiescent tests; however, we paused the machine before running any analysis tools from Dom0. Using the tools, we obtained a memory dump, a process list, and a module list. The results we obtained from the tools that were run while the DomU VM was paused confirmed the results of previous work, [1] namely that VMI can be performed on a paused VM. We also reverted to our snapshot and ran the equivalent Unix process list and module list tools inside the VM. This allowed us to compare the results obtained from the VMI tools running against a paused VM with the results of tools running within the actual VM. The results were the same (excepting the presence of the ps process itself which was to be expected). We did not repeat the work done in [1] in which they test the ability for VMI to detect the presence of a rootkit that caused the results of the ps process run from inside the VM and the ps VMI utility to be different.

We decided to have the virtual machine in a state close to idle for our first set of experiments. Since there would be little activity on the machine, we expected little variation in the observed process list. In this low utilization state, the CPU utilization according to the System Monitor was approximately two percent. Having a light load minimized the number of changes that were taking place in RAM.

For the next set of tests, we tasked the virtual machine with compiling the linux kernel to simulate a heavier load on the virtual machine and then perform the same tests and observe the results. We also ran the *top* utility in batch mode to collect the average CPU load throughout the compile. We used a script to start top, time the make of the kernel and then kill top. After averaging the CPU load across all ten runs of the experiment, we found that when we did not run the VMI the average CPU utilization over the compile of the kernel was 69.4%. When we did run the VMI the average CPU usage was 68.6%. We attribute this change to the Dom0 using the CPU and will further investigate this in the results section. During this experiment, we also measured how long it took to compile the kernel, we did this with the *time* command.

We also timed how long the VMI operations required to complete from the Dom0 during each test that we ran. This was to measure how the tools performed and how the load on the system affected that performance. The results of this will be presented in the results section.

The goal of creating a lightly loaded test case and heavily loaded test case was to observe whether the XenAccess tools work just as well in each case, or whether there are major problems with attempting to use VMI on a nonquiescent VM. We were also able to measure the performance of the tools this way. We recognize that there are many different types of load on a system, we took the generic case of high CPU utilization, but we could have tested it with a different type of load.

We wanted to keep in mind that with a forensic examination the integrity of the data is of the upmost importance. If it can be shown that not pausing the virtual machine does not affect the collection of data by means of virtual machine introspection, then it can potentially speed up the collection process. It would also allow VMI to be more useable for the purposes of system monitoring as outlined in [4].

XenAccess uses the Xen control library to provide access to the memory pages within the specified VM's address space [4]. XenAccess provides an API to be used by

programs to provide memory introspection. It should be pointed out that while this thesis focused on CentOS, the techniques and programs run could be used to analyze any type of VM, thanks to the generic nature of the XenAccess API [4].

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS AND CONCLUSION

A. RESULTS

With the testing environment configured as outlined in the previous section, we began our experiments by testing a lightly loaded system both in a quiescent state and a nonquiescent state. After running the tests while the virtual machine was paused, we obtained a baseline for how the tests should go. The baseline tests consisted of collecting a memory dump, a process list, and a module list from the paused DomU VM. This data was then compared against the output of corresponding Unix utilities executed inside the virtual machine. The process list and module list were found to match (with the exception of the ps process appearing in the process list collected within the running DomU). The full listing of the processes and modules is included in the Appendix. This demonstrated that while the machine was paused the XenAccess tools can obtain useful forensic data.

For the tests that were conducted on a lightly loaded system, the average CPU load for the duration of the light load test was approximately 2% according to the activity monitor that we had running on the virtual machine. This test was initiated from the same snapshot VM state used in all of the testing. For this test, the same data (memory, process list, module list) was collected from the DomU. For each of the ten runs of the test, the process list and module list matched exactly the process list and module list collected from the paused test. We tested the difference in the data collected by using the *diff* program; this allowed us to see the locations in the data that differed.

Since the Dom0 maps out a page of memory by copying the page from the DomU's address space and placing it into its own address space, the page may currently be in use by the VM, so it is possible that the page changes in the VM right after it is copied into Dom0 so that the pages in Dom0 and DomU become different. This change is negligible from our perspective because the Dom0 page represents a moment in time snapshot of the corresponding DomU page. In the worst case scenario, it is possible that each DomU page is changed directly after it is copied. Even in this worst case scenario,

useable forensic data is obtained; however, this scenario is extremely unlikely. The same issues exist when performing live forensics analysis on a physical machine, and this has been deemed an acceptable risk in many cases [24].

