Simulating Windows-Based Cyber Attacks Using Live Virtual Machine Introspection

Dustyn A. Dodge, Barry E. Mullins, Gilbert L. Peterson, James S. Okolica
Air Force Institute of Technology
Wright Patterson AFB, Ohio
{ **dustyn.dodge, barry.mullins, gilbert.peterson, james.okolica.ctr** } @afit.edu

Keywords: Virtual machine introspection, virtualization, Xen

Abstract

Static memory analysis has been proven a valuable technique for digital forensics. However, the memory capture technique halts the system causing the loss of important dynamic system data. As a result, live analysis techniques have emerged to complement static analysis. In this paper, a compiled memory analysis tool for virtualization (CMAT-V) is presented as a virtual machine introspection (VMI) utility to conduct live analysis during simulated cyber attacks. CMAT-V leverages static memory dump analysis techniques to provide live system state awareness. CMAT-V parses an arbitrary memory dump from a simulated guest operating system (OS) to extract user information, network usage, active process information and registry files. Unlike some VMI applications, CMAT-V bridges the semantic gap using derivation techniques. This provides increased operating system compatibility for current and future operating systems. This research demonstrates the usefulness of CMAT-V as a situational awareness tool during simulated cyber attacks and measures the overall performance of CMAT-V.

1. Limitations of Existing Analysis Techniques

Static analysis is commonly used by computer forensic investigators to extract valuable intelligence data. This procedure traditionally involves shutting down the system to prevent observer effects that might corrupt system data or trigger time-bomb attacks that detect probing and erase the contents of the hard drive. Then, an image of the disk is created and analyzed using static analysis tools [1] [2]. Though this approach produces valuable results, it does not capture dynamic system data such as random access memory (RAM), open network connections or active processes. Many types of malware exist that leverage volatile system memory [3]. As a result, live analysis techniques have emerged to provide a more complete picture of the system state.

Live analysis can be implemented through software-based monitoring applications (anti-virus, spyware, etc) [4] or hardware-based devices [5]. These approaches however have significant limitations. Software applications, whether installed on the target system or executed using an imported device (CDROM, USB, etc), cannot be executed without making changes to the system. These observer effects prevent investigators from obtaining an objective view of the system state. In addition, many types of malware hide themselves from being detected. If the target system is compromised by malware, data reported from the system itself might not be trustworthy. Live analysis that is detectable by malware is also a problem. Malware could be designed to cause increased damage if application-based monitoring systems are detected. This has been shown possible using hardware-based monitoring techniques [6].

Not only does live analysis apply in the realm of forensic analysis but in modeling and simulation (M&S) as well. Previous work in cyber attack M&S create environments for but do not focus on observing the system state itself. For example, work by Kuhl and Sudit uses a simulated computer network with synthetic network traffic to test IDS sensors [7]. Though this approach is useful to test network situational awareness, synthetic environments may not reflect real world operation [8]. To overcome these limitations, virtualization has been identified as a more realistic simulation environment for live analysis to occur.

2. Virtualization

Virtualization, originally introduced in the 1970's [9] has seen an emergence since 1990 due to advancements in operating system compatibility and hardware support [10]. Virtualization simulates multiple guest operating systems (OSs) enabling them to use the same physical hardware resources. The simulated guest OS, or virtual machine (VM), interfaces with the underlying physical hardware through a virtual machine manager (VMM). The VMM resides in privilege ring 0, a kernel-level domain defined by the x86 architecture [11]. Operating in this privileged domain, the VMM actively coordinates the use of hardware resources for each VM. Consequently, the VMM has complete oversight over the state of all guest VMs. The VMM is abstracted from the guest VMs providing VMM-to-VM as well as VM-to-VM isolation.

In light of these characteristics, virtualization has been identified as an efficient and contained simulation environment for cyberwarfare training [12] [13] [14]. Using a virtualized environment, realistic cyber attack simulations can be run to develop cyberwarfare strategies for detecting malware, analyzing its behavior and defeating current and future threats. However, effective malware analysis requires complete situational awareness of the dynamic VM state. As a result, research in live VM analysis or virtual machine introspection (VMI) has emerged.

