Forensics Examination of Volatile System Data Using Virtual Introspection

Brian Hay Kara Nance
Department of Computer Science
University of Alaska Fairbanks
Fairbanks, AK 99775-6670
{brian.hay, ffkln}@uaf.edu

ABSTRACT
While static examination of computer systems is an important part of many digital forensics investigations, there are often important system properties present only in volatile memory that cannot be effectively recovered using static analysis techniques, such as offline hard disk acquisition and analysis. An alternative approach, involving the live analysis of target systems to uncover this volatile data, presents significant risks and challenges to forensic investigators as observation techniques are generally intrusive and can affect the system being observed. This paper provides a discussion of live digital forensics analysis through virtual introspection and presents a suite of virtual introspection tools developed for Xen (VIX tools). The VIX tools suite can be used for unobtrusive digital forensic examination of volatile system data in virtual machines, and addresses a key research area identified in the virtualization in digital forensics research agenda [22].

Categories and Subject Descriptors
D.4.8 [Operating Systems]: Performance – *Monitor*; D.4.6. [Operating Systems]: Security and Protection

General Terms
Management, Experimentation, Security, Verification.

Keywords
Virtual Machine Monitor, VIX, Digital Forensics, Virtual Introspection.

1. INTRODUCTION
As computer systems become increasingly ubiquitous and complex, digital forensic techniques must necessarily evolve to address the associated new challenges being faced by digital forensics examiners during incident response. The ability to determine how defenses failed, and to what extent the affected systems have been compromised is critical to recovering from an incident, and ensuring that systems are more resilient to attack in the future. This can be accomplished through live forensic analysis of volatile system data using virtual introspection.

2. BACKGROUND
Traditional static analysis techniques, where images of storage media are examined using tools such as Encase [9] or FTK [1], have the ability to unearth tremendous volumes of data including electronic documents, browsing history, email records, installed programs, and even files that users believe they have deleted. This approach typically involves powering the target system down and creating a forensically valid copy of the digital media, using write blocking devices where necessary to ensure that the image acquisition process does not modify the original media. While a wealth of information can be gathered using this approach, there are many system properties, such as the process list, open network ports, installed kernel modules, and volatile memory contents, which cannot be retained and examined as part of a forensics analysis using these techniques. As systems become increasingly reliant on network connectivity, other system characteristics, such as the list of open ports, may be more relevant to an investigation than the email or browsing history. Unfortunately, the traditional static digital forensics analysis techniques fail to provide that information.

One solution to address this issue is to perform live analysis on a target system, whereby an investigator performs a forensic examination of a system while it is in a running state. In such cases the examiner has the ability to gather information which is not typically available through the static analysis approach. Live analysis of volatile system data presents significant challenges and risks. In the digital forensics equivalent of the observer effect, the examiner simply cannot examine a live system without making some change to it, however minor. From the moment the investigator logs on to the target system, logs are recorded, temporary files are created and deleted, network connections can be opened and closed, history files are updated, and registry entries are queried, added, and modified. All of these activities change the state of the system, and as such may contaminate the evidence that the investigator hopes to collect. Any evidence discovered cannot be recorded on the target system without further system modification, so a network connection is often used to pipe data to another system under the investigator's control, where the results of the live analysis can be recorded. It is also unlikely that a target system contains all of the utilities required to perform a forensics analysis, so some additional media loaded with analysis tools may be attached to system, resulting in more system activity that potentially contaminates or overwrites essential digital evidence.

In addition, it is possible that a compromised host may have a rootkit installed which will deliberately attempt to hide evidence, such as open network ports, user accounts, or the presence of files and folders in a file system. Under such circumstances some or all of the evidence obtained during live analysis may be incorrect or misleading. In the worst case a system could be

configured to detect a live analysis attempt (e.g., suppose that a suspect installed a service that requires a user to hit the escape key within 5 seconds of login to maintain system configurations) and to delete incriminating data if such an attempt was detected. Even without such "movie plot" issues live analysis is substantially more challenging than the more traditional static analysis approaches.

Some hardware based devices are available that aim to address the danger inherent in live analysis by using the DMA controller to acquire the contents of system memory without operating system or CPU interaction [4, 8, 10]. However, some research has raised the possibility that such devices could actually be presented with a different view of memory than that seen by the CPU (and therefore the operating system and applications), and as such that memory areas may be masked from the analysis. If the target system was carefully configured this may allow data vital to a forensic investigation, such as malware, to be hidden from the examiner [3].

Although live analysis is fraught with difficulties, it does offer the opportunity to gather valuable information that static analysis cannot provide, although the process of static analysis is substantially less complex. However, the recent resurgence of system virtualization and its application to commodity processors may provide a method by which live analysis of a system is possible, without the associated risk of contamination of the target system evidence.

3. VIRTUALIZATION

Virtualization is not a new concept, with origins stemming back to IBM's release of VM for the System/360 in 1972 [26], but the recent growth of x86 virtualization products such as Virtual PC/Server [16], VMware [30], QEMU [23], KVM [13], Xen [33], and Parallels [18] have provided new opportunities and challenges for digital forensics investigators as virtualization becomes increasingly mainstream. Virtualization provides the ability to host multiple guest operating systems on a single physical host system in paravirtualized or fully virtualized modes and represents a natural evolution from the need to host multiple isolated users on a single mainframe. Each guest operating system operates in a virtual environment in which it is unaware of other guest operating systems that may be running on the same physical host. As a result, virtualization facilitates the sharing of pools of computing resources while maintaining isolation between virtual machines [26].

