UC Davis

UC Davis Previously Published Works

Title

Investigating the Implications of Virtual Machine Introspection for Digital Forensics

Permalink

https://escholarship.org/uc/item/7hs6t0nw

Authors

Nance, Kara
Hay, Brian
Bishop, Matt

Publication Date

2009-03-01

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Investigating the Implications of Virtual Machine Introspection for Digital Forensics

Kara Nance and Brian Hay
Department of Computer Science
University of Alaska Fairbanks
Fairbanks, AK
ffkln@uaf.edu and brian.hay@uaf.edu

MarttBishop
Computer Science Department
University of California Davis
Davis, CA
bishop@cs.ucdavis.edu

Abstract— **Researchers and practitioners in computer forensics currently must base their analysis on information that is either incomplete or produced by tools that may themselves be compromised as a result of the intrusion. Complicating these issues are the techniques employed by the investigators themselves. If the system is quiescent when examined, most of the information in memory has been lost. If the system is active, the kernel and programs used by the forensic investigators are likely to influence the results and as such are themselves suspect. Using virtual machines and a technique called virtual machine introspection can help overcome these limits, but it introduces its own research challenges. Recent developments in virtual machine introspection have led to the identification of four initial priority research areas in virtual machine introspection including virtual machine introspection tool development, applications of virtual machine introspection to non-quiescent virtual machines, virtual machine introspection covert operations, and virtual machine introspection detection.**

Keywords-virtualization, digital forensics, cirtual machine introspection, VMI.

I. INTRODUCTION

Digital forensics is an increasingly important and diverse area of computer science, charged with providing vital evidence to legal proceedings, determining methods of attack, and gathering data to determine what vulnerabilities were exploited in an effort to compromise a particular system. However, researchers and practitioners in this field typically base their analysis on information that is either incomplete or produced by tools that may themselves be compromised as a result of the intrusion.

The traditional incident response procedure, known as quiescent analysis, is to take custody of the compromised system, shut it down, copy the storage media, and then analyze the copy using a variety of forensics tools. The shutdown process amounts to either invoking the normal system shutdown sequence, or pulling the power cord from the system to effect an instant shutdown [1]. In both cases it is likely that the storage media no longer accurately reflects the state of the system before shutdown. In the case of the normal shutdown, data is read and written to the hard disk, which may delete or overwrite forensically relevant content, whereas in the case of the immediate power-off, data in the disk cache may not be synchronized with the hard disk. A more fundamental problem also exists, in that much of the information of interest to the examiner, such as the process list, kernel configuration, network status, encryption keys, and unencrypted data, may only exist in RAM, and as such will not appear in the forensic image of the hard disk.

As the importance of this volatile data to digital forensics analysis has become more evident, non-quiescent analysis techniques, in which the system is examined during operation, have become increasingly common. This method can provide the investigator with access to the volatile information that is inaccessible during quiescent analysis, but it too suffers from some serious flaws. The first problem, known as the observer effect, is that any operation performed during the live analysis process modifies the state of the state system, including the contents of memory and storage media, and as such can taint any evidence recovered during the investigation. The second problem is that non-quiescent analysis relies, at some level, on the system utilities that exist on a potentially compromised system. As such the results can be manipulated by an attacker to be false or misleading. In addition, any forensic utilities executed during the live analysis can be detected by a sufficiently careful and skilled attacker, who can at that point change behavior, delete important data, or actively obstruct the investigator's efforts.

Ideally, the forensic investigator would be able to analyze the system from a distance, leaving no trace of the probing (or even of altering data, if such is done). In essence, what is needed is an invisible, undetectable path over which the target system can be (passively and actively) monitored. Such an approach requires special hardware or some form of virtualization. The latter is available in the guise of a technology called virtual machine introspection.

The goal of this paper is to apply virtual machine introspection to the forensic problems discussed above. We identify several research issues that need to be examined in order to determine the value of virtual machine introspection as a forensic tool, how it can best be used, and what its limitations are.