The second test involved a heavily loaded system. To generate a heavy load, we started compiling a Linux kernel within the DomU VM and reran the tests. We ran a baseline test of compiling the Linux kernel without performing any VMI. The average results of the time it took to compile are listed in Table 1. We again performed the tests for ten runs on both a paused system and while the system was still running. The results differed between the paused test and the running test; however, the results differed because it was not possible for us to capture the exact same moment during the execution of the process. The differences in the process list reflected that there were different make and gcc processes running which reflects the state of the machine, so it can be shown that although these results did not match each other the results that were collected from the running machine still likely reflected the state of the machine at the time they were run.

	Real Time	User Time	System Time
Average	19:25.756	13:31.953	04:50.774
max	19:57.014	13:33.519	04:55.958
min	19:00.883	13:30.209	04:44.206
Standard deviation	00:16.998	00:01.080	00:03.321

Table 1. Time to compile kernel without VMI

For the heavily loaded system, we ran top during the compile. This allowed us to collect data on the CPU utilization while the kernel was compiling. When we did not run the VMI utilities, the average CPU usage over the ten runs was 69.4%. The average CPU usage when we performed the VMI over the ten runs was 68.6%. This difference is likely due to the contention on the Dom0 trying to use the CPU. Since the Dom0 has access to both CPUs and the DomU only has access to one, things running on the Dom0 could kick the DomU process out. We observed that during the top trace of the process, the CPU utilization drops significantly during the time when the actual VMI memory dump is taking place. This could be due to caching issues or the VMI process running on

the Dom0. It could also be that while the DomU system was blocking for I/O it was blocked longer than it would have been otherwise, and this has driven down the CPU utilization rates.

It is also the case here that since there are many changes happening in RAM, and data is being modified and moved around, the snapshot of memory will be different if done multiple times. This is the same behavior that we would see if we were to pause the VM and take a snapshot, then unpause, and allow processing to continue for a brief time and then pause and take another snapshot.

We looked at the time that it took the kernel to compile both while we were not doing VMI and how long it took for it to compile while doing VMI at the same time. This was done using the *time* command. The results are given in Table 1 and 2. This shows, on average, there is a potentially noticeable increase in time taken when the VMI is performed while the kernel is compiling, however this difference is still eclipsed by the length of the compile. This increase in time could potentially be exploited by software running inside the VM and be used to detect if VMI is being performed. We hypothesize that this increase in time is due to the cache being flushed on the processor, since during VMI each page is copied and mapped into the Dom0 and since Dom0 can control both processors, it is likely that all of that information is flushing out the cache. This flushing out of cache could provide a more consistent snapshot of memory since the information in cache will be written out to memory as it is replaced by the Dom0 process information. We leave the testing of this hypothesis to future work. It should also be noted, however, that the increase in time taken does not necessarily mean that VMI is happening; it could just mean that the Dom0 machine is experiencing a larger load. It could be that any large process on the Dom0 would cause a similar reaction in the DomU. More research is necessary before conclusions can be drawn on this; we recommend that experiments be performed that compare similar length processes.

	Real Time	User Time	System Time
Average	19:45.538	13:33.098	04:51.309
max	19:54.454	13:34.288	04:58.869
min	19:37.980	13:31.768	04:47.969
Standard deviation	00:04.738	00:00.898	00:03.347

Table 2. Time to compile kernel with VMI

We also tested how long the VMI memory dump took under the different loads. The results of this test are shown in Figure 3. The average time it took for the memory dump on a quiescent machine was 6.975 seconds. The average time it took on a machine with a light load was 7.021 seconds. On a heavy loaded system, it took on average 9.068 seconds. This is a substantial difference. This difference is most likely due to the load on the CPU and the Dom0 has competition for the other CPU during the Heavy load, whereas for the light load and for the paused machine there are not many other processes to compete with it.

