Non-intrusive process-based monitoring system to mitigate and prevent VM vulnerability explorations

Chun-Jen Chung*, JingSong Cui†, Pankaj Khatkar* and Dijiang Huang*
*School of Computing Informatics and Decision Systems Engineering
Arizona State University, Tempe, AZ, USA
{chun-jen.chung, pkhatkar, dijiang}@asu.edu
†Computing School, Wuhan University, Wuhan, China
jscui@whu.edu.cn

***Abstract*—Cloud is gaining momentum but its true potential is hampered by the security concerns it has raised. Having vulnerable virtual machines in a virtualized environment is one such concern. Vulnerable virtual machines are an easy target and existence of such weak nodes in a network jeopardizes its entire security structure. Resource sharing nature of cloud favors the attacker, in that, compromised machines can be used to launch further devastating attacks. First line of defense in such case is to prevent vulnerabilities of a cloud network from being compromised and if not, to prevent propagation of the attack. To create this line of defense, we propose a hybrid intrusion detection framework to detect vulnerabilities, attacks, and their carriers, i.e. malicious processes in the virtual network and virtual machines. This framework is built on attack graph based analytical models, VMM-based malicious process detection, and reconfigurable virtual network-based countermeasures. The proposed framework leverages Software Defined Networking to build a monitor and control plane over distributed programmable virtual switches in order to significantly improve the attack detection and mitigate the attack consequences. The system and security evaluations demonstrate the efficiency and effectiveness of the proposed solution.**

Keywords – Software Defined Networking, Attack Graph, Intrusion Detection, Countermeasure Selection, Virtual Machine Introspection.

I. Introduction

Security is one of the concerns that still make people think twice before migrating to the cloud. Virtualization introduces several attack surfaces for the cloud, like hypervisor, virtual machines (VMs), virtual network [1] to name a few. Among them, VMs are the most important resources for user and the most vulnerable target for the attacker. In traditional data centers, where system administrators have some control over the host machines, vulnerabilities can be detected and patched. However, patching known security holes in a cloud, where customers usually have the privilege to control the software installed on their VMs, may not work effectively and any action by the administrator might violate the Service Level Agreement (SLA) [2]. Furthermore, these vulnerable VMs are not only harmful to their users, but also pose a threat to other VMs. The challenge is to establish an effective detection and response system for accurately identifying vulnerabilities and malicious processes on users' VMs, rapidly detecting attacks from internal and external network, and efficaciously minimizing the impact of security breaches to cloud users.

Detection and removal of malicious code is not easy unless each VM is equipped with malicious code detection and monitoring tool. In worse case, malicious code will reside stealthy somewhere in the cloud system and such a hidden malicious code becomes a ticking bomb. If the malicious code is controlled by a remote attacker, s(he) is able to recruit other vulnerable VMs and launch a multi-step or coordinated Distributed Denied of Service (DDoS) attack. Goal for the attacker is to get control of a target VM. After controlling the VM, a remote attacker is able to monitor, intercept, and change the state and actions of other software on the system. The controlled malicious code is also able to hide itself and even disable the host intrusion detection system, which is a significant threat to a cloud. Therefore, malicious code detection and mitigation is a very important security tool for protecting VMs from being controlled by attackers.

Vulnerable VMs are ports of entry for a variety of security threats such as DDoS attacks, spamming, and malware distribution. Owners of compromised VMs are most often unaware that their systems are being used by someone else. Such a virtual machine whose security has been breached, either by a virus or other means, and consequently allows an unauthorized user to remotely control the system for malicious activities without the owner's knowledge is known as a *Zombie VM*. While the proliferation of Zombie VMs presents a substantial threat to the cloud system and network security, BotNets represent an even grave danger. In recent years, many resources have been dedicated towards the detection of compromised hosts in various domains, but there has been very little focus on the detection and prevention of zombies in the cloud environment.