3. Virtual Machine Introspection

VMI involves monitoring the state of a VM during execution and provides an inherently more secure environment for analysis to take place. Utilizing the VMM's privileged position, VMI allows isolation from user-level attack, complete VM oversight and the ability to intervene on VM activities. In spite of these advantages however, a significant challenge in VMI development is overcoming the semantic gap [15]. The semantic gap describes the disconnect between the raw data gathered by a VMI utility and the meaning of the data within the context of the VM. Though the VMM can access raw data from memory, interpreting the data provided is not as straightforward. OS-specific data structures, like page tables used to map logical addresses to physical for example, are not explicitly provided to the VMM. These data structures can vary between OS distributions and even between service packs. Several approaches have been used to overcome this inherent semantic gap. These approaches, along with other VMI characteristics, will now be discussed.

3.1 Related Work

VMI systems have been categorized by their level of VM interference, how semantic awareness is achieved, and the ability for VM playback [16]. This paper uses these categories as a guide for differentiating VMI utilities but with slightly different criteria.

In this paper, VMI interference is defined as any detectable change in state or operation of the system being monitored. A technique used by some VMI utilities is to place software hooks within the system to act as a tripwire to signal when specific events occur [17]. Other approaches have used process runtime modification to identify hidden rootkits within a process list [18]. These are effective approaches; however they are subject to observer effects. Any direct modifications to the VM contaminate the system state. Some less invasive approaches pause the VM for data consistency [19] or for creating VM checkpoints [20] [21]. Though it has yet to be shown, suspending the VM itself could cause abnormal execution timing or page fault events [22]. Non-interfering VM tools use purely passive VM monitoring [23] [24]. Though these preserve VM state, there are limitations to these approaches as well. Passive VMI systems rely solely on the user to respond to threats detected. Also, without suspending the system, there is no guarantee that by the time events are reported, the system state has not changed.

Semantic awareness can be achieved by observing events specific to either the software or hardware architectures. Observing software-specific architectures involves extracting data such as active processes, threads, register or user information. As such, this approach requires OS-specific semantic information. This information can be explicitly provided a priori via OS-specific offsets [23] or OS libraries [15]. Alternatively, semantic information can also be derived from data collected by the VMI utility (i.e., memory dump). However, even derivation methods detect patterns based on previous knowledge about a particular OS. Observing hardware architectures removes the need for OS-specific semantic information. These techniques focus on observing events at the microprocessor level such as page faults, interrupts, and I/O requests [20] [25] [26]. This approach is OS independent. This may seem like the ideal solution; however interpretation of hardware-level activity is notably more difficult. Without any user-level context for the hardware-level events observed, conclusions made about the system state are limited in scope [27].

Finally, some VMI systems provide event replay. This allows investigators to step through past VM execution to strategically analyze changes in system state. Replay applications use event logging [20] or checkpointing [21] to revert the VM to a previous state. This also allows investigators to alter the execution during replay and allows the VM execution to run under different scenarios. However, by altering the execution state, previous execution from that point forward is lost.

4. Methodology

The following sections outline the problem definition, approach and experimental design for this research.

4.1 Problem Definition

Many VMI systems provide forensic analysis techniques to provide situational awareness of the VM state. These techniques extract dynamic system data during cyber attack modeling and simulation. However, in order to provide meaning to the data, precompiled OS-specific semantic information must be provided. This limits the portability of VMI systems to particular OS distributions or service packs. Those VMI applications that derive OS semantics are mainly shown to operate on

Linux-based guest VMs. Others, that are Windows specific, target a specific data structure which offers a limited view of the dynamic system state. This research attempts to create a novel Windows-based VMI utility for monitoring cyber attacks that not only derives OS-specific semantic information, but provides multidimensional views of the live system state. In addition, all of the system's dynamic memory will be parsed without interfering with the VM itself. As such, the goals of this research are the following:

- Create a Windows-based VMI application
- Verify live-analysis functionality using various cyber attack scenarios
- Evaluate VMI performance and system overhead

4.2 Approach

The following sections describe the approach used for VMI development. The CMAT-V memory analysis tool is described then assumptions made are discussed.