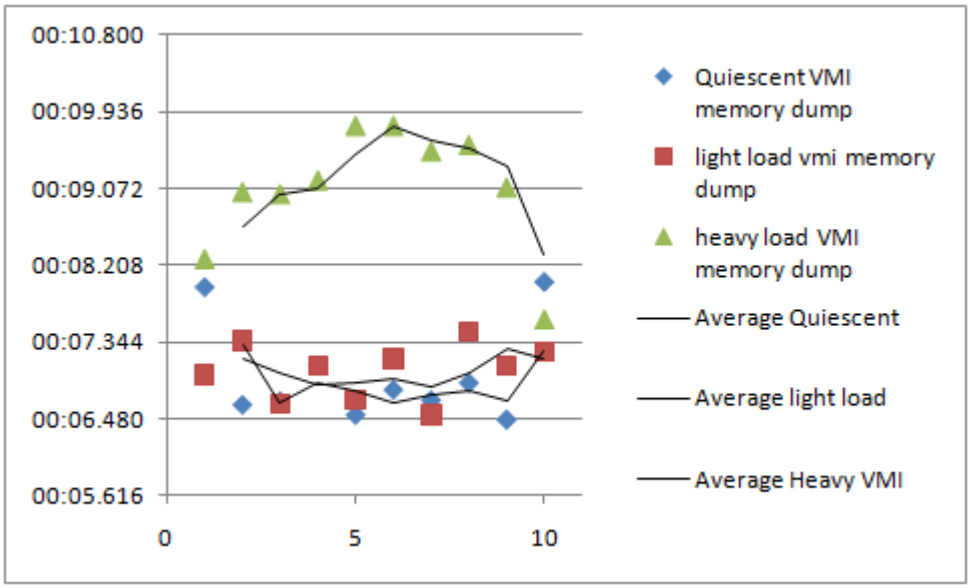

Figure 3. Time taken for VMI memory dump utility to run with different loads

Based on these results, it appears that using VMI on a running VM does not inhibit the opportunity to perform forensics on the VM.

B. CONCLUSION

Our experimental results suggest that forensics may be performed on a nonquiescent virtual machine with what appears to be the same quality as on a paused VM. That being said, when there is an incident, the argument for allowing the system being investigated to continue running should be a compelling one as it almost certainly best to pause the virtual machine in order to contain the damage and stop the intrusion or incident. That may not be the goal of the investigation, and it may be to observe and document the intrusion along the way, in which case pausing the VM would not be the best action. Our results show that useful data can then be collected even while the VM is under heavy load.

VMI on a nonquiescent VM is very similar to doing live analysis on a physical machine. There will be the same sort of issues with collecting a memory dump while a machine is running whether it is virtual or not. With VMI, however, obtaining the data may be less observable than using the tools installed within the VM itself, and so will help to mitigate the observer effect of live analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RELATED AND FUTURE WORK

A. RELATED WORK

There is some related research going on in virtual machine introspection, and much research that can still be done. For this project, we initially looked at using vprobes by vmware [25]. This product is advertised as a stethoscope for your virtual machine. It is geared towards problems that may be occurring within the virtualization layer and investigating those problems to troubleshoot issues in the vmm while running. While this is very similar to what we are doing, it is, fundamentally, a different problem set. Since VMware products are proprietary software, their source code is unavailable, and we were not able to adapt vprobes to our purposes and chose to investigate other options. Since Xen is open source, it is much easier to adapt the tools to work in the required environment. With some work, vprobes could probably be changed to allow the collection of forensics data.

Another project that focuses on Xen is from a group at the University of Alaska called Virtual Introspection for Xen or VIX [1]. This project was developed to be able to perform virtual machine introspection in Xen much like XenAccess does. It works by pausing the virtual machine and then collecting the data in a similar way as XenAccess. We were unable to test its applications to nonquiescent virtual machines but hypothesize that it would yield similar results as XenAccess.

On the line of the VMI tool development research recommended by [17], a new paper has been presented on Virtuoso [26] that presents an easier way to develop tools that can use VMI by first writing the tool in the guest, and then it can automatically port it to a VMI tool.

B. FUTURE WORK

This thesis focused on Xen, but there are many other virtualization platforms available. We hypothesize that these concepts will carry over to other platforms, but we leave the testing of this hypothesis to future work.