Virtualization also facilitates replication, mobility, rollbacks, restarts, and reduces the hardware and time requirements associated with performing the same actions on multiple physical hosts. The recent rapid growth of virtualization technologies demonstrates the widespread utilization and increased acceptance of virtualization in both research and production environments. Virtualization provides a method to harness the power of the continually increasing multi-core and multi-socket systems and takes advantage of the foreseeable continuation of growth in this area. Crosby and Brown [5] predict that virtualization is "poised to deliver profound changes to the way that both enterprises and consumers use computer systems." In addition to the potential benefits for enterprise users and data centers, the computer-using public is also likely to benefit from advances in virtualization. For example, the United States National Security Agency's NetTop project involves the use of virtualization on common computing hardware to "[remove] security functionality from the control of the end-user OS and applications. Important security functions such as communications encryption can be placed in a separate protected environment that cannot be influenced by user software." [17]

Along with the many advantages brought about by virtualization, there are additional technological challenges that virtualization presents, which include an increase in the complexity of digital forensics investigations as well interesting questions regarding the forensics boundaries of a system. One of the nine research areas identified in the virtualization and digital forensics research agenda [22] is virtual introspection. Advances in virtual introspection research will greatly empower the digital forensics investigator, but present significant open challenges to the virtual introspection researcher.

4. VIRTUAL INTROSPECTION

4.1 Background

Virtual Introspection (VI) is the process by which the state of a virtual machine (VM) is observed from either the Virtual Machine Monitor (VMM), or from some virtual machine other than the one being examined. While virtual introspection is not yet available as a production tool, there have been some recent efforts that demonstrate the potential uses for this technique including the following:

- Virtual introspection for IDS/IPS [2, 7, 14, 21]
- VM-Based Malware Detection System [12, 15, 24, 27, 35]
- Virtual Access Control [20]

The XenAccess project, led by a PhD student at the Georgia Tech Information Security Center, is currently attempting to produce an open source virtual introspection library that will allow "researchers to experiment with the many uses of memory introspection without needing to focus on the low-level details of introspection." [32]

4.2 Application to Digital Forensics

Assuming the continued growth of applications of virtualization, digital forensics investigators are likely to encounter many cases where the target of an investigation is a virtual machine, or set of virtual machines. In such cases virtual introspection is likely to allow an investigator to perform live system analysis in such a manner that the state of the target system remains unchanged as a result of the analysis, which is not the case with traditional live analysis of physical systems. In addition, it may be possible to perform the analysis in such a way that the target system would be unable to detect the monitoring.

While it may seem that the mere fact that the state of the target system is not modified during the analysis makes the analysis undetectable, it remains to be determined whether some approach could allow the target system to detect the virtual introspection monitoring, either conclusively or with some elevated probability. Some possible avenues through which this

may be possible, and which therefore warrant further investigation, include:

- Timing analysis. For example, the target VM may be able to detect unusual patterns in the frequency at which it is scheduled for execution, caused by the need for the VI application to pause the VM while performing analysis.
- Page fault analysis. For example, the target VM may be able to detect unusual patterns in the distribution of page faults, caused by the VI application accessing pages that have been swapped out, or causing pages that were previously swapped out to be swapped back into RAM.

While there is no indication that detection of the VI monitoring is possible, it is important to consider whether such a determination could be made if VI is to be applied to digital forensics, particularly if the results of a VI-based investigation are to be used as evidence in a legal setting.

Simply being able to detect that a system is running as a VM is not considered significant for this application, as the intent of the tools is specifically to allow for live analysis of VMs. In all major virtualization products available today it is possible for a user to reliably determine that the operating system is running on a VM rather than directly on physical hardware. While there have been some claims of undetectable VM "rootkits" [11] it does not appear that such virtualized rootkits can be reliably disguised at this time. In fact, there seem to be many advantages of cooperative virtualization, in which the VM not only knows it is running in a virtualized environment, but cooperates to some extent with the VMM to improve overall performance [19, 31], although these benefits may require tradeoffs, such as reduced system portability. The ability to detect virtualization has been problematic for some applications in the past, in that VMs were used more commonly as honeypots than production systems, so that attackers who detect virtualization may assume that they found a honeypot system rather than a production host. However, with the increased use of virtualization for production systems [6], simply detecting a virtual environment no longer signals that the system is a non-production host. It may, however, lead the attacker to attempt to break out of the virtual environment through vulnerabilities in the VMM, thus compromising the physical host, although it is hoped that VMM designers strive to make the VMM as simple as possible, and therefore easier to provide security assurances for than the typical highly complex operating system.

The concept of the semantic gap is often raised in relation to virtual introspection. The semantic gap refers to the difference between the presentation of data from volatile memory by the operating system (e.g., to userspace processes in the target VM), and the raw format in which we can access memory using virtual introspection techniques. However, this is an inconvenience rather than a fundamental limitation, and all that is required is that virtual introspection applications perform the same translation of the raw volatile memory data to information that the operating system already performs. Depending on the depth to which the digital forensics investigator intends to examine the system, writing such tools may be time consuming, but at any given moment there is no data available to the virtual machine operating system that is not also available to the virtual introspection application, although accessing it is likely to require that the VI application perform a complex navigation through multiple levels of indirection.

