2011 30th IEEE International Symposium on Reliable Distributed Systems

Automated Discovery of Credit Card Data Flow for PCI DSS Compliance

Jennia Hizver and Tzi-cker Chiueh
Department of Computer Science
Stony Brook University
Stony Brook, USA
{jhizver, chiueh}@cs.stonybrook.com

***Abstract*—Credit cards are key instruments in personal financial transactions. Credit card payment systems used in these transactions and operated by merchants are often targeted by hackers to steal the card data. To address this threat, the payment card industry establishes a mandatory security compliance standard for businesses that process credit cards. A central pre-requisite for this compliance procedure is to identify the *credit card data flow*, specifically, the stages of the card transaction processing and the server nodes that touch credit card data as they travel through the organization. In practice, this pre-requisite poses a challenge to merchants. As the payment infrastructure is implemented and later maintained, it often deviates from the original documented design. Without consistent tracking and auditing of changes, such deviations in many cases remain undocumented. Therefore building the credit card data flow for a given payment card processing infrastructure is considered a daunting task that at this point requires significant manual efforts.**

This paper describes a tool that is designed to automate the task of identifying the credit card data flow in commercial payment systems running on virtualized servers hosted in private cloud environments. This tool leverages virtual machine introspection technology to keep track of credit card data flows across multiple machines in real time without requiring intrusive instrumentation of the hypervisor, virtual machines, middleware or application source code. Effectiveness of this tool is demonstrated through its successful discovery of the credit card data flow of several open and closed source payment applications.

Keywords-virtual machine; payment system; card data flow; compliance;private cloud

I. INTRODUCTION

Credit cards are commonly used in payment transactions worldwide. Payment systems operated by merchants and financial institutions executing the transactions accept card charges and provide card processing service. These payment systems thus make high-value targets for financially motivated cyber attackers because the valuable card data they contain can be sold on black markets for considerable monetary gains. Consequently, negligence in securing payment systems increases the risk of credit card data loss as illustrated by a number of high-profile payment system breaches that have occurred in the recent years [1]. These successful attacks caused loss of tens of millions of credit and debit card account information and resulted in significant financial liabilities to the owners of these payment systems.

To tighten the security in payment systems, the Payment Card Industry Security Standards Council developed and released the Payment Card Industry Security Standard (PCI-DSS) [2]. The PCI-DSS standard established stringent security requirements to safeguard sensitive payment card data. All entities that store, process, or transmit card data are required to comply with the PCI-DSS to ensure that their payment systems are better protected from unauthorized exposure. Noncompliant entities receive monthly fines and eventually may lose their ability to process card payments.

The key pre-requisite for PCI DSS compliance is construction of the card data flow diagram for a payment processing network. That is, a merchant must determine precisely how card data flow through its payment systems from their inception, where they traverse the network, and where they reside. This discovery process and the resulting card data flow diagram help merchants understand which IT equipments in its organization touch the card data so as to tighten the security of these IT equipments according to the PCI DSS compliance requirements. A card data flow could start from a payment card swipe at a store, or a card number input by a user into an E-commerce web site, and consists of all intermediate stops in a merchant's IT network at which the card information is examined or processed. Data loss prevention (DLP) tools are a class of tools that are designed to identify network packets, files or database tables that contain sensitive information such as personal medical records or credit card numbers. DLP tools are highly effective when dealing with unencrypted data, but are largely powerless when credit card data are encrypted during their processing and exchange, as in the case of PCI DSS-compliant payment systems. Today, no known tool exists that could automatically discover the card data flow of a potentially distributed payment system. Therefore, currently the only solution to this problem is manual card data flow reconstruction based on outputs from DLP tools and network sniffers, and system design documents. As expected, such manual efforts are extremely time-consuming and labor-intensive, because the required information is difficult to obtain and often spread across a variety of IT elements and applications.

The goal of this research is to develop a tool that is capable of automatically and accurately extracting the card data flow of payment systems that run on virtualized servers

1060-9857/11 $26.00 © 2011 IEEE
DOI 10.1109/SRDS.2011.15

hosted in private cloud environments. We focus on payment systems running on virtualized servers because virtualization technology is quickly rising to predominate in enterprise data centers, and many payment systems start to run inside virtual machines hosted on virtualized physical servers [3-5]. The design principles underlying the proposed tool are equally applicable to payment systems running directly on physical machines, but it is easier to implement them in virtualized environments because of the availability of virtual machine introspection technology.