Traditional methods to detect malware rely primarily on anti-malware agent installed on a VM to search the executable malicious code using pattern matching techniques. With such "in-the-box" agent-based approach, both detecting agent and detected results are visible and vulnerable to an attack, where the malware can temper with the result and disable the agent to hide the malicious code. To address this problem, more and more solutions have been proposed recently to place the malware detection engine outside the VM, then using *VM Introspection* (VMI) technology to detect and monitor the ma-

COLLABORATECOM 2013, October 20-23, Austin, United States
Copyright © 2013 ICST
DOI 10.4108/icst.collaboratecom.2013.254107

licious process in the VM [3][4][5]. Such "out-of-box" agent-free approach significantly improves the tamper resistance of malware detection engine [4]. However, introspection causes the so-called *semantic gap*, where the high-level semantic view of guest operating system are missing, e.g. active process, loaded kernel module, and system calls.

To address the problems described above, we propose a hybrid defense-in-depth intrusion detection framework based on our previous work NICE [6] to detect and monitor the vulnerability and attacks in the cloud. For better detection of attacks, our framework incorporates attack graph analytical model to describe the detected vulnerabilities in each VM and creates the vulnerability dependency based on VM's reachability in the virtual network. For the VM-based malicious process detection, we propose a non-intrusive agent-free detection and monitoring tool using VMI technology. In order to address semantic gap challenge, we reconstruct the semantic view to traverse thread dispatched database. We also reconstruct the complete process list of kernel and compare the user-level processes with the kernel-level process using cross-view technology to identify the hidden process.

Using Software Defined Networking (SDN) has been gradually adopted by commercial companies such as Citrix XenServer [7] and VMWare NSX [8]. SDN provides the ability to control the traffic in the virtual network for the QoS purpose, it also can be used to improve security and mitigate the attacks in a virtual cloud networking environment, for example, to build the basic firewalls, VPN, and network-based intrusion framework. We leverage SDN to build a monitor and control plane over distributed programmable virtual switches in order to significantly improve the attack detection and mitigate the attack consequences.

The proposed framework does not intend to improve any of the existing intrusion detection algorithms. Instead, we create this framework based on Xen virtualization platform and establish a system having following features:

- A light-weight network based intrusion detection engine in the dom0 of each cloud server for capturing and analyzing the network traffic.
- Attack graph analytical model for describing vulnerabilities and their dependencies in the cloud system.
- VM process monitor for monitoring and detecting hidden malicious processes in each VM using VMI and semantic reconstruction technologies.
- Countermeasure selection by matching and correlating alerts from intrusion detection engine, vulnerabilities in the attack graph and signal from VM process monitor.
- Deployment of virtual network reconfiguration-based countermeasure through network controller using Software Defined Networking (SDN).

This paper is organized as follows. Section II presents related work. System overview and models are described in Section III, section IV describes system design. Proposed mitigation and countermeasures are presented in Section V and Section VI evaluates our framework in terms of security and system performances. Section VII describes future work and concludes this paper.

II. Related Work

Detection of compromised machines currently takes place largely at host level and/or network level. At the host level, while anti-virus and anti-spyware systems are effective in catching and preventing the spread of known threats [9], majority of users do not keep these security software updated or properly configured. At the network level, IDSs and network firewalls fail to address already compromised vulnerabilities within the networks they protect. For example, when a system within the firewall becomes compromised by the careless actions of an authorized user, like allowing a trojan horse access to the internal network, then once such a system has been compromised, a personal firewall loses effectiveness because the attacker has already gained some level of control over the system.

Intrusion Detection System (IDS) attempt to detect and prevent the spread of compromised machines or their attacks by developing characteristic network traffic profiles, called signatures, and using them to identify attacks. However, similar to anti-virus solutions, IDSs are generally effective only as far as the malicious traffic pattern is detectable. In most cases, the intruder is able to continue to evade detection by blending malicious actions and activity with legitimate usage. Recent trends have shown intruders exploiting limitations and vulnerabilities within firewalls and IDS to better conceal the identities of zombies, thus making it harder to detect attacks. While system and network administrators attempt to combat this problem by addressing vulnerabilities in firewalls and anti-virus software as soon as they become known, zombies and their agents have been evolving even faster.