4.2.1 CMAT-V

CMAT-V builds upon CMAT, a compiled memory analysis tool for static forensic analysis [28]. CMAT-V is a prototype VMI application designed to conduct live forensic analysis of Windows-based guest VMs. Though the static analysis techniques used are applicable for most virtualization software packages, CMAT-V is designed for compatibility with Xen [29] virtualization software. CMAT-V also uses a modified version of XenAccess [23] as a framework to interface between Xen and CMAT. Both Xen and XenAccess were chosen due their open source availability. Figure 1 shows the overall CMAT-V architecture.

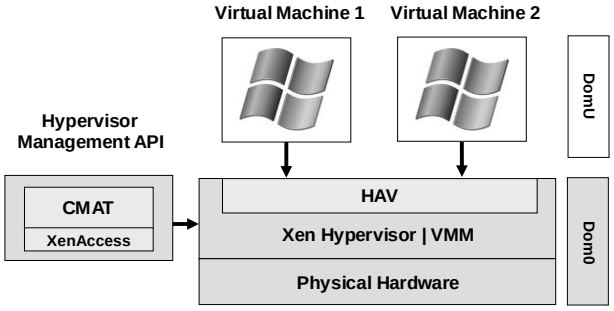

Figure 1 . CMAT-V Architecture

Xen uses privilege levels that correlate to particular Intel protection rings. Xen's privileged domain (Dom0) correlates to ring 0 while the user domain (DomU) correlates protection levels of rings 1-3. CMAT-V utilizes Xen's built in hypervisor management API to manage and monitor VM guests. The interface for the hypervisor management API resides in a trusted Dom0 guest. For this research the CentOS 5 operating system is used to manage user-level domains.

Xen supports both paravirtualization and full virtualization modes. In paravirtualization mode, the guest OS kernel is modified so that rather than making native system calls that must be translated by the VMM, hypercalls are made directly to Xen-supported physical hardware. This significantly reduces the overhead required by the VMM. For some proprietary guest OSs however, modifying the OS source code is not an option. As an alternative, full virtualization mode can be used. Full virtualization leverages hardware assisted virtualization (HAV) to allow the guest OS to use native system calls that are sent directly to the physical hardware. Intel-VT [11] and AMD-V [30] are the two most prominent HAV platforms in use today. Like paravirtualization, full virtualization also allows for reduced VMM overhead. Because CMAT-V targets proprietary Windows-based guests, Xen is run in full virtualization mode.

CMAT-V uses the XenAccess application programming interface (API) to configure and call Xen's built in VM management functions. In particular, CMAT-V uses a modified version of the XenAccess function `xa_access_pa()` which returns a mapped page and offset to a desired VM physical address. By default, XenAccess requires two user-provided files to call `xa_access_pa()`. The first file, `xenaccess.conf`, contains explicit VM configuration information like VM name, OS distribution and OS specific offsets to key data structures. This gives XenAccess exact locations to begin searching memory. The second file, `System.map`, contains OS specific data structure information. This information provides XenAccess a roadmap to strategically navigate through memory. In contrast, CMAT-V modifies XenAccess by removing the dependence on these user provided files. Instead, semantic information is derived from the memory itself. Subsequently, OS-specific symbol information is downloaded from the Microsoft Symbol Server [31].

After important semantic information has been established, CMAT-V leverages the static memory analysis techniques from CMAT to provide a multidimensional view of the state of dynamic system memory. Using these techniques CMAT-V is able to reconstruct the following:

- Logged on Users and Credentials
- Network Status Information
- Active Process Information
- Driver Information
- Open Files and Registry Keys

Unlike CMAT however, instead of accessing previously extracted memory dump data from a file, CMAT-V directly accesses the VM's dynamic memory. In addition, CMAT-V is able to simultaneously monitor multiple VMs on a single physical system. The system views created by CMAT-V also use cross-view validation [32] to identify inconsistencies in information that the VM reports and the information collected by VMI.