5. Virtual Introspection for Xen (VIX)

The Virtual Introspection for Xen (VIX) suite of tools has been developed as part of an ongoing virtual introspection research effort, which is focused on digital forensics. Although Xen does not feature some of the more advanced management tools offered by other commercial vendors, it was selected in large part because it was open-source software, allowing the researchers to modify the VMM should that become necessary (although to date this has not been required, as the necessary functionality is exposed by the *Xen Control* library, which is available as part of the Xen distribution). In addition, Xen is under current development, supports multiple guest operating systems, including Linux and Windows, and has been integrated into several mainstream Linux distributions [25, 28]. Furthermore, several active mail lists are dedicated to Xen development and usage issues [34].

Although Xen was used as the virtualization platform for this project, the techniques developed could be applied to any of the other major virtualization platforms, although the implementation details of the memory access would obviously differ. In each of these alternate platforms the VMM enforces the separation between virtual machines, and as such making pages from one VM available to other parts of the system requires at most a modification to the VMM, and, as is the case with Xen, the VMM may already provide such functionality and an API with which to invoke it.

While it does enable virtual introspection, the fact that the VMM has full access to the resources of all VMs that it manages also presents a significant risk, in that a compromised VMM can be easily leveraged to compromise all of the VMs on that system. The issue of whether VMs can ever be managed by a VMM, while simultaneously being protected from a compromised VMM remains an open research problem.

5.1 VIX Tool Suite

The VIX tools are designed to allow an investigator to perform live analysis of an unprivileged Xen virtual machine, which are known as DomU systems, from the privileged Dom0 virtual machine. Using this approach, neither the virtual machines nor the virtual machine manager require modification. VIX consists of a library of common functions, and a suite of tools which mimic the behavior of common Unix command line utilities, such as `ps`, `lsmod`, `netstat`, `lsof`, `who`, and `top`.

The basic approach taken by these tools is to pause the target virtual machine, acquire the data necessary to perform the requested function using read-only operations, and then un-pause the target VM. Using this approach we can ensure that the state of the VM does not change during the data acquisition process, and that the state of the VM is not modified while its execution is suspended.

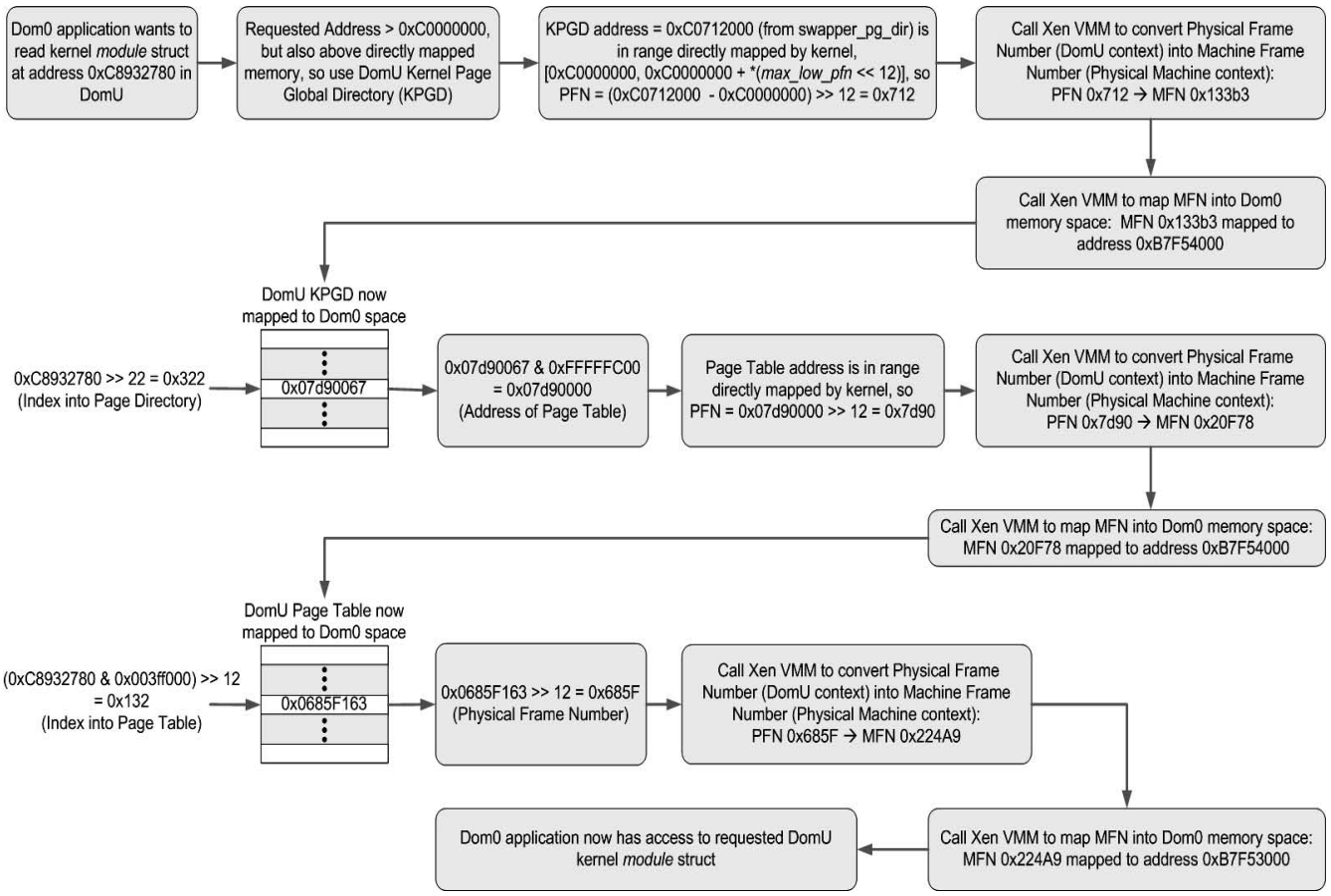

Figure 1: Example of mapping a page containing a DomU virtual memory address into Dom0.