There is also much work that can be done in virtual machine introspection. As mentioned earlier: tool development, detection of VMI, and the use of VMI for covert operations, such as implementing a honey pot type environment [27], and monitoring it using VMI, are some of the areas that can be further explored. The tools in this area are not as mature as they need to be for wide use. While the XenAccess toolkit is built with the idea that it will be expanded and brought to maturity, this has not happened yet, and although releasing it as an open source project has fostered some development, more can be done, such as expanding the toolset to provide more utilities to the investigator.

In the nonquiescent environment, there are more experiments we would have liked to perform that time did not allow. We would have liked to be able to test other operating systems, especially a windows environment. The XenAccess tools should allow for an easy migration to testing other environments. We were able to test the selected tools under a high CPU load, but it would have been interesting to see how the tools performed under other types of loads also, such as an I/O bound process. We were also able to test how the tools changed the processing time for compiling the kernel, it would be interesting to see how much the VMI affected a process whose processing time was closer to the time that the VMI process itself took. This would provide a better understanding of just how noticeable the VMI process is from within the VM.

There are also implications for being able to detect VMI from within the VM. As we mentioned in the results, we observed that the process took longer and had less CPU utilization on average over the course of the process. It is probably the case that this would only be indicative of the fact that virtualization is happening, but it remains to be seen how much VMI affects performance on the VM; especially when the VMI is done on a nonquiescent machine. The use of tools, such as Xen Access, could affect the ability of the virtual machine to detect the presence of virtual machine introspection, but that is outside the scope of this thesis and is left to future work.

APPENDIX

Selected output from the experiments.

A. OUTPUT FROM PAUSED PS LIST

[1] init
[2] migration/0
[3] ksoftirqd/0
[4] watchdog/0
[5] events/0
[6] khelper
[7] kthread
[10] kblockd/0
[11] kacpid
[57] cqueue/0
[60] khubd
[62] kseriod
[125] khungtaskd
[126] pdflush
[127] pdflush
[128] kswapd0
[129] aio/0
[284] kpsmoused
[308] ata/0
[309] ata_aux
[314] kstriped
[323] ksnapd
[334] kjournald
[365] kauditd
[398] udevd
[639] xenwatch
[640] xenbus
[1162] kmpathd/0
[1163] kmpath_handlerd
[1220] kjournald
[1403] iscsi_eh
[1442] ib_addr
[1449] ib_mcast
[1450] ib_inform
[1451] local_sa
[1454] iw_cm_wq
[1457] ib_cm/0
[1460] rdma_cm

[1476] brcm_iscsiuio
[1481] iscsid
[1482] iscsid
[1568] mcstransd
[1809] dhclient
[1861] auditd
[1863] audispd
[1885] restorecond
[1899] syslogd
[1902] klogd
[1992] portmap
[2023] rpciod/0
[2029] rpc.statd
[2063] rpc.idmapd
[2088] dbus-daemon
[2103] hcid
[2107] sdpd
[2126] krfcommd
[2173] pcscd
[2188] acpid
[2207] hald
[2208] hald-runner
[2215] hald-addon-acpi
[2220] hald-addon-keyb
[2229] hald-addon-stor
[2250] hidd
[2293] automount
[2315] hpiod
[2320] hpssd.py
[2340] sshd
[2356] cupsd
[2387] sendmail
[2398] sendmail
[2413] gpm
[2427] crond
[2461] xfs
[2488] atd
[2530] avahi-daemon
[2531] avahi-daemon
[2566] smartd
[2570] mingetty
[2571] mingetty
[2574] mingetty
[2577] mingetty
[2578] mingetty

[2579] mingetty
[2590] gdm-binary
[2656] gdm-binary
[2658] gdm-rh-security
[2661] Xorg
[2673] yum-updatesd
[2675] gam_server
[2686] gnome-session
[2722] ssh-agent
[2751] dbus-launch
[2752] dbus-daemon
[2759] gconfd-2
[2762] gnome-keyring-d
[2764] gnome-settings-
[2780] xrdb
[2784] metacity
[2788] gnome-panel
[2790] nautilus
[2794] bonobo-activati
[2797] gnome-vfs-daemo
[2799] eggcups
[2800] gnome-volume-ma
[2809] bt-applet
[2824] puplet
[2828] nm-applet
[2830] wnck-applet
[2832] trashapplet
[2835] pam-panel-icon
[2836] pam_timestamp_c
[2838] gnome-power-man
[2840] escd
[2852] nm-system-setti
[2854] mixer_applet2
[2857] mapping-daemon
[2858] yum-updatesd-he
[2862] notification-ar
[2864] clock-applet
[2887] gnome-terminal
[2889] gnome-pty-helpe
[2890] bash
[2908] gnome-system-mo
[2911] gnome-screensav