The tool we developed, called vCardTrek, provides a coarse-grained system-wide view of the card data flow across individual VMs that are actually involved in card data processing. To identify the trajectory of the card data flow, a payment processing request is sent to the entry point of a credit card processing system and the vCardTrek tool is employed to determine the set of machines that are invoked and exchange network packets as a result of this request. Leveraging virtual machine introspection capabilities [6], each network flow is then correlated with its sending and receiving processes. vCardTrek searches the address spaces of these processes for the clear text credit card data as they travel from the entry-point process to other card data handling processes along the way. Even though credit card data may be encrypted, they are decrypted and operated on in these processes, and therefore the clear text version of credit card data exist in these processes' memory. Once the processes whose memory contains credit card data are found, the machines involved in the credit card data flow are readily identified.

Two assumptions were made when developing vCardTrek. First, each card data handling process processes each request in a synchronous fashion, i.e., it reacts to an input request immediately and does not queue it for later processing. Second, when applying vCardTrek to a network to discover the credit card data flow, the network is in a "quiescent" state in the sense that only test payment transactions are running through the payment system and false positive caused by multiple concurrent requests is unlikely. Both assumptions seem to hold for real-world payment processing systems.

We have tested vCardTrek against 4 commercial payment applications and successfully built the card data flow path for each of these applications. To the best of our knowledge, this is the first known tool that could automatically discover the credit card data flow of distributed payment applications running in virtualization environments. Our implementation does not require modifications to the hypervisor, VMs, guest OS, or payment applications themselves. The virtual machine introspection capability leveraged by vCardTrek already exists in modern hypervisors, such as Xen and VMware's ESX. Because vCardTrek addresses a real current user pain point in PCI DSS compliance, we expect the availability of this tool could significant decrease the efforts and costs in meeting the security regulations stipulated in the PCI DSS standard.

II. BACKGROUND

A. Payment Systems

A payment application consists of multiple components communicating with one another using synchronous requests. When a credit card is swiped through a card reader or submitted online through a web browser, a payment authorization request is processed in real time by a number of payment application components, which run on multiple machines and serve to verify credit card information and to return a success or failure code to the originating application. In the synchronous communication model, a sending process forwards each request to the next receiving process along the card processing path and blocks until the corresponding response is received. Once the payment application verifies that an input request's credit card information is accurate and sufficient funds are available in the account, the request is granted permission to proceed with the purchase. A payment information processing request could take up to a few minutes. Additional processing steps may be triggered after a payment request is authorized, such as submission of payment data to storage, marketing data collection, payment reconciliation and settlement etc. Fig. 1 shows the data flow of a typical production-mode payment information processing system. As the virtualization technology has started to sweep the enterprise IT space, we assume different payment application components run on distinct virtual machines.

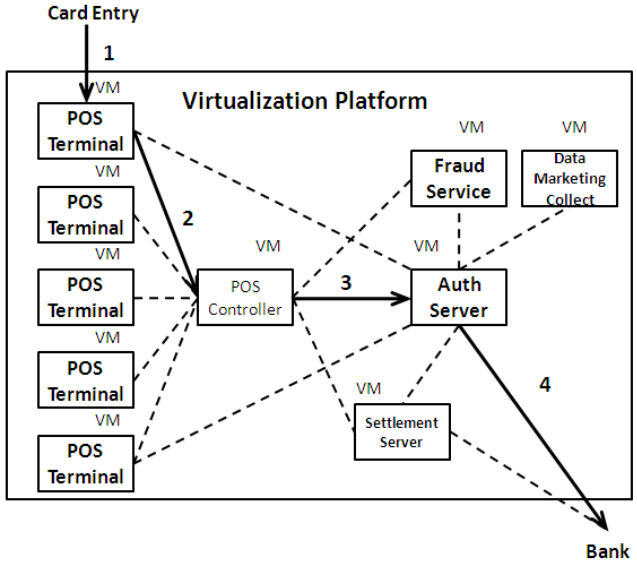

Figure 1. In a VM-based payment information processing system,, payment application components are installed on multiple VMs, which in turn could reside on one or multiple physical hosts. One of the possible card data flow paths is shown in solid lines. A card number is entered at a point-of-sale (POS) terminal in a store, travels to an authorization server that contacts the bank to obtain a payment authorization response. The dashed lines show other possible interactions among payment application components running in the virtual machines.

