An Introspection-Based Memory Scraper Attack against Virtualized Point of Sale Systems

Jennia Hizver and Tzi-cker Chiueh

Department of Computer Science, Stony Brook University, Stony Brook, USA
{jhizver, chiueh}@cs.stonybrook.edu

Abstract. Retail industry Point of Sale (POS) computer systems are frequently targeted by hackers for credit/debit card data. Faced with increasing security threats, new security standards requiring encryption for card data storage and transmission were introduced making harvesting card data more difficult. Encryption can be circumvented by extracting unencrypted card data from the volatile memory of POS systems. One scenario investigated in this empirical study is the introspection-based memory scraping attack. Vulnerability of nine commercial POS applications running on a virtual machine was assessed with a novel tool, which exploited the virtual machine state introspection capabilities supported by modern hypervisors to automatically extract card data from the POS virtual machines. The tool efficiently extracted 100% of the credit/debit card data from all POS applications. This is the first detailed description of an introspection-based memory scraping attack on virtualized POS systems.

1 Introduction

One of the most vulnerable links in the payment operations chain is the Point of Sale (POS) system deployed at physical locations where sales transactions and payment authorization take place (such as retail and hospitality businesses). The majority of POS systems are Windows-based computers running POS applications. Therefore, securing these systems is no different than securing any other Windows host. However, POS systems represent high-value targets for attackers who are financially motivated because they contain valuable card data that could be sold on black markets to reap considerable monetary gains. Negligence in securing POS systems carries a high risk as illustrated by a number of high-profile POS system breaches that have occurred recently. These successful attacks targeted at POS systems whose underlying hosts were not properly secured and eventually caused loss of tens of millions of credit card account information from merchants and credit card processors [1].

To improve security in payment processing systems, the Payment Card Industry (PCI) Security Standards Council developed and released two security standards: the Payment Card Industry Security Standard (PCI-DSS) and Payment Application Data Security Standard (PA-DSS) [2]. The PCI-DSS and PA-DSS standards established stringent security requirements to safeguarding sensitive card data. The current version of the PCI-DSS specifies 12 major requirement areas for compliance with over 200 individual control tests. All entities that store, process, or transmit card data

are required to comply with the PCI-DSS to ensure that their computer systems are better protected from unauthorized exposure. Noncompliant entities receive monthly fines and lose their ability to process credit card payments. The PA-DSS was derived from the PCI-DSS and was intended specifically to deal with secure POS software development lifecycle. The PCI Security Standards Council requires merchants to use PA-DSS compliant POS applications for credit and debit card transactions. The most important security requirement of these two standards is to prohibit storage of unencrypted card data. As a result, simply breaching a payment processing system and downloading card data stored on its disks is no longer a viable option. However, attackers have developed new attack techniques to obtain card data from the volatile memory of POS processes. In these attacks known as RAM scrapers, following infiltration of a POS system the attacker sets up a persistent operating system service designed to dump a POS process's virtual memory at specified time intervals and use regular expressions to parse the memory dump to extract card data. Attackers typically focus only on a POS process's virtual memory, rather than the whole system's physical memory. This strategy allows for fast processing and avoids excessive disk usage, which may be flagged as a malicious activity. Because the card data of a transaction commonly resides in unencrypted form in the volatile memory of a POS process when it processes the transaction, the memory scrapper attack is relatively easy to construct as compared with vulnerability-based exploit code.

One of the first RAM scraper attack incidents was reported in late 2008 [3]. In the 2009 Data Breach Investigation Report, Verizon reported 15 most prevalent threats and their frequency [4]. RAM scrapers surpassed phishing attacks and were ranked #14 among cyber security cases investigated by Verizon's Investigative Response team. In the 2010 Data Breach Investigation Report, Verizon reported that the use of RAM Scrapers to capture sensitive data continued to increase [5].

As virtualization technology becomes prevalent in enterprise data centers, POS systems also start to run on virtual machines hosted on virtualized physical servers [6-8]. These virtualized POS systems enable new forms of memory scraper attacks. This paper describes a study that aims to develop techniques that automate memory scraper attacks against commercial POS applications running inside virtual machines, and measure the effectiveness of such memory scraper attacks against commercial POS applications. We expect these attacks to emerge as a serious threat once virtualization technology takes hold in the POS market because the aggregate trusted computing base (TCB) of typically virtualized servers, which includes the hypervisor and a privileged domain (e.g. Dom0 in Xen), is too large to be free of security vulnerabilities.