B. OUTPUT FROM LIGHT LOAD PS LIST

[1] init
[2] migration/0
[3] ksoftirqd/0
[4] watchdog/0
[5] events/0
[6] khelper
[7] kthread
[10] kblockd/0
[11] kacpid
[57] cqueue/0
[60] khubd
[62] kseriod
[125] khungtaskd
[126] pdflush
[127] pdflush
[128] kswapd0
[129] aio/0
[284] kpsmoused
[308] ata/0
[309] ata_aux
[314] kstriped
[323] ksnapd
[334] kjournald
[365] kauditd
[398] udevd
[639] xenwatch
[640] xenbus
[1162] kmpathd/0
[1163] kmpath_handlerd
[1220] kjournald
[1403] iscsi_eh
[1442] ib_addr
[1449] ib_mcast
[1450] ib_inform
[1451] local_sa
[1454] iw_cm_wq
[1457] ib_cm/0
[1460] rdma_cm
[1476] brcm_iscsiuio
[1481] iscsid
[1482] iscsid
[1568] mcstransd
[1809] dhclient
[1861] auditd

[1863] audispd
[1885] restorecond
[1899] syslogd
[1902] klogd
[1992] portmap
[2023] rpciod/0
[2029] rpc.statd
[2063] rpc.idmapd
[2088] dbus-daemon
[2103] hcid
[2107] sdpd
[2126] krfcommd
[2173] pcscd
[2188] acpid
[2207] hald
[2208] hald-runner
[2215] hald-addon-acpi
[2220] hald-addon-keyb
[2229] hald-addon-stor
[2250] hidd
[2293] automount
[2315] hpiod
[2320] hpssd.py
[2340] sshd
[2356] cupsd
[2387] sendmail
[2398] sendmail
[2413] gpm
[2427] crond
[2461] xfs
[2488] atd
[2530] avahi-daemon
[2531] avahi-daemon
[2566] smartd
[2570] mingetty
[2571] mingetty
[2574] mingetty
[2577] mingetty
[2578] mingetty
[2579] mingetty
[2590] gdm-binary
[2656] gdm-binary
[2658] gdm-rh-security
[2661] Xorg
[2673] yum-updatesd

[2675] gam_server
[2686] gnome-session
[2722] ssh-agent
[2751] dbus-launch
[2752] dbus-daemon
[2759] gconfd-2
[2762] gnome-keyring-d
[2764] gnome-settings-
[2780] xrdb
[2784] metacity
[2788] gnome-panel
[2790] nautilus
[2794] bonobo-activati
[2797] gnome-vfs-daemo
[2799] eggcups
[2800] gnome-volume-ma
[2809] bt-applet
[2824] puplet
[2828] nm-applet
[2830] wnck-applet
[2832] trashapplet
[2835] pam-panel-icon
[2836] pam_timestamp_c
[2838] gnome-power-man
[2840] escd
[2852] nm-system-setti
[2854] mixer_applet2
[2857] mapping-daemon
[2858] yum-updatesd-he
[2862] notification-ar
[2864] clock-applet
[2887] gnome-terminal
[2889] gnome-pty-helpe
[2890] bash
[2908] gnome-system-mo
[2911] gnome-screensav
[3560] sshd
[3564] bash

C. EXAMPLE OUTPUT FROM HEAVY LOAD PS LIST

[1] init
[2] migration/0
[3] ksoftirqd/0
[4] watchdog/0
[5] events/0

[6] khelper
[7] kthread
[10] kblockd/0
[11] kacpid
[57] cqueue/0
[60] khubd
[62] kseriod
[125] khungtaskd
[126] pdflush
[127] pdflush
[128] kswapd0
[129] aio/0
[284] kpsmoused
[308] ata/0
[309] ata_aux
[314] kstriped
[323] ksnapd
[334] kjournald
[365] kauditd
[398] udevd
[639] xenwatch
[640] xenbus
[1162] kmpathd/0
[1163] kmpath_handlerd
[1220] kjournald
[1403] iscsi_eh
[1442] ib_addr
[1449] ib_mcast
[1450] ib_inform
[1451] local_sa
[1454] iw_cm_wq
[1457] ib_cm/0
[1460] rdma_cm
[1476] brcm_iscsiuio
[1481] iscsid
[1482] iscsid
[1568] mcstransd
[1809] dhclient
[1861] auditd
[1863] audispd
[1885] restorecond
[1899] syslogd
[1902] klogd
[1992] portmap
[2023] rpciod/0