The Xen Control library is used to allow the Dom0 system to perform operations on the DomU systems, including pausing the target VM prior to analysis, and un-pausing it afterwards. Another important function provided by the library is mapping memory pages assigned to the target VM into the address space of the Dom0 system, allowing the analysis to be performed.

When attempting to access a memory address in the context of a process in a Linux VM, there are several steps that must be performed prior to the relevant page being available to the Dom0 system. As with any Linux system, a memory address is only relevant in the context of a process, and in order to find the appropriate physical frame in memory we must consult the page tables to determine the corresponding physical frame number. In the case of physical machines that would be sufficient, but there is another layer of complexity added by virtualization because the physical frame numbers from the perspective of the VM must be translated into the physical frame numbers for the underlying physical hardware before the appropriate page can finally be made available to the Dom0 system. Figure 1 shows an example of this operation, where a page containing the address of a *module* struct, which contains information about a single kernel module, is mapped into Dom0 space on a 32-bit x86 system with PAE disabled.

The task of converting the DomU Physical Frame Number (PFN) to the frame number for the physical machine (MFN) is handled via a call to the Xen Control library, as is the operation that maps a given MFN into Dom0 memory space. However, the Dom0 application is responsible for correctly traversing the appropriate DomU virtual memory structures, and as such any page directory or page table needed during this traversal must also be similarly mapped into Dom0 to allow the application to access them. While this further complicates what would be a simple operation were it to be performed either directly on the DomU system, or on a physical system with no virtualization involved, it is a necessary operation required for virtual introspection. Of course, care must also be taken to unmap any mapped pages to prevent memory leaks.

At this point the Dom0 application can examine the mapped page. Many of the kernel structures examined by the VIX tools, such as the process list, make extensive use of pointers, which of course are also only valid in the context of the DomU process. As such, a task as seemingly trivial as walking a linked list becomes significantly more complex in a virtual introspection application. Much of this activity is now abstracted away from the individual VIX tools by the libvix API, but it is important to remember that it does occur so that applications built on libvix minimize the time for which execution of the target system is suspended.

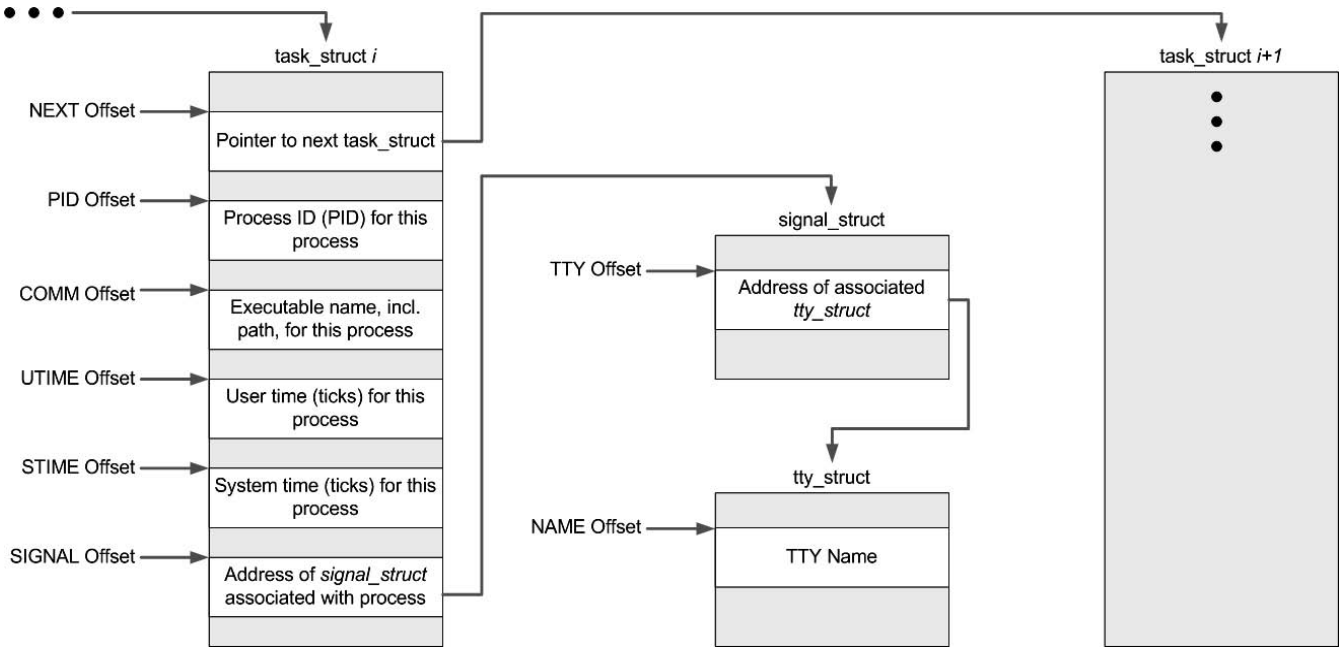

Figure 2: Fields and data structures involved in decoding a task_struct for a single process for a basic invocation of `vix-ps`.