B. Virtual Machine Introspection

The goal of this research is to develop an automated tool that could discover and extract the credit card data flow from a possibly distributed VM-based payment processing application. The design and implementation of this tool is based on the virtual machine introspection (VMI) technology. VMI refers to the method of monitoring and analyzing the state of a virtual machine from the hypervisor. The term "virtual machine introspection" was introduced by Garfinkel and Rosenblum to describe a host-based intrusion detection system for virtual machines [6]. Advances in VM introspection gave rise to its application to digital forensic analysis. Using VMI, unobtrusive live system analysis may be performed on a target virtual machine without interfering with the target system's operation in any noticeable manner. Although most previous VMI-related research has been conducted for VMs running Linux OS [7], the same technique can be equally effectively applied to Windows OS-based VMs if methods borrowed from Windows machine forensic analysis are properly adapted. Our tool utilized these methods to generate a list of active processes, extract useful information associated with a specific process, such as its virtual address space, list of open network sockets [8-10], etc.

III. SYSTEM DESIGN

A. Requirements

The development of vCardTrek is driven by the following requirements, which are derived from analyzing card data flows in real-world production environments:

- Effectiveness of vCardTrek does not require any modifications on the hypervisor, the guest OS, or the payment applications in the target virtualized payment application system.
- vCardTrek must be agentless, in that no additional software needs to be installed on the VMs on which the target payment application system runs.
- vCardTrek does not make any assumptions on the internal operations of the target payment application system being tracked other than the following: (1) The target application runs on a virtualized environment, and (2) Credit card numbers are transiently stored in memory in a particular form.
- vCardTrek imposes minimal performance overhead and thus could operate seamlessly in the background while the target payment application runs at full steam.

B. Virtualization Platform

The vCardTrek prototype described in this paper satisfies all the above requirements and is based on the Xen hypervisor [11]. Xen supports two types of virtual machines: unprivileged domains, called DomU domains, and a single privileged domain, called Dom0. In our design, we deploy vCardTrek in Dom0 and run payment applications in DomUs. By design, Dom0 is granted complete access to the entire state of the guest operating systems running in DomUs and can determine execution properties of DomUs by monitoring their run-time state through direct memory inspection using VMI interface provided by Xen, which allows for rapid memory analysis of DomUs satisfying the goals of application-independence and real-time, lightweight operation. Because other hypervisors, e.g. VMWare's ESX and Microsoft's Hyper-V, support similar VMI capabilities, vCardTrek is expected to be equally effective when ported over each of these hypervisors.

The credit card data flow of a distributed payment application is the trajectory of the credit card data flowing through the payment application components, which are assumed to run on DomU VMs. Moreover, because communications among payment application components may be encrypted, it is not always possible to detect credit card data from sniffed network packets. Given the entry point component of a target payment application, one needs to identify all other application components with which the first component directly or indirectly communicates, and then pinpoint those components that have the credit card data in their virtual address space. Therefore, the algorithmic outline of vCardTrek comprises the following high-level steps: (1) recursively tracing inter-VM communications starting from the entry-point VM that receives the test input request, (2) applying VMI to inspect the memory space of communicating VMs for the credit card number used in test input, and (3) reconstructing the data flow path based on the results from (1) and (2).

C. Main Components

vCardTrek consists of a network agent and an introspection agent, both running in the Dom0 VM of all physical machines on which the target payment applications are installed (Fig. 2). The network agent tracks inter-VM communications by examining the headers of network packets and constructs an inter-VM communication graph starting from the entry-point VM. This is possible because all network packets to and from a DomU VM must go through its associated Dom0 VM. The introspection agent searches the memory space of all VMs in the inter-VM communication graph for credit card numbers, and builds the resulting card data flow path.

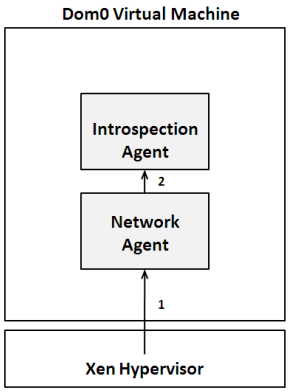

Figure 2. vCardTrek cosnists of a network agent that intercepts and analyzes network packet headers, and an intrspection agent that searches the memory space of communicating VMs for credit card numbers.

vCardTrek's network agent is a user-space program running in Dom0. As shown in Fig. 3, it intercepts network packets by hooking into Xen's network bridge between the physical interface card and virtual network interfaces (VNI) (in para-virtualized machines) or emulated network devices (in fully-virtualized machines). When the network agent intercepts a packet, it extracts its source and destination MAC addresses and port numbers. Given a quadruple of source MAC address, source port number, destination MAC address, and destination port number, the network agent determines the VMs that are involved in the corresponding network connection, and delivers this end point information to the introspection agent for further processing.