4.2.2 Assumptions

There are several assumptions made during this effort. First, it is assumed that the VMM itself has not been compromised. As such, any reported information from Dom0 is considered a trustworthy view of the VM system state. Also, though the same techniques can be applied for guest VMs running Linux it is assumed that the VM is using a Windows-based OS. As such, CMAT-V uses Microsoft's Symbol Server to retrieve updated OS information once the operating system distribution and version have been derived. Finally, it is assumed that malware within the VM is completely isolated from the VMM such that it cannot detect it is being monitored by a VMI application. If VMI is detectable, the malware running on the VM could then hide itself or simply shut down.

4.3 Experimental Design

Experiments are conducted on a Dell Latitude D630 laptop with an Intel Core 2 Duo T7300 processor, 2 GB of memory and a 120 GB hard drive. The processor includes Intel-VT technology which allows HAV mode operation. The VM guests are Windows XP SP3 with 512 MB of RAM. Once implemented, three tests which verify threat detection, quantify application performance and measuring CMAT-V impact on the overall system evaluate CMAT-V performance.

4.3.1 Threat Awareness

As previously discussed, the primary advantage of monitoring a VM from the hypervisor level is its complete and objective view of the system. This allows VMI utilities to uncover malware such as rootkits that hide execution activity from the guest OS [3]. For example, many Windows rootkits (e.g., direct kernel object manipulation (DKOM) rootkits) can be designed to modify a data structure called the executive process block or EPROCESS block to hide their existence from the user. EPROCESS uses a linked list to keep track of active processes. This linked list is shown in Figure 2.

Within the EPROCESS block is another block called KPROCESS that contains information needed to schedule the process for execution. When Windows utilities such as task manager are run, the EPROCESS list is searched to retrieve a list of all active processes.

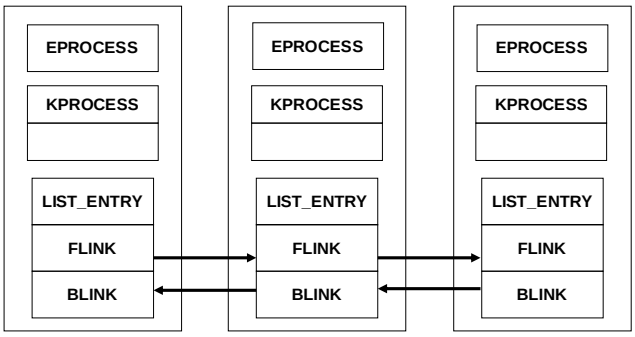

Figure 2. EPROCESS Structure

In order for a rootkit to remove a process from EPROCESS list, the links between the preceding and following process are modified to skip the target process altogether. This is shown in Figure 3. With the rootkit in place, utilities that rely on the EPROCESS linked list can no longer detect that the process is running.

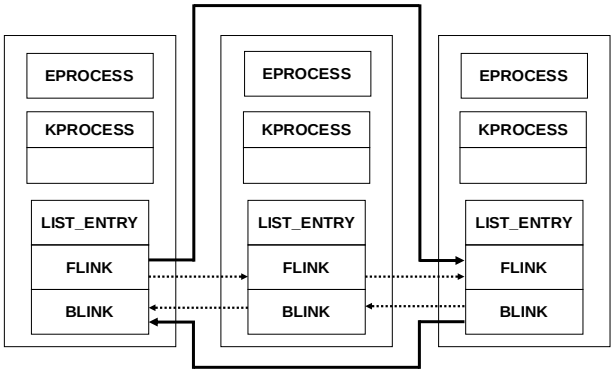

Figure 3. EPROCESS Rootkit

CMAT-V does not use the EPROCESS list as its primary method for process detection. Instead, it searches memory for known string patterns that indicate a KPROCESS block has been found. Once all valid KPROCESS blocks have been found, it can reconstruct the process list, exposing hidden processes. To simulate such an attack, a sample rootkit will be installed on a VM. The CMAT-V utility will then monitor the state of the VM. Based on reported information, classifications will then be made on CMAT-V's ability to provide evidence of the attack to the user.