The `ps` utility [29] is used on Unix systems to list the current processes, and when it is executed on a Linux host it makes use of system calls and the `/proc` filesystem to gather the relevant process data, which is then formatted and displayed to the user. The `vix-ps` utility performs the same functionality for a given DomU, using virtual introspection techniques, meaning that the system calls and `/proc` filesystem are essentially not available. The pseudo-code for the `vix-ps` utility is as follows:

```
Pause DomU
Adr <- Address of Task List Head
Do
    Adr <- Adr.next_task_adr
    Map page(s) for Adr into Dom0
    Decode task_struct
    Display data
    Unmap page(s) with Adr
While (Adr != Address of Task List Head)
UnPause DomU
```

Once a page containing a task_struct has been mapped to Dom0 in the `vix-ps` application, the relevant data must be extracted for display to the user. The data acquired in the case of the most basic version of the `vix-ps` command, equivalent to `ps e` on a Linux system, is shown in figure 2. Note that task_struct fields that contain addresses of other data structures must be viewed in the context of the DomU system, and as such additional pages may need to be mapped into Dom0 system in order to access the signal_struct and tty_struct data structures for a given process.

Decoding the task_struct provided in the mapped page in the case of `vix-ps` is a manual process, and while it would be tempting to use the task_struct defined for the kernel of the DomU system, we cannot assume that data structures present in the Dom0 system match the structures used by the DomU kernel to which the page, and the task_struct data, belongs. This issue is addressed by using flexible offset values into the various structures used by the VIX tools, which are initialized based on the kernel version of the DomU system being examined. A kernel module, named getOffsets, has been developed as part of this work, and has been used to determine the appropriate offsets for a variety of example Linux kernels. Work is also currently underway on a more flexible approach, in which the correct offsets are determined by probing the target VM for known or likely values.

5.2 VIX Example

To demonstrate the functionality of the VIX tools, a Xen system was configured to run two VMs, including the privileged Dom0 system, and an unprivileged DomU system called *testhvm1*. Figure 3 shows the output of the `xm list` command, which displays the current VMs on the system. Each of the current VIX tools takes a parameter that identifies the ID of the VM on which the tool will execute.

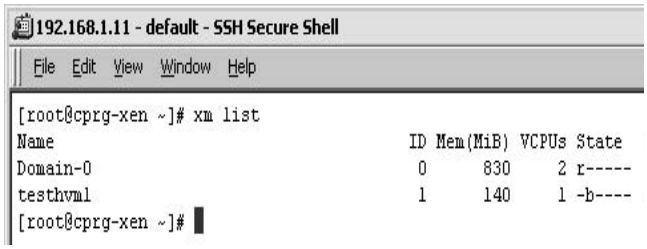

Figure 3: Xen system running with two virtual machines.