A memory scraper attack on a POS virtual machine may originate completely outside of the virtual machine, and is thus more difficult to detect. This is possible because of virtual machine (VM) state introspection capabilities provided by modern hypervisors, such as Xen and VMware's ESX. VM state introspection mechanisms allow one to examine a VM's state from another VM running on the same physical server, and are mainly used in security tools implementations, such as intrusion detection/prevention systems and malware detection applications [9-11]. VM state introspection enables security tools to actively monitor a VM's run-time state and events in a way that cannot be affected by the OS and applications running in the

monitored VM making the security tools isolated from attack code as powerful as kernel rootkits.

This paper describes the design and implementation of an introspection-based RAM scraper called V-RAM scraper, which is designed to extract card data from POS applications running on Windows-based VMs hosted on a Xen-based virtualized server. We have applied V-RAM scraper against nine commercial POS applications, and were able to extract card data from every payment processing transaction that passed through each application. To the best of our knowledge, this is the first successful demonstration of an introspection-based RAM scraper attack on virtualized POS systems.

2 Background

A traditional POS network contains a central payment processing server and a number of POS application terminals connected using standard client-server architecture. After a credit card is used at the POS terminal, the terminal connects to the central payment processing server in the merchant's corporate environment, which, in turn, provides payment card authorization (Figure 1A).

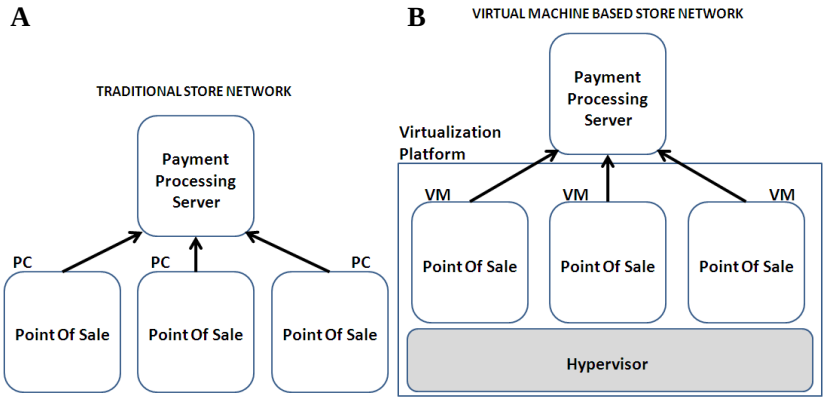

Fig. 1. (A) In the traditional POS network setup, each POS system is hosted on a separate platform in the store. The payment processing server is located in the corporate environment. (B) In a VM-based POS network setup, all POS systems are hosted on the same physical host.

Use of virtualization technology enables consolidation of the POS application hosts into a smaller number of physical machines, forcing POS applications to run inside virtual machines, as shown in Figure 1B. One example of such virtualization technology is Xen [12]. Xen was used in this study due to two important advantages: 1) it is an open-source hypervisor, and 2) it is capable of supporting multiple types of guest operating systems, including Windows and Linux. In Xen and similar virtual environments, out-of-VM RAM scraping attacks become even more powerful because if they could extract card data from one VM, they will be able to do so on all VMs running on the same physical server.

In Xen, the first VM, which boots automatically after the hypervisor is loaded, is called Dom0 domain. By default, Dom0 is granted special privileges for controlling other VMs including access to the raw memory of other VMs known as DomU domains. If an attacker could compromise Dom0 domain, she could then launch a RAM scraper attacker on virtual machines that run POS applications on the same physical server, as shown in Figure 2. Such an attack is possible because Dom0 is given the privilege to view the raw memory of all VMs on the same physical machine. Likewise, it is possible to mount a RAM scraper attack from a DomU domain if sufficient privileges are given to that domain to perform memory monitoring. For our implementation, we conduct the RAM scraper attack from within Dom0 because it already has the necessary privileges to view the raw memory of other VMs.