The next section reviews the idea of virtual machine introspection. Section 3 identifies four research issues: the development of forensic tools, the monitoring of active virtual machines (as opposed to those paused, or quiescent), active monitoring (in which data in the virtual machine is

altered), and the detection of virtual machine introspection from within the monitored virtual machine. We conclude with a look at other directions.

II. Virtual Machine Introspection

The increasing availability and use of virtualization enables investigators to have access to the complete state of a target system in a manner that is in no way reliant on the target system to provide information. In addition, this form of live analysis may prove to be extremely difficult for attackers to detect, if they can detect it at all. Suppose a computer system runs in a virtual machine (VM), supervised by a virtual machine monitor (VMM). The computer system has network connections, accounts, and in all ways appears to be an ordinary computer system. An attacker breaking into the system would probe it and proceed as though the attack were on a regular system—which, indeed, it is, as all user data (and other data) resides on this VM. But, when the attack is discovered or suspected, the security analysts do not begin running programs on the attacked system to analyze it. Instead, they go to either the VMM or a second VM running under the control of the VMM and "look inside" the attacked VM. This technique is called virtual machine introspection (VMI) and was first introduced by Garfinkel and Rosenblum [2].

A VM presents the appearance of hardware to those processes that run on it—and that includes the resident operating system. So, if the operating system in the VM uses pages, the page tables are also stored in the VM's memory space. When the process in the VM triggers a page fault, the operating system in the VM attempts to fetch the missing page. But it executes privileged instructions to do so, causing a trap to the VMM. Thus, the VM traps to the VMM, which sets the registers of the VM to make it appear that the process page fault trapped directly to the VM. It then returns control to the VM's operating system which determines the "trap" was caused by a page fault, and so loads the page, causing another trap to the VMM, which loads the page and makes it available to the VM, and then updates the VM's registers. It then returns control to the VM's operating system, which in turn returns control to the process. As far as the process is concerned, it interacted only with the operating system on the VM and the "hardware" (VM) itself. The interesting point is that the VMM has complete access to all memory in the VM, and can read and alter it as needed.

Given this, it is possible for a program to use the VMM to reconstruct the contents of a process's memory space, and even the contents of the VM's kernel memory, by using the page table for the VMM to obtain an image of the VM's memory. From that, it accesses the VM kernel memory to locate the process(es) of interest, and the associated memory pages. The program can then reconstruct exactly what the program was doing from this information.

This approach has additional advantages over standard methods of forensic analysis. First, if one performs quiescent forensic analysis by imaging the disk and analyzing it off line, one generally loses the contents of memory, which (as noted above) may be critical to accurately reconstructing the events. Second, if one performs non-quiescent forensic analysis by executing investigative programs on the system while it is running, the data obtained may be deceptive if the kernel, libraries, or programs themselves have been compromised by (for example) a rootkit. But if the analysis is done from the VMM, any compromised components of the operating system, libraries, or programs on the VM will be irrelevant as the tools bypass them, and run only on the VMM. In this case, the non-quiescent analysis may look at memory, but because the VM is not turned off; it is merely quiescent, with memory contents being retained. The quiescent analysis poses problems of consistency, because as the snapshot is being taken, the locations not yet read may change, producing an inconsistent representation of memory contents, which can make analysis more problematic. The tools and techniques in virtual machine introspection are evolving quickly as virtualization becomes more mainstream and its use in production systems increases.Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your word processor, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

III. Research Questions

Research in the application of VMI [3] has typically focused on intrusion detection rather than digital forensics. Recent work [4] has considered digital forensics as a primary VMI application area, and there has also been associated work in tools that construct forensically relevant information from memory images [5, 6].