In a virtualized environment, such as a cloud system, it becomes easy for the attacker magnify the loss. In order to prevent the compromised VM from attacking vulnerable VMs due to the possible security hole cased by the resource sharing, the detection and monitor tools are necessary to secure each VM in a cloud system. For the VMM based malware detection, Livewire [3] proposed first concept of placing an"out-of-VM" monitor and applying VMI technology to reconstruct the semantic view of the internal structure of the VM. However, it can only reconstruct low-level VM states (e.g., disk blocks and memory pages). The high-level VM states (e.g., processes, kernel module, and files) still require an intrusive way to bridge the semantic gap. VMwatcher [4] is another "out-of-box" approach that overcomes the semantic gap created by the missing information about detailed internal view of the system by "in-the-box" approaches. To close the semantic gap, it applies a technique called 'guest view casting' and non-intrusively reconstructs the high-level internal VM semantic views from outside. However, it VMwatch only focuses on the malware detection in VMs and cannot monitor or detect the security status in the virtual network.

Antfarm [10] is an implementation of VMM-based introspection techniques that tracks the activities of processes in

VMs by monitoring low-level interactions between guest operating systems (OS) and their memory management structures. It can determine when a guest OS creates, destroys, or context-switches the processes without explicit information about the OS. Lycosid [11] is a VMM-based hidden process detection and identification service. It uses Antfarm to obtain a trusted view of guest OS processes, invokes a user-level program to obtain a untrusted view of these processes, and then uses cross-view validation principle to detect malicious hidden OS processes. The drawback of Lycosid is that it uses an intrusive approach to obtain the processes information from guest OS. Maitland [12] introduced a light-weight introspection technique in absence of hypervisor, that provides the same levels of within-VM observability. To address isolation challenges in out-of-VM approaches, Out-Grafting [13] was proposed. It allows fine-grained process-level execution monitoring. Our framework uses non-intrusive approach to monitor and detect the internal process of VMs and identifies the suspicious processes with the help of attack graph model.

To model possible vulnerabilities and their dependencies in a cloud, we use attack graph. An attack graph is able to represent atomic attacks in a network and describe possible attack paths for attackers to reach his/her goal, which we consider is to get root privileges on a particular VM. Various tools for attack graph generation and analysis have been proposed. One such tool was proposed by O. Sheyner *et al*. [14][15]. To tackle scalability issues with attack representations, P. Ammann *et al*. [16] assumed attacks to be monotonic in nature, which allows attackers not to backtrack. TVA (Topological Vulnerability Analysis) tool developed by Jajodia [17] was another attack graph generation tool but it required some initial input by hand. MulVAL [18] was proposed by X. Ou *et al*. in which they utilized Datalog representation of network state and attack scenarios. We adapt the MulVAL approach to create our version of attack graph, Scenario Attack Graph (SAG).

III. System Overview and Models

A. Design Goals and Assumption

We establish a hybrid intrusion detection framework to detect and monitor the malicious traffic in the network and malicious process in each VM using VMI technology. To achieve that, we have following design goals:

- The framework should be able to capture all of vulnerabilities in the cloud system and enumerate all possible attack paths after analyzing the vulnerabilities and their dependencies.
- The framework should be able to detect malicious processes in VMs immediately when the process is created, but such detection shall not be done by any piece of code running on VM.
- The framework should be able to select the optimal countermeasure and deploy it before the attacker takes the next exploitation step.