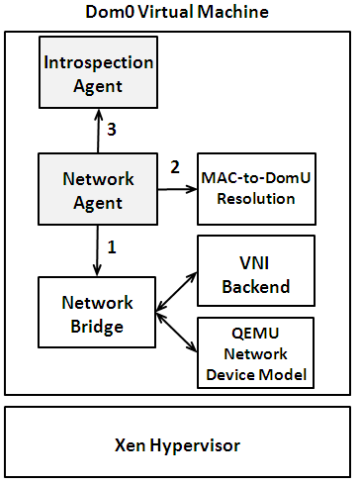

Figure 3. vCardTrek's network agent architecture. (1) Network agent taps into the network bridge in Dom0, extracts the ports and MAC addresses of the communicating VMs, (2) resolves the MAC address to the corresponding VM's ID, and (3) forwards the associated VM IDs and port numbers to the introspection agent.

vCardTrek's introspection agent is a user-space program running in Dom0 that receives the (src VM, src port, dst VM, dst port) information for each active network connection from the network agent, identifies the processes associated with each network connection, searches the memory of the identified processes for the credit card number used in the test input, and constructs the data flow path. From a VM ID and port number, the introspection agent applies VMI to examine the VM's kernel data structures containing socket and process information, as shown in Fig. 4, to identify the process in the VM that is bound to the port number. After identifying the process, the introspection agent maps the process's memory pages to Dom0 and searches the memory pages for the test credit card number. The introspection agent applies the above algorithm, starting with the entry-point process, and continues recursively until the physical memory space of a destination process no longer contains the test credit card number.

IV. IMPLEMENTATION

We implemented a vCardTrek prototype using the Xen hypervisor and fully-virtualized (HVM) Windows-based VMs (payment systems predominantly run Windows OS), on which payment applications are installed. The vCardTrek prototype consists of two parts: the network agent and the introspection agent, whose implementation details are described below.

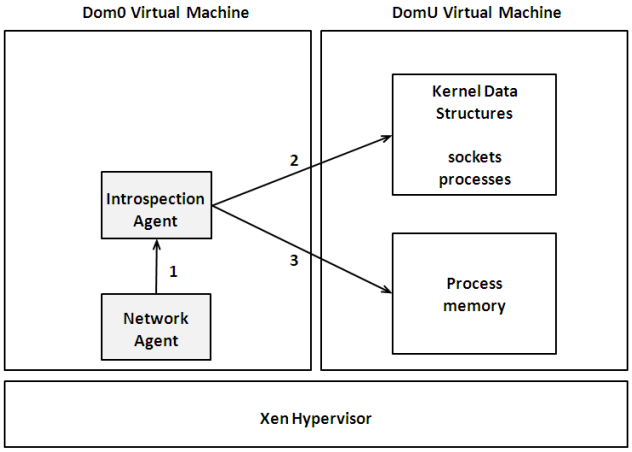

Figure 4. vCardTrek's introspection agent architecture. (1) Introspection agent obtains the DomUs names and ports numbers from the network agent, (2) finds the process ID bound to the specified socket on the sending and receiving VMs, and (3) searches the memory of the identified processes for the credit card number used in the transaction.

A. *Network Agent*

The network agent is implemented in C and makes use of the packet filtering tool *ebtables* to intercept all packets sent to or from DomUs. Ebtables is an open source utility that filters packets at an Ethernet bridge [12]. As of the 2.6 Linux kernel, the ability to perform bridge mode filtering using ebtables is natively included in the kernel and supported by default. Through command line arguments, ebtables is instructed to pass intercepted packets to the network agent using netlink sockets. Whenever the network agent receives a packet from ebtables, it parses the packet to extract (src MAC, src port, dst MAC, dst port) from the packet header. The src and dst MACs are then resolved to the DomU IDs using XenStore. In Xen, XenStore stores information about each DomU during its execution including the domain IDs and the corresponding MAC addresses. The network agent initiates an introspection request by sending the resulting (src DomU ID, src port, dst DomU ID, dst port) to the introspection agent. Introspection requests are processed in a multi-threaded fashion so that the network agent never blocks on these requests and allows the introspection agent to perform the VM analysis in parallel using separate threads.

The network agent maintains a table of all the (src MAC, src port, dst MAC, dst port) connections being currently analyzed by the introspection agent to avoid issuing redundant introspection requests while the request processing is in progress. Upon completion of an introspection request,

the corresponding connection record is removed from this connection table.

B. Introspection Agent

The main algorithm of the introspection agent consists of the following four steps. First, the agent maps the physical memory pages of the source and destination DomUs to Dom0, so that it can inspect their contents. Second, it parses the mapped pages to extract open sockets and identify the processes bound to the source and destination sockets of an inter-VM connection. Third, it identifies the portions of the mapped physical memory space that belongs to the identified processes, so that it can focus on those portions only, and searches these memory portions for the test credit card number used. In the final and fourth step, the card data flow is built.