These tools and techniques raise several interesting research questions for VMI, and providing answers to such questions is vital if VMI is to be used for digital forensics, as the results of digital forensics investigations can have serious and wide-ranging legal consequences. First, in order to read memory correctly, the process memory must be consistent, which in the simplest case means the VM must be paused. Can the attacker determine this? If so, the attacker may be able to exploit this knowledge to mislead the analysis, or to delay until the snapshot is taken before launching specific attacks. Second, can a consistent snapshot be taken without pausing the system? That is, given a non-static memory, can one obtain enough information from a snapshot, or series of snapshots taken over time, to determine what is happening? Third, beyond the obvious use of digital forensics, to what uses can this ability be applied? Certainly intrusion detection is an obvious one; another is to analyze the vulnerabilities on the system that allow an attacker to breach security. But the field of digital forensics, especially when applied to virtual machines is so new, ill-defined, and conceptually so wide, that it has applications in those areas, and many more beyond. These and other related question led to the identification of four initial priority research areas that need to be addressed in order to effectively investigate the limits of VMI. These include VMI tool development, applications to non-quiescent VMs, VMI covert operations, and VMI detection.

A. VMI Tool Development

Under any current virtualization system, compromise of the VMM is catastrophic to almost any security property of all virtual machines running under the VMM. As such, a good approach is to follow a policy in which the VMM is, to borrow a phrase from Einstein, "as simple as possible, but not simpler". The addition of any VMI capabilities to the VMM itself must, therefore, be carefully weighed against this goal in order to ensure that functionality that could be implemented in the virtual machine be identified and implemented in the appropriate location, rather than taking the easier route of simply adding functionality directly to the component with the highest privilege level on the system, namely the VMM. There are several approaches under current investigation that may allow the VMM to remain unmodified, or at most slightly modified, while enhancing the opportunities for VMI. For example, a privileged virtual machine (called Dom0) under Xen [7] serves, in many cases, as the direct interface to physical hardware, such as the network cards. So monitoring of non-privileged virtual machine (called DomU) interactions with hardware may require modifying Dom0. Extending this approach further, it is possible in some circumstances to relocate the underlying driver for physical hardware from Dom0 to a DomU (which is known as a "driver domain"). This allows an application running in an unprivileged VM to provide the introspection capabilities with respect to the interaction between target VMs and certain hardware components. Relocating the driver into the unprivileged DomU alters the mechanism performing the virtualization of the hardware, and not the virtual hardware itself, and so should be invisible to the processes and systems in DomU.

Mapping memory from DomU systems into Dom0 is not limited to read-only access, and as such it is also interesting to consider what additional capabilities might be gained, and what associated costs would be incurred, by modifying, even temporarily, memory within a target DomU VM. For example, write access could be used to provide VMI "breakpoints" in DomU applications, although doing so in a manner that preserved the difficulty of detecting VMI may be challenging (although such a tradeoff may be acceptable depending on the intended use, and may still be less detectable than current monitoring techniques, such as those employed by Sebek, for example). One approach in this category under current investigation is the temporary replacement of instructions in an executable with calls that trap to the VMM, which can then be redirected to Dom0 for processing.

B. Application to Non-quiescent VMs

The current state of the art in analyzing virtual machines allows an analyst to pause or suspend the VM, and take a dump (called a snapshot) of its memory. While a valuable investigative technique, this approach limits forensic analysis because it does not capture the causes of the changes to memory. More valuable would be the ability to observe the system calls and other functions actually executed; such information is critical to establishing what happened [8].

The application of virtual machine techniques and tools to non-quiescent VMs is an interesting area that is likely to lead to further research questions in order to test the limits of what information can be gleaned when virtual introspection tools are applied to non-quiescent VMs. In [9], Carrier describes the risks of analyzing incidents on the systems on which they occur (which he called "live analysis"). He focuses on the untrustworthiness of the system components, using as an example a system with a kernel-based rootkit. He proposes using tools that effectively emulate the kernel's routines for reading files, and suggests using system calls to read individual sectors of the disk. He noted that a rootkit that modified the kernel's read system call would be difficult to write and, hence, unlikely to be part of an attacker's repertoire.