4.3.2 Application Performance

A series of tests will be conducted to measure the performance of specific CMAT-V functions. Using a timing measurement such as the POSIX `clock_gettime` function, the number of CPU cycles required for CMAT-V functions will be measured. The

functions tested include initialization, OS detection and memory scanning routines. Using this information, the performance of CMAT-V will be characterized to quantify the overhead required for particular routines within the program. It is hypothesized that once semantic information has been derived and initialization has been completed that CMAT-V will be able to more quickly access known data structures.

4.3.3 System Impact

To evaluate the overall system overhead of running CMAT-V, several benchmarks will be run. These tests will use the Phoronix Test Suite [33] to conduct several Linux-based benchmarks within Xen's hypervisor management API. Phoronix was chosen because it is open source and contains a wide variety of benchmark workloads. First, the system will be configured to minimize all unessential processes running in the background. Then, baseline benchmarks will be run in the Dom0 guest VM to characterize stand-alone performance. Those same benchmarks will then be run while CMAT-V is executing. After all benchmarks have been run, baseline and experimental timing measurements will be compared. Any decrease in performance will then be assumed to have been caused by CMAT-V operation.

5. Conclusion and Work in Progress

This paper described CMAT-V, a VMI utility for live memory analysis of a guest VM. CMAT-V provides situational awareness during simulated cyber attack scenarios. Using static forensic analysis techniques, CMAT-V derives semantically relevant information from an arbitrary Windows memory dump. This information is then reported to the user to classify the state of the system. A methodology for system development and performance analysis was presented.

Current work in progress is focused on integrating CMAT's memory parsing functions with those defined in XenAccess. This work will focus on creating a customized version of XenAccess such that configuration files are not required to access physical memory. Once it is shown that CMAT-V can call XenAccess mapping functions, it can implicitly derive the configuration information needed. Once this is completed, CMAT will be modified to call the XenAccess API instead of a memory dump file. Finally, focus will turn to program optimization, sample rootkit development and performance testing.

6. References

[1] Access Data. Forensic Toolkit. (Last Accessed: March 2010). http://www.accessdata.com/forensictoolkit.html

[2] Guidance Software, Inc. EnCase. (Last Accessed: March 2010). http://www.guidancesoftware.com/

[3] G. Hoglund and J. Butler, *Rootkits*. Stoughton: Pearson Education, Inc., 2006.

[4] Symantec Corporation. AntiVirus, Anti-Spyware, Endpoint Security, Backup, Storage Solutions. (Last Accessed: March 2010). http://www.symantec.com/index.jsp

[5] H. Carvey, *Windows Forensic Analysis*. Burlington: Syngress Publishing, Inc., 2009.

[6] J. Rutkowska, "Beyond the CPU: Defeating Hardware Based RAM Acquisition," COSEINC Advanced Malware Labs Presentation, 2007.

[7] M. E. Kuhl and M. Sudit, "Cyber Attack Modeling and Simulation for Network Security Analysis," in *Proceedings of the 39th Conference on Winter Simulation: 40 years! The best is yet to come*, Washington D.C., 2007, pp. 1180-1188.

[8] A. Mukosi, J. A. van der Poll, and R. M. Jolliffe, "A Virtual Integrated Network Emulator on Xen (viNEX)," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Rome, 2009.

[9] R. P. Goldberg, "Survey of Virtual Machine Research," *Computer*, vol. 7, no. 6, pp. 34-45, Jun. 1974.

[10] M. Rosenblum, "The Reincarnation of Virtual Machines," *Queue*, vol. 2, no. 5, pp. 34-40, Jul. 2004.

[11] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manuals. (Last Accessed: March 2010). http://www.intel.com/products/processor/manuals/

[12] K. Stewart, J. W. Humphries, and T. Andel, "Developing a Virtualization Platform for Courses in Networking, Systems Administration and Cyber Security Education," in *Proceedings of the 2009 Spring Simulation Multiconference*, San Diego, 2009.