1) Accessing DomU Memory: Xen offers low-level APIs to allow Dom0 to map arbitrary physical memory pages of a DomU to its memory space. These APIs operate on DomU IDs obtained from the network agent. XenAccess is a Dom0 user-space introspection library developed for Xen that is built upon the low-level APIs provided by Xen [13]. We leverage the PyXaFS file system, which is part of the XenAccess tool suite, to map physical memory pages of DomU's kernel inside Dom0. PyXaFS exposes the memory of a DomU as a regular file on the host and allows the introspection agent to read a live DomU's memory image as if it is a normal file.

2) Parsing Sockets and Processes Structures: Given a DomU's physical memory space, which is made accessible as a file, vCardTrek applies VMI to reconstruct the high-level data structures embedded in the DomU's kernel. This requires intimate knowledge of the target VM's operating system structure in order to bridge the so-called semantic gap [14-16] between low-level memory pages and high-level kernel data structures, such as network connections, sockets, 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 OS, 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 over the years. We leverage this knowledge and effectively solve the problem of network socket and process identification in a live VM's physical memory pages using Volatility Framework [17]. Volatility is an open source Python-based framework that was specifically designed to assist forensic investigators with the examination of volatile memory. The extraction techniques utilized by Volatility work for Linux, Windows, and OS X, and and include such capabilities as obtaining the list of running processes, open network sockets and connections, ability to dump a process' addressable memory, etc.

Applying the Volatility Framework to a DomU's mapped physical memory space allows vCardTrek to extract the identification of the processes bound to the network sockets connections whose port numbers and addresses are intercepted by the network agent.

3) Searching the Process Memory: Given a process ID obtained in the above step, the introspection agent traverses the process's page directory and page table to find the physical memory pages owned by the process and scans only that process's physical memory pages for the test credit card number.

After the introspection agent identifies a payment application's process being involved in possible credit card data communications, it scans the process's physical memory space for the test credit card number, before it is deallocated or overwritten. The introspection agent may need to scan the same process multiple times. The first scan examines every memory page in the process, but each subsequent scan only inspects those memory pages that are modified since the last scan. We exploited Xen's dirty page tracking capability originally designed for live migration to identify modified pages between consecutive scans. This incremental scanning approach drastically decreases the credit card number search overhead in subsequent scans. If no credit card number is found after a specified number of scans of a given process, the introspection agent assumes the process is not in the credit card data flow.

4) Credit Card Data Flow Reconstruction: To build up the credit card data flow, the processes whose memory contains the test credit card data and the communication connectivity among them are combined into a graph.

When two processes of a payment application communicate, there are three possible state combinations after searching their memory pages, as shown in Fig. 5(A): (1) The test credit card data found in the memory of both processes, (2) the test credit card data found in the memory of either process but not both, and (3) the test credit card data is found in the memory of neither process.

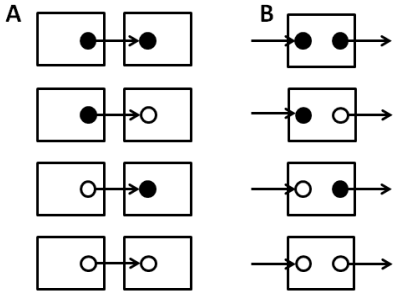

Figure 5. A rectangular box signifies a virtual machine. A black circle signifies the credit card number found in the memory of a process on a VM. An empty circle signifies no credit card number found in a process memory.The arrow indicates the direction of connection initiation, not traffic flow. (A) Possible states of two communicating processes at the time of communication. (B) Possible states of processes in a VM at packet receiving time and at packet sending (the same process may serve as the receiving and sending process).

Similarly, when the introspection agent scans a process's memory in a VM that serves as a credit card receiver and as a credit card sender, there are three possible state combinations, as shown in Figure 5(B). When the

introspection agent could not find the test credit card number in a process's memory, there are three possible cases. First, the process does not receive the test credit card data at all. Second, the process receives an encrypted version of the test credit card data, but does not decrypt it. Third, the process receives the test credit card data either in clear text or in encrypted form, but the introspection agent scans the process at an inopportune time, e.g. before the decryption of an encrypted card number or after the clear text card number is overwritten. To minimize the probability of Case 3, vCardTrek scans each communicating process multiple times.

Just because no credit card number is found in a process does not mean that the process cannot be part of a payment application's credit card data flow. For example, the process can receive an encrypted credit card number and pass it on to the next process without decrypting it. Therefore, the introspection agent has to scan all communicating processes regardless of whether the sending process contains the test credit card number.