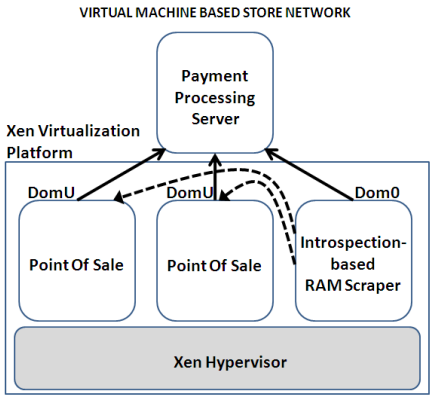

Fig. 2. Attack scenario: attacker compromises Dom0 and launches a RAM scraper attack.

Detailed description on how to compromise Dom0 is beyond the scope of this article. However, some of the common POS vulnerabilities are likely to exist on Dom0 in a virtualized POS environment because Dom0 domain is essentially a virtual machine running a full-blown operating system. There are several possible ways to gain access to Dom0 without proper authorization. Briefly, the top vulnerabilities contributing to POS, and, potentially, Dom0 compromises include missing or outdated security patches, use of default settings and passwords, unnecessary and vulnerable services on servers, and use of improperly configured remote access management tools. These vulnerabilities were commonly encountered and exploited by hackers in real life attacks against retail and hospitality businesses to obtain payment card data [5, 13, 14]. Exploitation of a vulnerable service running inside Dom0 could lead to a compromise and control of Dom0. As more and more functionalities are introduced to the Dom0 domain, the probability of such a compromise also increases. Another potential attack on Dom0 is through misuse of insecure remote management tools creating a “back door” in the system. Yet another possibility is to gain local access on one of the unprivileged VMs and from there to launch an attack on a service in the Dom0 domain. For example, a known vulnerability in code running in Dom0, CVE-2007-4993 [15], may be exploited from an unprivileged VM with the end result of this exploit is to execute chosen privileged

commands in Dom0. Accordingly, this study's assumption was that the access on Dom0 had been already gained by the attacker using one of the methods briefly described above.

Compromise of Dom0 and subsequent access to a VM's raw memory would not immediately compromise all payment data processed by the system. Only one card number is received per transaction, and as found by this study, it often exists in a POS process memory only a few seconds (sometimes milliseconds) before it is erased or overwritten upon completion of the transaction. Given the short data lifetime, an attacker has to acquire frequent VM memory images to ensure that all data entering the system is captured. Capturing full system memory images is not only time consuming, but may also increase the disk space usage alerting a system administrator of a suspicious activity, especially if numerous full memory snapshots from a number of VMs are taken concurrently.

Conversely, by looking only at the most probable segments of the memory that contain the data, the attacker could significantly reduce the memory search space and the hard disk usage and avoid false positives because number strings that look like payment card numbers but appear outside POS processes would be ignored. Therefore, if the attacker could fetch POS process memory pages exclusively, only small memory regions (as opposed to full system memory images) need to be searched for card data. However, determining which memory regions belong to the POS process may present a challenge. If a RAM scraper attack is conducted on the machine locally, the attacker has explicit access to high-level objects, such as processes, and can gather the required information. In a VM-based attack scenario, large amount of unstructured memory necessitates POS process identification before proceeding with payment data extraction. This study demonstrated that this challenge can be circumvented by leveraging VM introspection technique. The approach presented in this article effectively solved the problem of POS process identification significantly simplifying card data extraction.

3 The V-RAM Scraper Attack Tool

There are three steps in the V-RAM scraper tool. First, the tool maps the physical memory pages of the virtual machine on which the target POS application runs to its virtual address space, so that it can inspect and analyze their contents. Second, it identifies the portion of the mapped physical memory space that belongs to the target POS process or processes, so that it can focus on that portion only. Third, it searches the memory-resident portion of the target POS application process's virtual address space for possible payment card data.

Step 1: Mapping the Target VM's Physical Memory Pages

The Xen distribution provides a Xen Control library (libxc) for a Dom0 process to act on the DomU virtual machines, including pausing a DomU VM, resuming a paused DomU VM, reading a DomU VM's physical memory page, modifying a DomU's physical memory page, etc. Specifically, libxc provides a xc_map_foreign_range() function that is designed to map the physical memory space of a target DomU VM

into a Dom0 process's virtual address space so that the latter can easily manipulate the target VM's physical memory. The V-RAM scraper leverages this API function to map the physical memory pages of the VM on which the target POS application runs.