[2029] rpc.statd
[2063] rpc.idmapd
[2088] dbus-daemon
[2103] hcid
[2107] sdpd
[2126] krfcommd
[2173] pcscd
[2188] acpid
[2207] hald
[2208] hald-runner
[2215] hald-addon-acpi
[2220] hald-addon-keyb
[2229] hald-addon-stor
[2250] hidd
[2293] automount
[2315] hpiod
[2320] hpssd.py
[2340] sshd
[2356] cupsd
[2387] sendmail
[2398] sendmail
[2413] gpm
[2427] crond
[2461] xfs
[2488] atd
[2530] avahi-daemon
[2531] avahi-daemon
[2566] smartd
[2570] mingetty
[2571] mingetty
[2574] mingetty
[2577] mingetty
[2578] mingetty
[2579] mingetty
[2590] gdm-binary
[2656] gdm-binary
[2658] gdm-rh-security
[2661] Xorg
[2673] yum-updatesd
[2675] gam_server
[2686] gnome-session
[2722] ssh-agent
[2751] dbus-launch
[2752] dbus-daemon
[2759] gconfd-2

[2762] gnome-keyring-d
[2764] gnome-settings-
[2780] xrdb
[2784] metacity
[2788] gnome-panel
[2790] nautilus
[2794] bonobo-activati
[2797] gnome-vfs-daemo
[2799] eggcups
[2800] gnome-volume-ma
[2809] bt-applet
[2824] puplet
[2828] nm-applet
[2830] wnck-applet
[2832] trashapplet
[2835] pam-panel-icon
[2836] pam_timestamp_c
[2838] gnome-power-man
[2840] escd
[2852] nm-system-setti
[2854] mixer_applet2
[2857] mapping-daemon
[2858] yum-updatesd-he
[2862] notification-ar
[2864] clock-applet
[2887] gnome-terminal
[2889] gnome-pty-helpe
[2890] bash
[2908] gnome-system-mo
[2911] gnome-screensav
[3560] sshd
[3564] bash
[3594] top
[3595] make
[6732] make
[7408] make
[7532] sh
[7533] gcc
[7534] cc1
[7535] as

D. MODULE LISTING FROM THE VM (SAME FOR EACH TEST)

autofs4
hidp
rfcomm

l2cap
Bluetooth
lockd
sunrpc
ip_conntrack_netbios_ns
ipt_REJECT
xt_state
ip_conntrack
nfnetlink
xt_tcpudp
iptable_filter
ip_tables
ip6_tables
x_tables
be2iscsi
ib_iser
rdma_cm
ib_cm
iw_cm
ib_sa
ib_mad
ib_core
ib_addr
iscsi_tcp
bnx2i
cnic
ipv6
xfrm_nalgo
crypto_api
uio
cxgb3i
cxgb3
8021q
libiscsi_tcp
libiscsi2
scsi_transport_iscsi2
scsi_transport_iscsi
loop
dm_multipath
scsi_dh
video
backlight
sbs
power_meter
hwmon

i2c_ec
dell_wmi
wmi
button
battery
asus_acpi
ac
parport_pc
lp
parport
floppy
xen_vnif
xen_balloon
xen_vbd
pcspkr
i2c_piix4
i2c_core
xen_platform_pci
8139too
8139cp
mii
ide_cd
cdrom
serio_raw
tpm_tis
tpm
tpm_bios
dm_raid45
dm_message
dm_region_hash
dm_mem_cache
dm_snapshot
dm_zero
dm_mirror
dm_log
dm_mod
ata_piix
libata
sd_mod
scsi_mod
ext3
jbd
uhci_hcd
ohci_hcd
ehci_hcd