V. EVALUATION

To demonstrate the utility of vCardTrek, the tool was tested on 4 payment applications, three e-commerce shopping carts and a point-of-sale system. These applications were written in different programming languages, used different application platforms, and employed different encryption techniques to protect credit card data. Although the number of hosts participating in the credit card data flow did not exceed three in any of the test cases, we observed all possible state combinations shown in Fig. 5. The experimental testbed consisted of a virtualized server that used Xen version 3.3 as the hypervisor and Ubuntu 9.04 (Linux kernel 2.6.26) as the kernel for Dom0. vCardTrek tool was installed in the Dom0. In addition, the virtualized server hosted several DomU domains running Windows XP.

A. osCommerce

osCommerce is a popular free e-commerce and online store-management software program [18]. According to the osCommerce site, there are over 12,000 online shops worldwide using the program. osCommerce is a PHP/MySQL-based system that can be set up as a three-tier architecture including a front-end browser, a web server, and a database server.

osCommerce was installed and tested in DomU VMs as shown in Fig. 6. IIS 5.1 web server, PHP 5.3.3, and osCommerce application were installed in the "VM-webserver" DomU to host the osCommerce code (with Credit-Card-with-CVV2 module) and process web application requests. MySQL version 5.1.52 was installed in the "VM-database" DomU to store the osCommerce application data. The Internet Explorer browser in the "VM-client" DomU was used to browse products and submit payment information.

When testing the osCommerce system, we selected several items for purchase and submitted credit card information at checkout. Because each card transaction is expected to consistently span the same set of VMs and applications along the data flow path, only one card transaction is sufficient to expose the osCommerce behaviors in this regard.

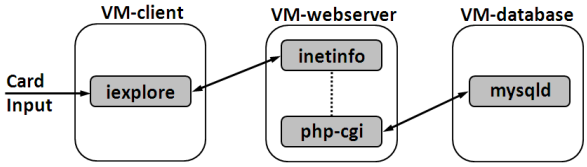

Figure 6. osCommerce data flow diagram.

Following the payment processing request, the vCardTrek tool determined the set of machines exchanging packets, identified the processes involved in these communications, and inspected the processes' memory for the credit card number used in the transaction, while allowing the applications to run throughout the analysis. vCardTrek successfully identified the test card number in memory of all the components along the card flow and built the card data flow path as shown in Fig. 6. Table 1 shows the average amount of time required to identify the credit card data flow for each test application.

TABLE I. THE AMOUNT OF TIME TO IDENTIFY THE FLOW

Application Name	Time, sec
osCommerce	7
AbleCommerce	9
X-Cart	9
CreditLine	8

B. AbleCommerce

AbleCommerce is a commercial shopping cart system with more than 10,000 stores worldwide using the program [19]. AbleCommerce is currently named on the PCI Security Standards Council's list of validated payment applications that have been assessed for compliance with PCI security standards. AbleCommerce is an ASP.NET/MSSQL-based system that can be set up as a three-tier architecture including a front-end browser, a web server, and a database server.

The trial version of AbleCommerce was installed and tested in DomU VMs as shown in Fig. 7. IIS 5.1 web server, .NET framework version 3.5, and AbleCommerce application were installed in the "VM-webserver" DomU to host the AbleCommerce code and to process web application requests. Microsoft SQL Express Server 2005 was installed in the "VM-database" DomU to store the AbleCommerce application data. The Internet Explorer browser in the "VM-client" DomU was used to browse products and submit payment information.

To build the card data flow, we followed the testing procedure as outlined in the osCommerce example. vCardTrek identified card data in the "VM-client" and "VM-webserver" memory. The test card number was not found in the "VM-database" memory, which suggested that either the credit card number was not submitted to the database or the credit card number was in an encrypted form when it settled in the database. The reconstructed card data flow path is shown in Fig. 7.

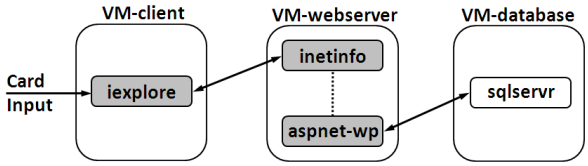

Figure 7. AbleCommerce data flow diagram.

C. X-Cart

The X-Cart e-commerce shopping cart software ranked in the top 10 among commercial shopping cart software applications used by online stores [20]. X-Cart is a PHP/MySQL-based system that can be set up as a three-tier architecture including a front-end browser, a web server, and a database server.