Because libxc is supported by the Xen hypervisor, neither the target VM nor the hypervisor require modification. Moreover, the target VM's physical memory space mapping to a Dom0 is transparent to the operating system or any user-level processes in the target VM, including the POS application. This means that it is difficult for a POS application running inside a VM to resist such memory mapping procedures.

Step 2: POS Process Identification

Instead of scanning the entire physical memory image of the target VM, the V-RAM scraper attempts to identify the user-level process running the target POS application, and scans only that process's physical memory pages for card data. Given a VM's physical memory mapped in the above step, the V-RAM scraper needs to apply VM state introspection [6] to make sense of it, particularly the high-level data structures embedded in the physical memory pages. This requires intimate knowledge of the target VM's operating system structure in order to bridge the so-called semantic gap [10, 11, 16] between the low-level memory pages and high-level kernel data structures, such as process list, page directories and tables, etc. It is non-trivial to reverse-engineer these guest OS-specific constructs, especially for a closed-source operating system such as Windows XP, which is the target of this project. Fortunately, a large body of knowledge about the Windows kernel's internal structure has been accumulated and documented by both black hats and white hats over the years. We leverage this knowledge and effectively solve the problem of POS process identification.

In Windows OS, the EPROCESS in-memory data structures are used to keep information about running processes [17]. To uncover the list of all running processes inside a VM, the V-RAM scraper extracts all the EPROCESS data structures by parsing the VM's physical memory pages mapped in the previous step. From an EPROCESS data structure, one can derive the corresponding process's attributes, such as the process' name and memory pages. All EPROCESS data structures are connected together in a double-linked-list and are stored in the address space of the System process. By recognizing the EPROCESS data structure of the System process and following the links embedded there, one could locate all other EPROCESS data structures.

The simplified representation of the EPROCESS data structure in Windows XP is shown in Figure 3. To discover the System process's EPROCESS data structure, one searches the physical memory pages for the known values of its 'DirectoryTableBase' and 'ImageFileName' fields. For instance, in Windows XP the System process' physical address of its Page Directory is always at 0x39000, which is recorded in the 'DirectoryTableBase' field, which is located at the 0x018 offset. The 'ImageFileName' field, which is located at the 0x174 offset, contains the value 'System'. Hence, the System EPROCESS data structure can be discovered by searching for byte strings of '00039000' and 'System' that are 0x15C bytes apart. Once the System process' EPROCESS data structure is known, all other EPROCESS data structures are readily available, as shown in Figure 3. Another well-known

approach to locate the EPROCESS list is to search for the special PsInitialSystemProcess symbol exported by the Windows kernel.

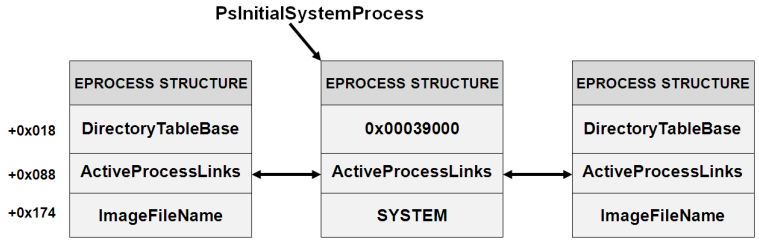

Fig. 3. The linked list of EPROCESS data structures. Each EPROCESS data structure has multiple data fields, which are located at pre-defined offsets with respect to the base of the structure.

Once the EPROCESS data structures are available, the V-RAM scraper identifies the target POS process by examining the name field of these EPROCESS structures. To identify the list of physical memory pages associated with the POS process, the V-RAM scraper first leverages the information in the 'DirectoryTableBase' field, which gives the base of the POS process's page directory, and then traverses the POS process's page directory and page table to find all the physical memory pages owned by the POS process.

The V-RAM scraper examines the target VM's physical memory pages using the user-space library XenAccess [18] that is specifically designed to facilitate VM state inspection.