There are additional problems, most notably that of consistency. The kernel cannot be trusted to enforce mutual exclusion between the analysis programs and the system for several reasons, including potential compromise of the kernel. Additionally, the kernel would have to lock large portions of memory (including its own) and disk, which would not only cause problems for other processes on the system, but is also simply not an ability that most kernels possess. Thus, one part of the "snapshot" of the process will reflect the state of the process at one time, and another part will reflect the state of the process at a later time. This is analogous to traditional problems of concurrency and synchronization without mutual exclusion, especially the readers-writers problem [10] and in this context may be called inconsistent static auditing [11].

It is unclear how inconsistent static auditing affects reconstruction of the state. The effect will depend on the organization of data within memory and the system. Specifically, let a "snapshot" be composed of one or more units, defined as the contents of (virtual) memory or disk that the monitoring process can read as a single entity. Then the contents of each unit will be consistent because they are "locked" while the I/O transferring the contents from the object to the memory of the monitoring process is in progress. But the contents of different units may be inconsistent. So, if the critical information for the analysis is in the same unit; if those units can be shown to have been taken while the process was quiescent (for example, swapped out); or if the nature of the inconsistency can be determined and its effect on the analysis compensated for, the analysis should be unaffected.

How to determine this, and how to compensate for it—or determine that it cannot be compensated for and inform the analyst—requires a detailed analysis of what the VMI tools do and how they do them. This speaks directly to the testing and measuring of the effectiveness of these tools, because the environment in which they are used may affect their accuracy. When the target is a paused VM, evaluating the tools is straightforward. But when the target is an active VM, the evaluation must take into account inconsistencies of the sort described above, and the tools must know whether they can compensate for these inconsistencies (and if so how), or merely report the problem to the investigators. As changes are made in VMI techniques and extended to non-quiescent

VMs, the investigation of the detectability of the tools becomes an increasingly interesting research question.

C. VMI Covert Operations

The development of VMI tools such as VIX [4] has resulted in some unanticipated capabilities beyond the original intent. With access to the presumed private data associated with a particular VM, a VMI tool has the inherent potential to inject new data or code or modify existing data on the VM. Since VMI capabilities potentially provide ancillary powers that can be used for good or evil, those capabilities, in turn, introduce a plethora of interesting research problems beyond those originally identified. A VM should ultimately be able to protect itself from, or at least detect, the activities of a malicious user (insider from the VM perspective), a malicious VMM administrator (outsider to the VM, but insider to the VMM), and a malicious remote attacker (outsider to both the VM and VMM).

VMI Modification investigates the application of deception techniques [12] involving the modification of the VM's (virtual) hardware and processes from Dom0. The idea is to simulate hardware failure, or to change program instructions or data in order to influence the actions of the suspect programs. The analyst can present the illusion of hardware that is broken, or that works intermittently to slow an attack. When the suspect program interacts with other processes on the VM, the analyst can alter the way the processes act so as to present a VM with multiple inconsistent configurations. Numerous studies have reported on the use of deception as a defensive mechanism [13, 14, 15, 16, 17]. The use of virtual machine eliminates the need for host-based deception to use a modified kernel and modified software. This type of modification raises all the issues of monitoring non-quiescent systems discussed above.

The range of potential research problems, such as the ability for a given VM to protect itself certainly has some similarities and potentially may be viewed as analogous to those in non-virtualized environments. It remains to be determined how closely coupled the two research areas are and, if the issues in the virtualized environment supersede those evident in the analogous non-virtualized environment. In addition, there are obvious questions about how well the approaches to addressing these issues in a non-virtualized environment will map to the virtualized environment. These issues area closely related to the ability to detect VMI, as that would likely be a logical step in addressing issues associated with VMI covert operations.