The trial version of X-Cart was installed and tested in DomU VMs as shown in Fig. 8. IIS 5.1 web server, PHP 5.3.3, and the X-Cart application were installed in the "VM-webserver" DomU to host the X-Cart code and to serve web application requests. MySQL version 5.1.52 was installed in the "VM-database" DomU to store the X-Cart application data. The Internet Explorer browser in the "VM-client" DomU was used to browse products and submit payment information.

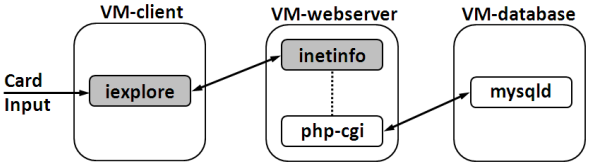

Figure 8. X-Cart data flow diagram.

To reconstruct the card data flow, we followed the testing procedure as outlined in the osCommerce example. vCardTrek identified card data in the "VM-client" and "VM-webserver" memory. The card number was not found in the "VM-database" memory, which suggested that either credit card number was not submitted to the database or the credit card number was in an encrypted form when it settled in the database. The reconstructed card data flow path is shown in Fig. 8.

D. CreditLine

CreditLine is a payment processing client-server application installed on point-of-sale systems [21]. CreditLine is currently named on the PCI Security Standards Council's list of validated payment applications that have been assessed for compliance with PCI security standards.

The trial version of CreditLine was installed and tested in DomU VMs as shown in Fig. 9. When testing the CreditLine system, we invoked the vCardTrek tool and performed a credit card transaction at the point-of-sale client application running in the "VM-client" machine. The "VM-client" established a connection with the "VM-server", and the "VM-server" further initiated a connection to an external IP. The test card number was not found in memory of the processes that established the connection between the "VM-client" and the "VM-server". However, the test card number was found in memory of the "VM-server"'s process that established the external connection request, thus suggesting that the credit card number was in an encrypted or encoded form when it exited the "VM-client" and entered the "VM-server". The reconstructed card data flow path is shown in Fig. 9.

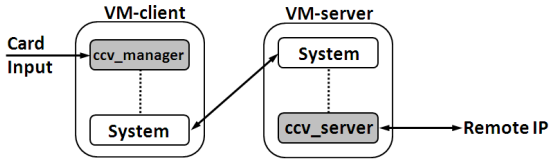

Figure 9. CreditLine data flow diagram.

VI. RELATED WORK

Several studies explored the problem of request processing path discovery in distributed systems. These can be roughly divided into "white box" and "black box" methods. White box, or instrumentation-based methods, require modification of middleware or applications to record events or inject unique identifiers that are used to reconstruct request processing paths [22-24]. Although capable of delivering precise measurements, white box methods usually have limited applicability in enterprise environments because they require knowledge, often source code, of the specific middleware or applications in order to enable instrumentation. For situations where instrumentation is not possible, path inference through black-box monitoring has also been considered [25-28]. "Black-box" monitoring does not require modification to the application or its framework. Unlike white box methods, black box methods analyze natively generated network packets and/or system logs, and infer request processing paths using statistical analysis. Platform agnostic black box methods are less precise but more suitable for enterprise environments where systems are composed of software from different vendors, usually without access to the application source code.

The novel tool described in this article (vCardTrek) bridges the two approaches combining the high precision of the white box methods with non-invasiveness and platform independence of the black box methods. Similarly to the white box approach, vCardTrek monitors unique IDs (credit card numbers) to track requests, but without the necessity of artificial injection of these IDs. Similarly to the black box approach, vCardTrek treats the machines as black boxes and does not require knowledge of the application components or modification to the OS, middleware, or application source code to enable request tracking. Therefore, vCardTrek represents a novel approach to resolve the problem of card data flow tracking and as such is expected to be of high practical value in industry applications.

Similarly to the unique identifier approach used in this study, Pauw et al. [28] described a mechanism for understanding the transaction flow in web services-based applications by locating application-specific conversation

identifies (such as order numbers) in the content of the messages passed between the application components in distributed applications. The approach treated the application modules as black boxes and relied on capturing the content of messages passed between the modules. The messages were assumed to have structured content. The captured messages were analyzed to reveal the correlations between instances of messages based on the extracted identifiers. Compared to our method, the scope of this tool is limited to web services-based applications only. This approach also has limited applicability for systems in which messages have encrypted content.

In another related study, vPath [29] technique was developed for request processing path discovery in virtualization environments. vPath used a virtual machine monitor to watch threads and network activity and infer request processing paths in para-virtualized systems that communicated using synchronous RPC messages. Similar to our tool, vPath prototype was also implemented in Xen hypervisor but the implementation required modifications to the hypervisor to identify threads that sent/received TCP messages and changes to the guest OS to deliver information about TCP connections. Compared to vPath, our tool works on both para-virtualized and fully virtualized systems and does not require modifications either to the hypervisor or to the guest OS.