Step 3: Card Data Extraction

Once the target POS process's memory pages are identified, the V-RAM scraper routinely searches them for card data using the following patterns. Payment card numbers are sequences of 13 to 16 digits. The card issuer is identified by a few digits at the start of these sequences. For instance, Visa card numbers have a length of 16 and a prefix value of 4. MasterCard numbers have a length of 16 and a prefix value of 51-55. Discover card numbers have a length of 16 and a prefix value of 6011. Finally, American Express numbers have a length of 15 and a prefix value of 34 or 37. Therefore, finding these card numbers in memory can be accomplished by searching for ASCII strings that match the following regular expression: ((4\d{3})|(5[1-5]\d{2})|(6011))-?\d{4}-?\d{4}-?\d{4}|3[4,7]\d{13}.

However, sequences of 13 to 16 digits with proper prefix values are not always card numbers. Each potential card number obtained by the above search procedure has to be further verified using the Luhn algorithm [19], which is a simple checksum formula that is commonly used to validate the integrity of a wide variety of identification numbers.

4 Testing Results and Analysis

We set up a virtualized server that uses Xen version 3.3 as the hypervisor and Ubuntu 9.04 (Linux kernel 2.6.26) as the kernel for Dom0. In addition, we set up DomU domains running Windows XP. Trial versions of nine PA-DSS compliant POS applications, which are listed in Table 1, were installed and tested in these DomU VMs. The V-RAM scraper was installed in Dom0 and launched to test if it can extract card data from these POS applications.

Table 1. The test results summary.

	Description	Card Data Detected	Transient Stack Data	Persistent Stack Data	Transient Heap Data	Persistent Heap Data
App #1	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction and ***after*** the transaction completion.	+	-	+	+
App #2	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction only.	-	-	+	-
App #3	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed for the duration of the transaction only.	+	-	+	-
App #4	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction and ***after*** the transaction completion.	+	-	+	+
App #5	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction only.	+	-	+	-
App #6	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction and ***after*** the transaction completion.	+	-	+	+
App #7	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction only.	+	-	+	-

	Description	Card Data Detected	Transient Stack Data	Persistent Stack Data	Transient Heap Data	Persistent Heap Data
App #8	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction only.	+	-	+	-
App #9	PA-DSS compliant POS application for Windows OS. Used by restaurant and retail businesses.	Yes, existed during the transaction only.	-	-	+	-

When testing each POS application, we invoked the V-RAM scraper tool and performed several card transactions using test card numbers, as shown in Figure 4, to extract these test card numbers during and immediately after each transaction using the tool. Because we are mainly interested in how effectively these POS applications hide card data when they perform transactions involving card numbers, a small number of test transactions are sufficient to expose their behaviors in this regard.

Following each card transaction, the V-RAM scraper captures a snapshot of the test POS process's physical memory pages every second, while letting the POS application continue to run as the memory snapshot is being taken. The performance overhead of memory snapshotting is largely unnoticeable. For each captured snapshot, the V-RAM scraper searches for any credit card number patterns.

As expected, the V-RAM scraper was able to successfully identify all test card numbers in the memory snapshots of all test POS applications when the transactions are being processed, as shown in Figure 5. Moreover, the V-RAM scraper is able to identify other card related information including the card expiration date, CVV number, and the card holder's name within the same memory segment as the corresponding card number.

Based on the timings and the portions of the snapshots from which card data are extracted, we classify each card data extraction instance into four categories:

(1) Transient/Stack: The card data are uncovered from a stack region while the associated transaction is being processed.
(2) Persistent/Stack: The card data are uncovered from a stack region after the associated transaction is completed.
(3) Transient/Heap: The card data are uncovered from a heap region while the associated transaction is being processed.
(4) Persistent/Heap: The card data are uncovered from a heap region after the associated transaction is completed.

Fig. 4. We ran several card transactions against each POS application by entering transaction-specific information, such as sale amount, card number, expiration date, CVV number, card holder name etc.

```
00CA7910 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA7920 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA7930 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA7940 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA7950 34 35 35 36 31 35 36 33  37 32 38 33 33 37 39 38 4556156372833798
00CA7960 00 00 00 00 30 34 31 32  00 00 00 00 34 35 30 2E ....0412....450.
00CA7970 30 30 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00..............
00CA7980 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA7990 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA79A0 00 43 56 56 32 20 47 4F  4F 44 20 4D 41 54 43 48 .CVV2 GOOD MATCH
00CA79B0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA79C0 33 35 34 00 00 00 00 00  00 00 00 00 00 00 00 00 354.............
00CA79D0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA79E0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
00CA79F0 00 00 00 4A 6F 6E 20 4A  6F 6E 65 73 00 00 00 00 ...Jon Jones....
00CA7A00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
```

Fig. 5. Detailed information uncovered about a test card, including the card number (4556156372833798), the card expiration date (0412), the CVV number (354), and the cardholder's name (Jon Jones) were identified within the process memory.