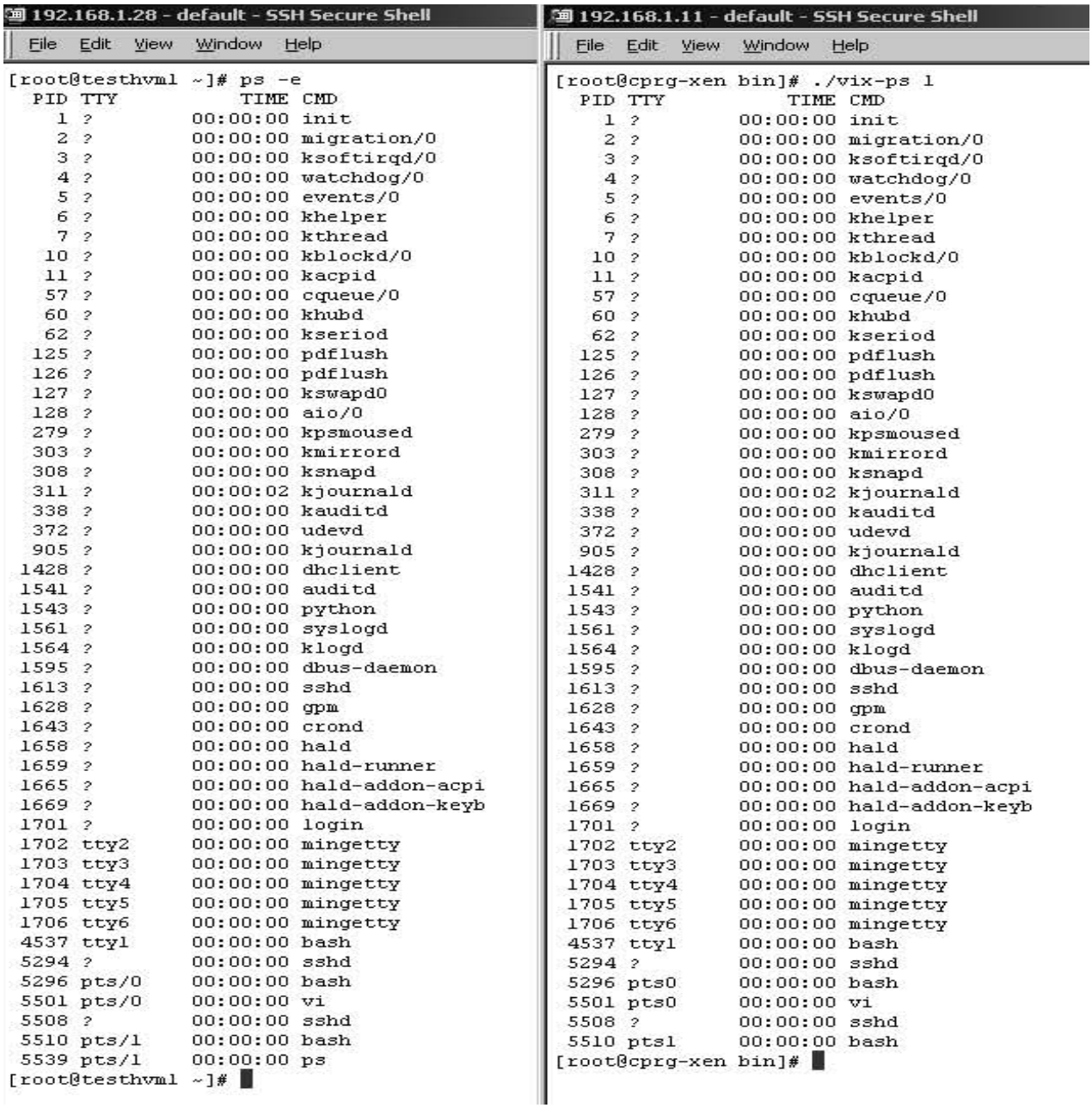

Figure 4: Comparison of process listings taken from within a VM using the traditional ps command on the DomU (left), and from the Dom0 system using virtual introspection (right).

Figure 4 shows the output of the `ps ecommand` on the *testhvm1* DomU VM on the left, followed by the corresponding output from the `vix-ps` command run against *testhvm1* (ID 1) from the Dom0 system on the right. The listings are identical, other than the addition of the last entry on the left, which is the `ps` command itself that was obviously running at that time, and which had completed execution before the `vix-ps` command was invoked.

As shown in figure 5, a second DomU system, called *testhvm-rk1*, has been added to the demonstration system. This VM is a Linux system onto which a basic rootkit has been installed for the purposes of this exercise. This rootkit is capable of hiding modules and process, and as a result the true state of the system may not be apparent to a user of that system, including a forensic investigator.

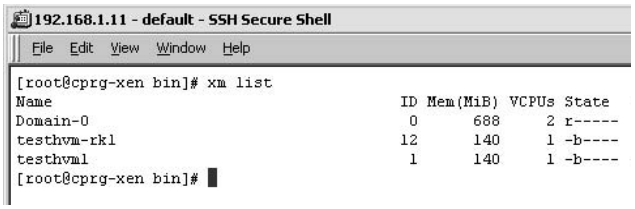

Figure 5: Xen system running with three virtual machines. The testhvm-rk1 VM has a rootkit installed.