D. VMI Detection

The VMI Detection investigates under what conditions, and to what degree of certainty, a process on the VM can determine it is being monitored. In both of the previously mentioned deployment environments, digital forensics and honeynets, it is important to understand the extent to which the monitoring can be detected from within the target virtual machine. In many cases, a user running processes in a virtual machine can reliably determine that those processes are running in a VM [18, 19], but this is a different and essentially irrelevant issue. In the past, x86 virtual machines were seldom deployed as production systems, and an attacker who detected virtualization could reasonably assume that the compromised system was part of a honeynet, or a low value target, at which point they could either disconnect or modify their behavior. However, the surge in VM deployment for production systems, and the likely continued increase in mainstream usage of VMs as multi-core, multi-processor systems become common place, means that the fact of virtualization does not imply that the system is a honeypot. It is important to be able to determine whether a user on a virtual machine can also detect the VMI monitoring, and if so to what extent and under what conditions. It would be easy to suggest that no detection can be possible in cases such as VIX, based on the fact that the VMI techniques require no modification to the target VM, nor is there any monitoring process running on the target VM. However, it is not clear that this is the case, even if the VMI is performed while the target VM is paused and only read operations on the target VM's resources are employed. Several channels have already been identified through which the target VM may be able to determine that VMI is actually occurring, or even that it is likely to be occurring, including the following:

- Analysis of the wall clock timing of VM execution. On a virtualized system, VMs are scheduled for execution in much the same way that processes are scheduled on a modern operating system, and as such there are always "pauses" in the execution of a VM whether it is being monitored or not. But the pattern of execution frequency or the periods for which the VM is paused may vary from some norm during VI monitoring, which may provide an indicator by which the monitoring can be detected from the VM, despite the VI process not having made any change to the VM state.
- Analysis of page faults. When a process requests access to a memory address, the relevant data may either exist in RAM or may have been paged out to disk, in which case the process must wait while the appropriate data is loaded from swap to RAM. If the VI process results in data associated with a target VM being either paged out or in, a process within the VM may be able to detect the inconsistency between the actual state of the page (in RAM or swap) and the expected state. As before, this is likely to occur during normal operations on a virtualized system, but the pattern with which these inconsistencies occur may provide an indication that monitoring is underway.
- Paravirtualization versus full virtualization. Virtual machines can be deployed in either fully virtualized or paravirtualized modes. A fully virtualized VM requires no modification to the guest operating system, and the VM behaves in exactly the same manner that it would were it not virtualized. Under the paravirtualized model, the guest system is changed in some manner to allow the VM to

cooperate with the VMM to implement the virtualization. An example of such a change is the replacement of some privileged instruction in the guest operating system with calls directly to the VMM. There are advantages and disadvantages to both approaches. It will be instructive to determine to what extent the VM can cooperate with the VMM (i.e., use paravirtualization) without enabling VI monitoring to be detected.

This is certainly not an exhaustive list, but is representative of the types of issues that must be examined before any claims of "undetectable monitoring" can be made. Research efforts may result in the discovery of methods that provide absolute and reliable indications that monitoring is underway. Alternatively, we may discover that the VM may be able to detect anomalies that could lead a user to suspect, but not be certain, that monitoring is taking place. In either case, the extent to which the monitoring can be detected, and the circumstances that enable such detection, will affect the implementation of the tools and the procedures for the use of the tools (for example, in a digital forensics investigation, an investigator may want to use tools for which no detection method is known first, followed by those methods which may be detectable).

The research in non-quiescent systems described in the previous section will influence the issue of detectability. A key question is whether the target VM must remain paused for the duration of a single VI operation (such as a search of process memory for encryption keys), or whether reliable results can be obtained by spreading the operation across multiple periods during which the target VM is not scheduled for execution. Some operations may be sufficiently simple that they fit in the normal unscheduled periods, whereas other may require significantly longer access to the state of the VM, and unusually long periods in which the target VM is paused might be detectable by an attacker. A potential solution to this issue may be to quickly record the relevant portions of the VM state, so the analysis can be conducted over a longer time period, while allowing the VM to continue operation, thus modifying the original state but not the copy. These and other emerging questions provide for a rich research arena in VMI.