The successful card data extractions the V-RAM scraper is able to perform against the nine test POS applications fall into category (1), (3) and (4). Category (2) is rare because memory words allocated on the stack are automatically freed and possibly overwritten when they are no longer needed. However, this is still possible in theory,

because some stack frames, e.g., the last stack frame used in a long function call chain, may never have a chance to be overwritten long after they are freed.

In contrast, memory words allocated from the global heap have a much longer life time, because application programs need to explicitly free them when they are no longer needed, but application programs rarely do so. As a result, card data stored on the heap exist for at least the duration of the associated transaction, which typically takes up a few seconds to complete, and in many cases continue to exist even after the associated transaction is completed, suggesting that many POS application developers did not explicitly de-allocate and sanitize these card data-containing heap memory words. Note that languages that support automatic garbage collection, such as Java, mitigate this problem somewhat but do not completely eliminate it, because card data-containing heap memory words need to be not only de-allocated, but also zeroed out.

5 Discussions

The goal of this study was to assess the vulnerability of commercial POS applications to introspection-based RAM scraper attacks. These attacks proceed in a way that is completely transparent to the attacked POS system and thus represent a very attractive option for an attacker to quickly extract sensitive information from a large number of POS instances.

The test results demonstrated that the V-RAM scraper prototype could successfully extract all card data from the volatile memory of POS processes in fast, efficient, and un-intrusive manner while the card transactions are in progress. Card data capture was possible because the data were stored in an unencrypted form in memory and therefore susceptible to a window of vulnerability exploited by the memory scraper tool during the time when transactions were being processed.

One cause of this vulnerability is POS application developers failing to follow secure coding techniques. After a payment transaction is completed, it is mandatory to clean up all occurrences of unencrypted card data stored in stack and heap memory. However, this coding practice is not always followed in practice. In idle POS systems, this coding error leads to card data persistence in volatile memory for as long as several days greatly increasing the risk of card data exposure. Even when the software developers follow secure coding technique, it is still possible for the attack tools such as the V-RAM scraper described in this study to extract card data from running POS applications. This is possible because card data need to be present on the stack while a card transaction is being processed, which may easily take several seconds providing enough time for the V-RAM scraper to extract card data from a monitored POS VM.

Although Xen was used as the virtualization platform for this study, the V-RAM scraper attack tool is expected to be equally effective on any other virtualization platform supporting VM introspection capabilities. For example, VMware's ESX provides a VMsafe API with similar functionalities to the Xen Control library. While VM state introspection allows for good visibility and enables deployment of security solutions that are immune to tampering by malicious software inside monitored VMs, the same introspection capability also creates the threat of high precision attacks and opens new avenues for attackers to obtain payment data without breaching the POS application hosts with the attack tool described in this article as one example.

This security threat associated with VM introspection is not addressed by the current version of the industry security standard (PCI DSS). Adoption and inclusion of VM introspection capabilities into POS environments should be re-assessed in light of the feasibility demonstrated by the V-RAM scraper attack tool developed in this study.

Potential solutions to V-RAM scraper attacks are outside the scope of this paper. Nevertheless, there are several promising venues. One possibility is a programming system allowing a developer to annotate a piece of data structure as sensitive and for the compiler to automatically encrypt it when stored on the stack or heap. With this approach, it would be more difficult to detect sensitive data using plaintext-based scanning and matching. Another possibility is to introduce fine-grain access control mechanisms to control inter-VM communications (including VM introspection requests) according to a security policy manipulable only by the hypervisor [20]. Reduction in data persistence and information leakage can be also achieved by asking developers to follow privacy-protecting coding practices. One simple coding practice involves zeroing out heap memory locations that contain sensitive data before they are marked for de-allocation. Finally, a VM should be able to protect itself through detection of VM introspection [21].