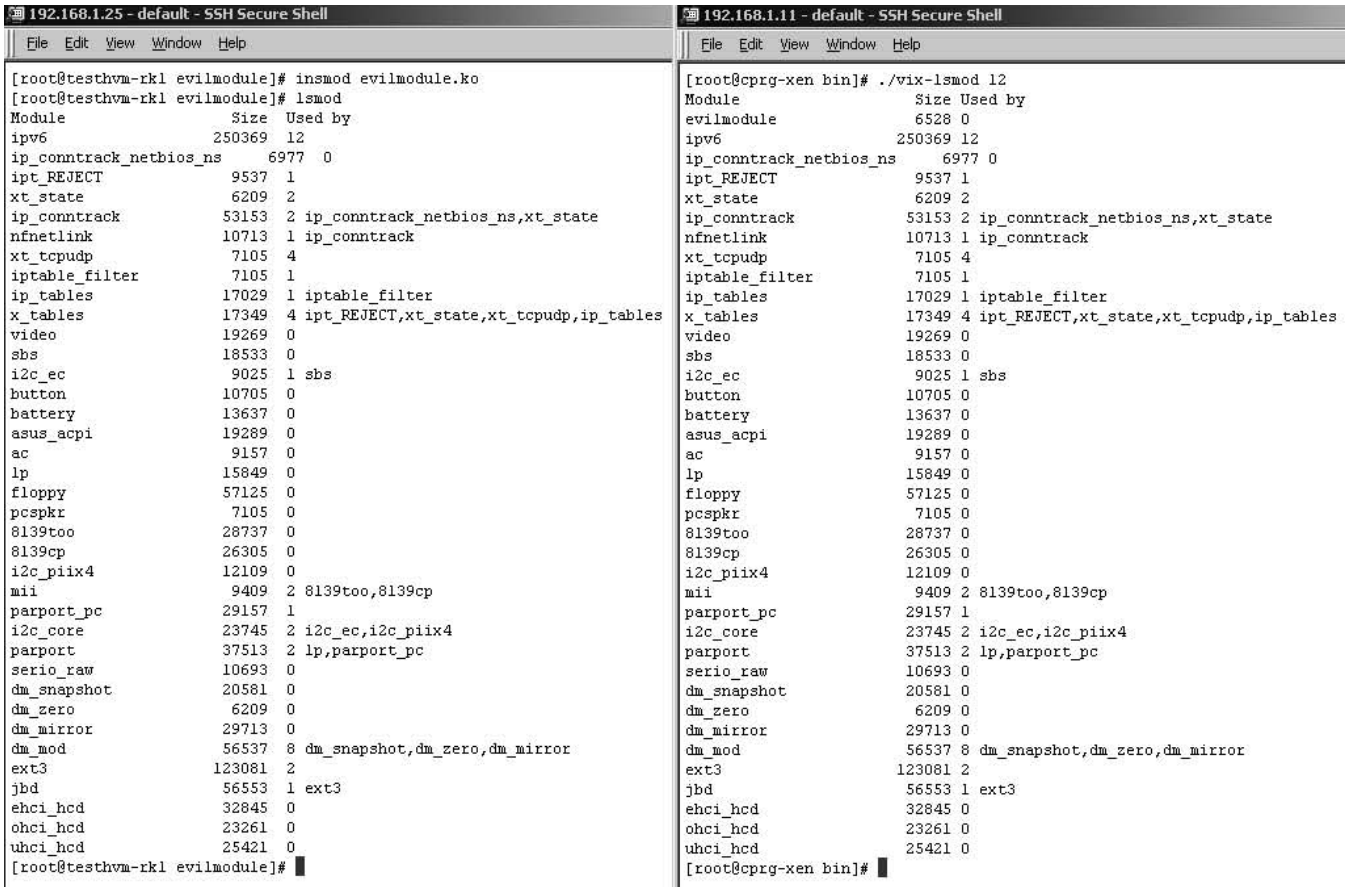

Figure 6: Comparison of module listings taken from within a VM with a hidden kernel module (evilmodule) installed using the traditional lsmod command from DomU (left), and from the Dom0 system using virtual introspection (right).

For example, when executed on the DomU system, the `lsmod` utility does not display the presence of the *evilmodule* module. However, by using virtual introspection from Dom0 to directly access memory, without relying on any of the DomU operating system functionality, the `vix-lsmod` tool correctly reports the true state of the system, which includes the hidden kernel module, as shown in figure 6.

6. DISCUSSION AND CONCLUSION

This paper presents a discussion of virtual introspection and introduces the VIX suite of tools to advance the research agenda for virtualization in digital forensics. Although progress has been made in the application of virtual introspection to digital forensics, much work remains to be done in this field.

6.1 Future Work

The following list includes some of the next steps that would contribute to the state of virtual introspection both as applied to digital forensics, and to the VI field in a more general sense:

- Support for multiple operating system types. While the Linux 2.6 kernel is already relatively well supported by VIX tools, support for other operating systems, particularly Windows, is an important step to allowing the VIX tools to have widespread practical significance.
- Analysis of the extent to which the virtual introspection techniques can be detected by the target virtual machine. For example, it is possible that timing analysis or page fault monitoring may provide some indication to the target VM that monitoring was taking place, despite the fact that the VM is not active during the VI process. If virtual introspection is to be applied to digital forensics for legal investigations it is important to understand this issue.
- Application of these VI techniques to other virtualization platforms. In particular the VMware suite of virtualization products is an important target, as those products currently dominate the marketplace, particularly in the enterprise environment. While the mechanism by which pages from the VMs are made available to the VI applications may change, it should be possible to retain a common code base for decoding the pages once they are made available.
- Applications of these techniques to other domains. A domain being actively investigated at this time is honeypots, where the current monitoring techniques for high interaction honeypots typically involve hidden monitoring processes running on the honeypot systems themselves, which are detectable by skilled intruders. Honeypots often run as virtual machines, and the ability to use virtual introspection may make the

monitoring less visible to the intruder, allowing honeypot researchers to gather more information about the activities of attackers.

- Development of a framework to allow virtual introspection across physical hosts. In large scale virtualized environments VMs are typically not restricted to a particular physical hosts, but are instead dynamically load balanced across a pool of physical hosts, even while running. In such cases it would be valuable to allow the forensic investigator to monitor the VM from a single physical host, no matter where in the pool the VM was currently executing. Such techniques would also be applicable to data centers, allowing administrators to monitor VMs deployed across a large number of physical hosts throughout the data center from a central monitoring point.

6.2 Conclusions

Assuming that the current trend towards system virtualization continues, Virtual Introspection techniques can be utilized by forensics investigators to perform unobtrusive live analysis on target virtual machines, with greatly reduced risk of contaminating evidence on the target system, While there are still many open problems in this area, including a quantification of the extent to which Virtual Introspection is detectable from the target system, the VIX suite of tools currently provides an excellent proof of concept of the utility of VI in digital forensics.

Acknowledgements

The authors would like to acknowledge the contributions of Mark Pollitt in leading the team that defined the research agenda for virtualization in digital forensics.

7. REFERENCES

[1] Access Data. Retrieved August 10, 2007 from http://www.accessdata.com/

[2] Asrigo, K. L. Litty, D. Lie. Using VMM-Based Sensors to Monitor Honeypots. In *Proceedings of the 2nd ACM/USENIX International Conference on Virtual Execution Environments (VEE 2006),* June, 2006.

[3] Beyond the CPU: Defeating Hardware Based RAM Acquisition. Retrieved November 15, 2007 from http://i.i.com.com/cnwk.1d/i/z/200701/bh-dc-07-Rutkowska-ppt.pdf

[4] Carrier, B., and Grand, J. A hardware-based memory acquisition procedure for digital investigations. *The International Journal of Digital Forensics & Incident Response*. Retrieved November 15, 2007 from www.sciencedirect.com.

[5] Crosby, S., and Brown, D. (2006). The Virtualization Reality. *ACM Queue*, December/January 2006-2007, pp. 34-41

[6] Data Center Management Research Report September 2007. Retrieved November 15, 2007 from http://www.novell.com /products/zenworks/orchestrator/data_center_research_report_sep2007.pdf

[7] Garfinkel, T. and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the 10th Annual Symposium on Network and Distributed System Security (NDSS 2003)*, pages 191–206, Feb. 2003.