E. SCRIPT USED TO START HEAVY LOAD TESTS

```
#!/bin/sh

top –b –i > cpu_stats.txt &
cd /tmp/linux-2.6.18
time make
pkill -9 top
```

F. TABLES OF TIMING RESULTS

Run	Real time	User time	System time
1	19:09.727	13:30.209	04:47.802
2	19:22.446	13:31.628	04:53.119
3	19:26.606	13:32.612	04:51.577
4	19:19.343	13:32.086	04:48.875
5	19:24.444	13:32.251	04:50.683
6	19:27.535	13:31.921	04:52.675
7	19:51.013	13:33.223	04:49.663
8	19:00.883	13:30.309	04:44.206
9	19:18.545	13:31.769	04:53.184
10	19:57.014	13:33.519	04:55.958
Average	19:25.756	13:31.953	04:50.774
max	19:57.014	13:33.519	04:55.958
min	19:00.883	13:30.209	04:44.206
stddev	00:16.998	00:01.080	00:03.321

Table 3. Time for make without VMI

Run	Real time	User time	System time
1	19:37.980	13:31.768	04:47.969
2	19:45.182	13:33.126	04:47.978
3	19:42.589	13:31.872	04:49.999
4	19:47.298	13:33.748	04:54.179
5	19:41.504	13:33.380	04:48.171
6	19:50.110	13:33.487	04:52.151
7	19:42.522	13:32.383	04:51.950
8	19:54.454	13:34.218	04:58.869
9	19:45.519	13:32.713	04:51.213
10	19:48.221	13:34.288	04:50.613
Average	19:45.538	13:33.098	04:51.309

max	19:54.454	13:34.288	04:58.869
min	19:37.980	13:31.768	04:47.969
stddev	00:04.738	00:00.898	00:03.347

Table 4. Time for make with VMI

Run	Real time	User time	System time
1	00:08.273	00:00.660	00:04.804
2	00:09.024	00:00.692	00:04.468
3	00:09.000	00:00.684	00:04.908
4	00:09.156	00:00.864	00:04.652
5	00:09.766	00:00.692	00:04.980
6	00:09.766	00:00.980	00:04.840
7	00:09.478	00:00.860	00:04.708
8	00:09.550	00:00.884	00:04.932
9	00:09.073	00:00.824	00:04.712
10	00:07.595	00:00.628	00:05.012
Average	00:09.068	00:00.777	00:04.802
max	00:09.766	00:00.980	00:05.012
min	00:07.595	00:00.628	00:04.468
stddev	00:00.682	00:00.119	00:00.169

Table 5. Time for VMI memory dump on heavy load

Run	Real time	User time	System time
1	00:06.974	00:00.644	00:05.084
2	00:07.353	00:00.692	00:05.140
3	00:06.650	00:00.656	00:05.144
4	00:07.082	00:00.756	00:04.964
5	00:06.695	00:00.756	00:05.028
6	00:07.157	00:00.648	00:05.312
7	00:06.524	00:00.760	00:05.036
8	00:07.464	00:00.536	00:05.192
9	00:07.082	00:00.780	00:05.204
10	00:07.232	00:00.828	00:04.984
Average	00:07.021	00:00.706	00:05.109
max	00:07.464	00:00.828	00:05.312
min	00:06.524	00:00.536	00:04.964
stddev	00:00.311	00:00.086	00:00.110

Table 6. Time for VMI memory dump on light load

Run	Real time	User time	System time
1	00:07.968	00:00.652	00:05.032
2	00:06.635	00:00.688	00:05.080
3	00:06.674	00:00.688	00:05.116
4	00:07.082	00:00.756	00:04.964
5	00:06.521	00:00.804	00:04.960
6	00:06.805	00:00.748	00:05.004
7	00:06.688	00:00.740	00:05.096
8	00:06.884	00:00.640	00:05.180
9	00:06.467	00:00.692	00:04.880
10	00:08.027	00:00.720	00:05.184
Average	00:06.975	00:00.713	00:05.050
max	00:08.027	00:00.804	00:05.184
min	00:06.467	00:00.640	00:04.880
stddev	00:00.567	00:00.050	00:00.099