6 Related Work

Previous studies on attacks targeting sensitive in-memory data and originating outside the host mainly focused on developing techniques for locating cryptographic keys within large blobs of memory, such as full system memory images. Shamir and Someren [22] described visual and statistical methods to efficiently search and identify encryption keys from system memory where the attacker required physical access to the system. Petterson [23] reviewed key recovery techniques from full-system memory images of Linux systems using the knowledge of encryption key holding data structures of open source crypto applications. While the research demonstrates that keys can be successfully located by guessing in-memory values of the variables surrounding the key, in attacks on closed source software, the attacker will not have access to the application source code and thus, can not apply the method. Halderman et al., [24] presented several attack scenarios exploiting DRAM remanence effects through acquisition of full system memory images to extract cryptographic key material. This attack also required physical access to the system. Ristenpart et al., [25] investigated the problem of confidential information leakage via side channels in a cross-VM attack, which neither required physical access to the system nor full-system memory acquisitions. Although in non-virtualized multi-process environment side channel attacks based on inter-process leakage have been shown to enable extraction of secret keys [26, 27], the cross-VM attacks presented in [25] were more coarse-grained than those required to extract cryptographic keys. In addition, the authors were not aware of any published extensions of these attacks to the virtual machine environment.

The attack scenario investigated in this study differed from the above examples because it did not require either physical access to the host or acquisition of full-system memory images. Moreover, the attack targeted short-lived transient data,

unique per each transaction, and therefore, required frequent and efficient interactions with the system memory to capture all the data passing through the system. This efficiency requirement was accomplished by leveraging the virtual machine introspection technique.

The term "virtual machine introspection" was introduced by Garfinkel and Rosenblum to describe the operation of a Livewire host-based intrusion detection system for virtual machines [9]. Subsequently, several other systems, such as Lares and VMwatcher applied VM introspection to monitor hosts running in virtual machines by reconstructing the semantics of the internal state within the VM with the goal of detecting malicious activity [10, 11]. Only limited details were given regarding the implementation of the introspection mechanisms employed in those systems. Conversely, XenAccess was the first open source project that described the introspection implementation in detail [28]. It also provided access to the programming APIs, therefore enhancing experimental research in IDS/IPS and malware detection.

Advances in VM introspection gave rise to its application in digital forensic analysis. Using VM introspection, unobtrusive live system analysis may be performed on the target virtual machine without changing the system state during the data acquisition process. The research on this topic is still ongoing, but there are several interesting techniques that have been developed for analyzing volatile memory in Linux virtual machines using VM introspection [29]. Despite the fact that most research has been conducted in Linux OS, analogous Windows OS based techniques can be developed utilizing methods borrowed from forensic analysis in non-virtualized systems. Our tool utilized these methods to generate a list of active processes, extract information relating to a specific process, and reconstruct the virtual address space of a process [30-32].

7 Conclusions

Payment card data processed by POS systems have been a covetous target for financially motivated attackers. Recognizing this threat, the payment card industry has issued multiple security standards to tighten the security requirements on POS systems. Although a step forward, these standards cannot prevent all possible attacks against the POS systems in the field. Memory scraping attack, in which the attacker scans the physical memory of POS application processes to extract payment card information, is particularly noteworthy because it aims directly at the most valuable data touched by POS systems. As POS systems start to run on virtualized platforms, newer forms of memory scraping attack become possible. This paper successfully demonstrates an introspection-based memory scraping attack that leverages VM state introspection capabilities offered by modern hypervisors such as Xen and VMware's ESX to extract payment card data from POS processes running inside VMs that execute on the same physical machine. The attack tool was applied against nine commercial POS applications resulting in extraction of 100% of card data. Although the proposed attack is contingent upon a successful break-in into the TCB of a POS virtualized server, the fact that such breaches have been reported previously suggests that this attack is not theoretical presenting a real threat.

Because the vulnerability exploited by the proposed memory scraper attack is a programming error, a compiler that can automatically encrypt payment card data and destroy them when the associated memory words are de-allocated, appears to be the best solution to this problem.