[13] A. Futoransky, F. Miranda, and J. Orlicki, "Simulating Cyber-Attacks for Fun and Profit," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Rome, 2009.

[14] M. G. Wabiszewski, T. R. Andel, B. E. Mullins,

and R. W. Thomas, "Enhancing Hands-on Network Training in a Virtual Environment," in *Proceedings of the 2009 Spring Simulation Multiconference*, San Diego, 2009.

[15] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in *Network and Distributed System Security Symposium*, San Diego, 2003.

[16] K. Nance, B. Hay, and M. Bishop, "Virtual Machine Introspection: Observation or Interference," *IEEE Security and Privacy*, vol. 6, no. 5, pp. 32-37, Sep. 2008.

[17] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An Architecture for Secure Active Monitoring Using Virtualization," in *Proceedings - IEEE Symposium on Security and Privacy*, Oakland, 2008, pp. 233-247.

[18] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "VMM-based Hidden Process Detection and Identification using Lycosid," in *ACM/Usenix International Conference On Virtual Execution Environments*, Seattle, 2008, pp. 91-100.

[19] B. N. K. Hay, "Forensics Examination of Volatile System Data Using Virtual Introspection," in *ACM SIGOPS Operating Systems Review*, New York, 2008, vol. 42, no. 5, pp. 74-82.

[20] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen, "ReVirt: Enabling Intrusion Analysis through Virtual-Machine Replay," in *OSDI'02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, New York, 2002, pp. 211-224.

[21] A. Joshi, S. T. King, G. W. Dunlap, and P. M. Chen, "Detecting Past and Present Intrusions through Vulnerability-Specific Predicates," in *ACM Symposium on Operating Systems Principles*, Brighton, 2005, pp. 91-104.

[22] B. Hay, K. Nance, and M. Bishop, "Live Analysis: Progress and Challenges," *IEEE Security and Privacy*, vol. 7, no. 2, pp. 30-37, 2009.

[23] B. D. Payne, M. D. P. d. Carbone, and W. Lee, "Secure and Flexible Monitoring of Virtual Machines," in *Annual Computer Security Applications Conference, ACSAC*, Miami Beach, 2007, pp. 385-397.

[24] L. Litty, H. Lagar-Cavilla, and D. Lie, "Hypervisor Support for Identifying Covertly Executing Binaries," in *SS'08: Proceedings of the 17th conference on Security symposium*, Berkley, 2008, pp. 243-258.

[25] L. Litty and D. Lie, "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, 2006, pp. 6-11.

[26] S. T. Jones, A. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Antfarm: Tracking Processes in a Virtual Machine Environment," in *Proceedings of the 2006 USENIX Annual Technical Conference*, Boston, 2006, pp. 1-14.

[27] J. Pfoh, C. Schneider, and C. Eckert, "A Formal Model for Virtual Machine Introspection," in *VMSec'09: Proceedings of the 1st ACM Workshop on Virtual Machine Security*, Chicago, 2009, pp. 1-9.

[28] J. S. Okolica and G. L. Peterson, "A Compiled Memory Analysis Tool," in *Research Advances in Digital Forensics VI*. New York: Springer Science+Business Media, 2010, InPublication.

[29] J. N. Matthews, et al., *Running Xen*. Boston, USA: Pearson Education, Inc., 2008.

[30] Advanced Micro Devices Inc. AMD64 Architecture Programmers Manual. (Last Accessed: April 2010). http://support.amd.com/us/Processor_TechDocs /26569.pdf

[31] J. Okolica and G. Peterson, "Windows Operating Systems Agnostic Memory Analysis," in *Proceedings of the Digital Forensic Research Workshop Conference (DFRWS)*, 2010.

[32] Y. M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski, "Detecting Stealth Software with Strider Ghostbuster," in *International Conference on Dependable Systems and Networks*, Redmond, 2005, pp. 368-377.

[33] Phoronix Media. Phoronix Test Suite. (Last Accessed: April 2010). http://tests.phoronix-test-suite.com/