VII. DISCUSSIONS AND CONCLUSIONS

Because PCI DSS compliance requires explicit identification of hosts that participate in credit card data processing, the PCI community is desperately in need of a tool that can automatically discover the credit card data flow of a legacy payment application system that is potentially distributed across multiple hosts. This paper describes the design and implementation of the first known tool designed specifically for such discovery called vCardTrek, and shows its effectiveness against three e-Commerce and one POS payment processing applications. A key feature of vCardTrek is that it can discover a distributed payment application's credit card data flow even if the communication between its component processes is encrypted. As for future work, we will extend vCardTrek to work for distributed payment applications that are installed on physical machines rather than virtual machines, and to work for payment applications that use asynchronous rather than synchronous communications.

REFERENCES

[1] "Chronology of Data Breaches, http://www.privacyrights.org/ar/ChronDataBreaches.htm."

[2] "PCI Security Standards Council, https://www.pcisecuritystandards.org/."

[3] "Bringing virtualization and thin computing technology to POS, http://www.pippard.com/pdf/virtualized_pos_whitepaper.pdf."

[4] "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."

[5] "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."

[6] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," Proc. Network and Distributed Systems Security Symposium, 2003, pp. 191-206.

[7] B. Hay and K. Nance, "Forensics examination of volatile system data using virtual introspection," ACM SIGOPS Operating Systems Review, vol. 42, Apr. 2008, pp. 75-83.

[8] A. Schuster, "Searching for processes and threads in Microsoft Windows memory dumps," Proc. of the 6th Annual Digital Forensic Research Workshop, July 2006, pp. 10-16.

[9] "Memparser analysis tool, http://www.dfrws.org/2005/challenge/memparser.shtml."

[10] "An Introduction to Windows memory forensic, http://forensic.seccure.net/pdf/introduction_to_windows_memory_forensic.pdf."

[11] "What is Xen?, http://www.xen.org/."

[12] "Ebtables, http://ebtables.sourceforge.net/."

[13] "XenAccess Documentation, http://doc.xenaccess.org/."

[14] B. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: an architecture for secure active monitoring using virtualization," Proc. IEEE Symposium on Security and Privacy, May 2008, pp. 233-247 .

[15] X. Jiang, A. Wang, and D. Xu, "Stealthy malware detection through VMM-based "out-of-the-box" semantic view reconstruction," Proc. ACM Conference on Computer and Communications Security, Oct. 2007, pp. 128-138.

[16] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Antfarm: tracking processes in a virtual machine environment," Proc. USENIX Annual Conference, 2006, pp. 1-14.

[17] "The Volatility Framework: Volatile memory artifacts extraction utility framework, https://www.volatilesystems.com/default/volatility."

[18] "osCommerce: Open Source E-Commerce Solutions, http://www.oscommerce.com/."

[19] "AbleCommerce: Featured Clients, http://www.ablecommerce.com/Featured-Clients-C49.aspx."

[20] "Welcome to X-CART: PHP shopping cart software & ecommerce solutions, http://www.x-cart.com/."

[21] "Payment Processing Software, http://www.911software.com/."

[22] "Application Response Measurement, http://www.opengroup.org/management/arm/."

[23] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: problem determination in large, dynamic internet services," Proc. International Conference on Dependable Systems and Networks, 2002, pp. 595-604.

[24] M. Schmid, M. Thoss, T. Terrain, and R. Kroeger, "A generic application-oriented performance instrumentation for multi-tier environments," Internaltional Symposium on Integrated Network Management, 2007, pp. 304-313.

[25] B. Sengupta and N. Banerjee, "Tracking transaction footprints for non-intrusive end-to-end monitoring," International Conference on Autonomic Computing, June 2008, pp. 109-118.

[26] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," Proc. ACM Symposium on Operating Systems Principles, 2003, pp. 74-89.

[27] B. Sengupta, N. Banerjee, A. Anandkumar, and C. Bisdikian, "Non-intrusive transaction monitoring using system logs," IEEE Network Operations and Management Symposium, Apr. 2008, pp. 879-882.

[28] W. D. Pauw, R. Hoch, and Y. Huang, "Discovering conversations in web services using semantic correlation analysis," Proc. IEEE International Conference on Web Services, July 2007, pp. 639-646.

[29] B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urhaonkar, and R. N. Chang, "vPath: precise discovery of request processing paths from black-box observations of thread and network activities," Proc. USENIX Annual Conference, June 2009