Table 7. Time for VMI memory dump on quiescent light load

Run	Real time	User time	System time
1	00:06.735	00:00.596	00:05.128
2	00:06.379	00:00.644	00:05.044
3	00:06.419	00:00.736	00:04.792
4	00:06.407	00:00.784	00:05.016
5	00:06.868	00:00.660	00:05.128
6	00:06.901	00:00.604	00:05.160
7	00:06.617	00:00.728	00:05.032
8	00:07.057	00:00.512	00:05.292
9	00:06.727	00:00.552	00:05.224
10	00:06.160	00:00.560	00:05.024
Average	00:06.627	00:00.638	00:05.084
max	00:07.057	00:00.784	00:05.292
min	00:06.160	00:00.512	00:04.792
stddev	00:00.281	00:00.089	00:00.138

Table 8. Time for VMI memory dump on quiescent heavy load

LIST OF REFERENCES

[1] B. Hay and K. Nance, "Forensics examination of volatile system data using virtual introspection," *SIGOPS Oper. Syst. Rev.*, Vol. 42, Apr. 2008, pp. 74–82.

[2] D. Bem and E. Huebner, "Computer forensic analysis in a virtual environment," *International Journal of Digital Evidence*, Vol. 6, 2007.

[3] D. Barrett and G. Kipper, *Virtualization and Forensics: A Digital Forensic Investigator's Guide to Virtual Environments*, Syngress, 2010.

[4] B. D. Payne, M. D. P. de Carbone, and W. Lee, "Secure and Flexible Monitoring of Virtual Machines," *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 2007, pp. 385–397.

[5] S. Peisert, S. Karin, M. Bishop, and K. Marzullo, "Principles-driven forensic analysis," *In Proc. of the 2005 workshop on New security paradigms*, 2005, pp. 85-93.

[6] B. Nelson, A. Phillips, F. Enfinger, C. Steuart, *Guide to Computer Forensics and Investigations.* Canada: Thomson Learning, 2004.

[7] J. Pfoh, C. Schneider, C. Eckert, "A formal model for virtual machine introspection," *In Proceedings of the 1st ACM workshop on Virtual machine security*, 2009, pp. 1–10.

[8] J. A. Wheeler and H. Zurek, *Quantum Theory and Measurement* Princeton Univ. Press, 1983, pp. 62–84.

[9] Volatile Systems, "Volatility | Memory Forensics | Volatile Systems." May 2011, https://www.volatilesystems.com/default/volatility.

[10] M. Burdach, "Digital forensics of the physical memory," *Warsaw University*, 2005.

[11] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, May. 2005, pp. 39–47.

[12] VMware, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist," VMware, Nov. 10, 2007

[13] K. Nance, M. Bishop, and B. Hay, "Virtual Machine Introspection: Observation or Interference?," *Security & Privacy, IEEE*, vol. 6, 2008, pp. 32–37.

[14] Xen, "What is Xen Hypervisor?" Xen, May 2011, http://www.xen.org/files/Marketing/WhatisXen.pdf.

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, and M. Zaharia, *Above the Clouds: A Berkeley View of Cloud Computing*, 2009.

[16] K. Kortchinsky, "Cloudburst, a VMWare Guest to Host Escape Story," Black Hat USA, 2009

[17] K. Nance, B. Hay, and M. Bishop, "Investigating the Implications of Virtual Machine Introspection for Digital Forensics," *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, 2009, pp. 1024–1029.

[18] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," *In Proc. Network and Distributed Systems Security Symposium*, 2003, pp. 191–206.

[19] CentOS, "The Community ENTerprise Operating System," CentOS, May 2011, http://www.centos.org.

[20] XenAccess, "project source," http://xenaccess.googlecode.com/svn/trunk/.

[21] XenAccess, "XenAccess: Main Page," http://doc.xenaccess.org/.

[22] J. Urrea, "An analysis of Linux RAM forensics," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2006.

[23] J. Schultz, "Offline forensic analysis of Microsoft Windows XP physical memory," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2006.

[24] B. D. Carrier, and J. Grand, "A hardware-based memory acquisition procedure for digital investigations," *Digital Investigation*, 2004, pp. 50–60.

[25] VMware, *VProbes Programming Reference*, VMware, 2010

[26] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection," *In Proc. IEEE Symposium on Security and Privacy*, 2011, pp. 297–312.

[27] The Honeynet Project, *Know Your Enemy*. Boston: Addison-Wesley, 2002.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California