[8] Grand Ideas Studio: Tribble. Retrieved November 15, 2007 from http://www.grandideastudio.com/src/portfolio.php?cat=&prod=14

[9] Guidance Software, Inc. EnCase. Retrieved August 10, 2007 from http://www.guidancesoftware.com/

[10] Hit by a Bus: Physical Access Attacks with Firewire. http://www.security-assessment.com/files/presentations /ab_firewire_rux2k6-final.pdf

[11] Introducing Blue Pill. Retrieved November 15, 2007 from http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html

[12] Jiang, X., Wang, X., and Xu, D. 2007. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA, October 28 - 31, 2007). CCS '07. ACM, New York, NY, 128-138. DOI=http://doi.acm.org /10.1145/1315245.1315262

[13] Kernel based Virtual Machine. Retrieved November 18, 2007 from http://kvm.qumranet.com/kvmwiki.

[14] Kourai, K. and Chiba, S. 2005. HyperSpector: virtual distributed monitoring environments for secure intrusion detection. In *Proceedings of the 1st ACM/USENIX international Conference on Virtual Execution Environments* (Chicago, IL, USA, June 11 - 12, 2005). VEE '05. ACM, New York, NY, 197-207. DOI=http://doi.acm.org /10.1145/1064979.1065006

[15] Litty, L. and Lie, D. 2006. Manitou: a layer-below approach to fighting malware. In *Proceedings of the 1st Workshop on Architectural and System Support For Improving Software Dependability* (San Jose, California, October 21 - 21, 2006). ASID '06. ACM, New York, NY, 6-11. DOI= http://doi.acm.org/10.1145/1181309.1181311

[16] Microsoft Virtual PC Server. Retrieved July 15, 2007 from http://www.microsoft.com/windows/products/wi

[17] National Security Agency Central Security Service – Technology Profile Fact Sheet. Retrieved November 15, 2007 from http://www.nsa.gov/techtrans/techt00011.cfm

[18] Parallels. Retrieved July 25, 2007 from http://www.parallels.com/

[19] ParavirtBenefits. Retrieved November 15, 2007 from http://virt.kernelnewbies.org/ParavirtBenefits

[20] Payne, B. D., Sailer, R., Cáceres, R., Perez, R., and Lee, W. 2007. A layered approach to simplified access control in virtualized systems. *SIGOPS Oper. Syst. Rev.* 41, 4 (Jul. 2007), 12-19. DOI= http://doi.acm.org/10.1145 /1278901.1278905

[21] Petroni, N. L. and Hicks, M. 2007. Automated detection of persistent kernel control-flow attacks. In *Proceedings of the*

14th ACM Conference on Computer and Communications Security (Alexandria, Virginia, USA, October 28 - 31, 2007). CCS '07. ACM, New York, NY, 103-115. DOI= http://doi.acm.org/10.1145/1315245.1315260

[22] Pollitt, M., Nance, K., Hay, B., Dodge, R., Craiger, P., Burke, P., Marberry, C., and Brubaker, B. Virtualization and Digital Forensics: A Research and Education Agenda in *Journal of Digital Forensic Practice*. Taylor & Francis, Philadelphia, PA. (in press)

[23] QEMU. Open Source Process Emulator. Retrieved on November 18, 2007 from http://fabrice.bellard.free.fr/qemu/.

[24] Quynh, N. A. and Takefuji, Y. 2007. Towards a tamper-resistant kernel rootkit detector. In *Proceedings of the 2007 ACM Symposium on Applied Computing* (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 276-283. DOI= http://doi.acm.org/10.1145/1244002.1244070

[25] Red Hat Enterprise Linux 5 – Virtualization. Retrieved November 15, 2007 from http://www.redhat.com/rhel/virtualization/

[26] Rosenblum, M. 2004. The Reincarnation of Virtual Machines. *Queue* 2, 5 (Jul. 2004), 34-40.

[27] Seshadri, A., Luk, M., Qu, N., and Perrig, A. 2007. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA, October 14 - 17, 2007). SOSP '07. ACM, New York, NY, 335-350. DOI= http://doi.acm.org/10.1145/1294261.1294294

[28] SLES 10 – Novell Virtualization Technology. Retrieved November 15, 2007 from http://www.novell.com/documentation/vmserver/pdfdoc/virtualization_basic/virtualization_basics.pdf

[29] UNIX man pages: ps. Retrieved November 15, 2007 from http://unixhelp.ed.ac.uk/CGI/man-cgi?ps

[30] VMware. Retrieved November 18, 2007 from http://www.vmware.com.

[31] VMware White Paper: Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Retrieved November 15, 2007 from http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

[32] XenAccess Documentation. Retrieved November 15, 2007 from http://xenaccess.sourceforge.net/doc/index.html

[33] Xensource. Retrieved July 27, 2007 from http://www.xensource.com/xen/xen/nfamily/virtualpc/default.mspx

[34] Xen: Mailing Lists. Retrieved November 15, 2007 from http://lists.xensource.com/

[35] Xu, M., Jiang, X., Sandhu, R., and Zhang, X. 2007. Towards a VMM-based usage control framework for OS kernel integrity protection. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies* (Sophia Antipolis, France, June 20 - 22, 2007). SACMAT '07. ACM, New York, NY, 71-80. DOI= http://doi.acm.org/10.1145/1266840.1266852