IV. Conclusions

Virtual Machine Introspection may help the digital forensics community address some of the limitation of current investigative techniques, particularly as the use of virtualization becomes more commonplace. However, the results of digital forensics investigations can have serious and significant legal and societal consequences, and as such new tools and techniques should be introduced into this domain very carefully. This paper describes several open problems which will determine how applicable VMI really is to digital forensics, and help to determine the reliability of results obtained using VMI techniques. Identified areas for future research include virtual machine introspection tool development, applications of virtual machine introspection to non-quiescent virtual machines, virtual machine introspection covert operations, and virtual machine introspection detection.

References

[1] B. Carrier, File System Forensic Analysis, Addison-Wesley, Boston, MA (2005).

[2] T. Garfinkel, 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.

[3] K. Nance, M. Bishop, and B. Hay. Virtual Machine Introspection: Observation or Interference? IEEE Security & Privacy. Volume 6 Issue 5, Set.-Oct. 2008 pp. 32-37.

[4] B. Hay, and K. Nance. Forensics examination of volatile system data using virtual introspection. SIGOPS Oper. Syst. Rev. 42, 3 (Apr. 2008), 74-82. DOI= http://doi.acm.org/10.1145/1368506.1368517

[5] N. Petroni, A. Walters, T. Fraser, and W. Arbaugh, FATKit: A Framework for the Extraction and Analysis of Digital Forensic Data from Volatile System Memory, Digital Investigation Journal 3(4):197-210, December 2006

[6] A. Walters and N. Petroni, "Volatools: Integrating Volatile Memory Forensics into the Digital Investigation Process," Black Hat (Feb. 2007). Retrieved December 20, 2008 from www.blackhat.com/presentations/bh-dc-07/Walters/Paper/bh-dc-07-Walters-WP.pdf.

[7] P. Barham, B. Dragovic, K. Frasier, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," Proceefings of the 2003 Symposium on Operating Systems Principles (Oct. 2003).

[8] S. Peisert, M. Bishop, S. Karin, and K. Marzullo, "Principles-Driven Forensic Analysis," Proceedings of the 2005 New Security Paradigms Workshop pp. 85–93 (Sep. 2005).

[9] B. Carrier, Risk of Live Digital Forensic Analysis, Communications of the ACM 49(2) pp. 56–61 (Feb. 2006).

[10] P. Courtois, F. Heymans, and D. Parnas, Concurrent Control with Readers and Writers, Communications of the ACM 14(10) pp. 667–668 (Oct. 1971)

[11] M. Bishop, Computer Security: Art and Science, Addison Wesley Professional, Boston, MA (2002), sec. 24.4.1.1, pp. 702–703

[12] V. Neagoe and M. Bishop, Inconsistency in Deception for Defense, Proceedings of the New Security Paradigms Workshop pp. 31–38 (Sep. 2006).

[13] W. Cheswick, An Evening with Berferd, in Which a Cracker is Lured, Endured, and Studied, Proceedings of the Winter 1992 USENIX Conference pp. 163–173, (Jan. 1992).

[14] B. Hernacki, Bennett, J., and Lofgran, T. Symantec Deception Server: Experience with a Commercial Deception System, Proceedings of the Seventh International Symposium in Recent Advances in Intrusion Detection pp. 188–202 (Sep. 2004).

[15] J. Michael, On the Response Policy of Software Decoys: Conducting Software-based Deception in the Cyber Battlespace, Proceedings of the 26th Annual International Computer Software and Applications Conference pp. 957–962 (Aug. 2002).

[16] N. Rowe, Counterplanning Deceptions to Foil Cyber-Attack Plans, IEEE Workshop on Information Assurance pp. 221–228 (June 2003).

[17] C. Stoll, Stalking the Wily Hacker, Communications of the ACM 31(5) pp. 484–497 (May 1988).

[18] P. Ferrie, Attacks on Virtual Machine Emulators. Symantec Advanced Threat Research. Retrieved January 15, 2008 from http://www.symantec.com/avcenter/reference/ Virtual_Machine_Threats.pdf

[19] J. Rutkowska, Red Pill... or how to detect VMM using (almost) one CPU instruction, 2005. Retrieved February 15, 2008 from http://invisiblethings.org/papers/redpill.html.