8 References

1. Chronology of Data Breaches, http://www.privacyrights.org/ar/ChronDataBreaches.htm
2. PCI Security Standards Council, https://www.pcisecuritystandards.org/
3. Evolution of Malware: Targeting Credit Card Data in Memory, https://www.trustwave.com/downloads/whitepapers/Trustwave_WP_Evolution_of_Malware_.pdf
4. 2009 Data Breach Investigations Supplemental Report, http://www.verizonbusiness.com/resources/security/reports/rp_2009-data-breach-investigations-supplemental-report_en_xg.pdf
5. 2010 Data Breach Investigation Report, http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf
6. Restaurant Chain Upgrades Systems and Cuts 2,000 Servers Using Virtual Machines, http://download.microsoft.com/documents/customerevidence/7146_jack__in_the_box_cs.doc
7. Bringing virtualization and thin computing technology to POS, http://www.pippard.com/pdf/virtualized_pos_whitepaper.pdf
8. MICROS Systems, Inc. Announces Deployment of MICROS 9700 HMS at M Resort Spa Casino in Las Vegas, http://www.micros.com/NR/rdonlyres/3E357BE8-70DB-468D-B9AB-68F0E784527F/2296/MResort.pdf
9. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proceedings of the 10th Annual Symposium on Network and Distributed System Security, pp. 191--206 (2003)
10. Payne, B. D., Carbone, M., Sharif, M., Lee, W.: Lares: an architecture for secure active monitoring using virtualization. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 233--247 (2008)
11. Jiang, X., Wang, A., Xu, D.: 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, pp. 128--138 (2007)
12. What is Xen?, http://www.xen.org/
13. Critical Vulnerabilities Identified to Alert Payment System Participants of Data Compromise Trends, http://usa.visa.com/download/merchants/bulletin_critical_vulnerabilities_041509.pdf
14. Top Five Data Security Vulnerabilities Identified to Promote Merchant Awareness, http://usa.visa.com/download/merchants/Cisp_alert_082906_Top5Vulnerabilities.pdf
15. Common Vulnerabilities and Exposures: CVE-2007-4993, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4993
16. Jones, S. T., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H.: Antfarm: tracking processes in a virtual machine environment. In: Proceedings of the 2006 USENIX Annual Technical Conference (2006)
17. Russinovich M.E., Solomon, D.A.: Microsoft Windows Internals. Microsoft Press (2005)
18. XenAccess Documentation, http://doc.xenaccess.org/
19. Luhn, H. P.: Computer For Verifying Numbers. In: Office, U. S. P. (ed), USA (1954)
20. Sailer, R., Jaeger, T., Valdez, E., Caceres, R., Perez, R., Berger, S., Griffin, J. L., Van Doorn, L.: Building a MAC-Based Security Architecture for the Xen Open-Source

Hypervisor. In: Proceedings of the 21st Annual Computer Security Applications Conference, pp. 276--285 (2005)
21. Nance, K., Bishop, M., Hay, B.: Investigating the Implications of Virtual Machine Introspection for Digital Forensics. 2009 International Conference on Availability, Reliability and Security (2009)
22. Shamir, A., Someren, N.: Playing hide and seek with stored keys. Financial cryptography, pp. 118--124 (1998)
23. Petterson, T.: Cryptographic key recovery from Linux memory dumps. In: Chaos Communication Camp (2007)
24. Halderman, J., Schoen, S., Heningen, N., Clarkson, W., Paul, W., Calandrino, J., Feldman, A., Appelbaum, J., Felten, E.: Lest we remember: cold boot attacks on encryption keys (2008)
25. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. Conference on Computer and Communications Security, pp. 199--212 (2009)
26. Percival, C.: Cache missing for fun and profit. BSDCan, Ottawa (2005)
27. Osvik, D. A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. RSA Conference Cryptographers Track (2006)
28. Payne B., Carbone M., W., L.: Secure and Flexible Monitoring of Virtual Machines. In: Proceedings of the Annual Computer Security Applications Conference (2007)
29. Hay, B., Nance, K.: Forensics examination of volatile system data using virtual introspection. SIGOPS Operating Systems Review 42(3), pp. 75--83 (2008)
30. Schuster, A.: Searching for processes and threads in Microsoft Windows memory dumps. In: Proceedings of the 6th Annual Digital Forensic Research Workshop, pp. 10--16 (2006)
31. Memparser analysis tool, http://www.dfrws.org/2005/challenge/memparser.shtml
32. An Introduction to Windows memory forensic, http://forensic.seccure.net/pdf/introduction_to_windows_memory_forensic.pdf