In this work, we assume that the hypervisor is trusted and secured, which means the hypervisor properly isolates

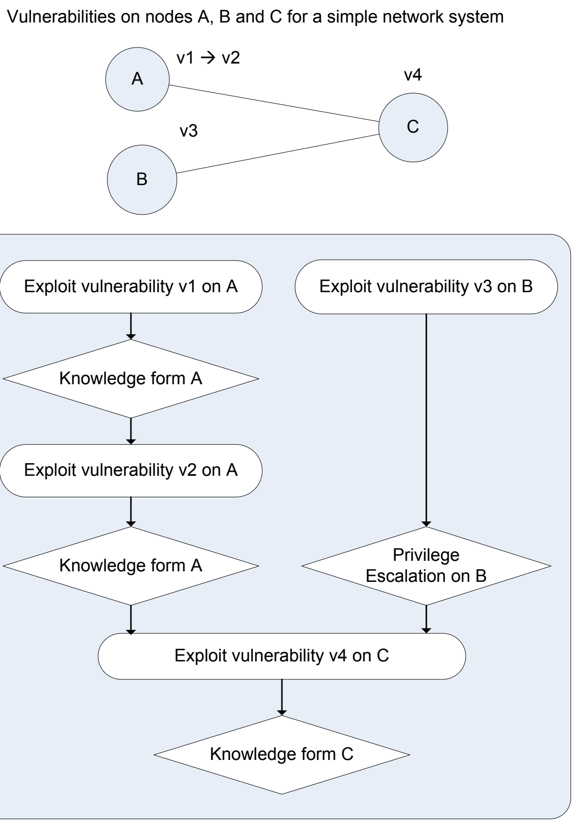

Vulnerabilities on nodes A, B and C for a simple network system
A
B
C
v1 → v2
v3
v4
Exploit vulnerability v1 on A
Knowledge form A
Exploit vulnerability v2 on A
Knowledge form A
Exploit vulnerability v3 on B
Privilege Escalation on B
Exploit vulnerability v4 on C
Knowledge form C

Figure 1. A simple attack graph example.

the resources and environment for the running VMs. The hypervisor is protected against any exploits launched by the attacker and the detection engine installed in the hypervisor is invisible to the attacker. We also assume that users are able to install vulnerable software and execute any malware or malicious code in their VM. System administrator cannot patch the software or remove the malicious code without users' agreement. However, the Cloud Service Provider (CSP) allows to block the traffic issued by such processes. Furthermore, the VM outage caused by cloud system reliability [19] is out the scope of this paper.

B. Attack Graph Model

An attack graph is a modeling paradigm to illustrate all possible attack paths in a network that can be exploited by internal or external attackers. It is a crucial model to understand threats and then to decide appropriate countermeasures in a protected network [20]. In an attack graph, each node represents a condition or an action. A condition node is a precondition and/or a consequence of an exploit. It represents a system configuration or an access privilege that should be true in order to exploit any other vulnerabilities. An action node is a step that attacker exploits an existing vulnerability in order to compromise a VM. It depends upon existence of one or more conditions along the path and is not necessarily an active attack since a normal protocol interaction can also be used for an attack.

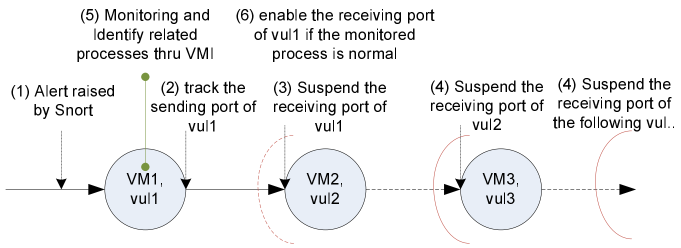

(5) Monitoring and Identify related processes thru VMI
(6) enable the receiving port of vul1 if the monitored process is normal
(1) Alert raised by Snort
(2) track the sending port of vul1
(3) Suspend the receiving port of vul1
(4) Suspend the receiving port of vul2
(4) Suspend the receiving port of the following vul..
VM1, vul1
VM2, vul2
VM3, vul3

Figure 2. suspend-check-forwarding model.

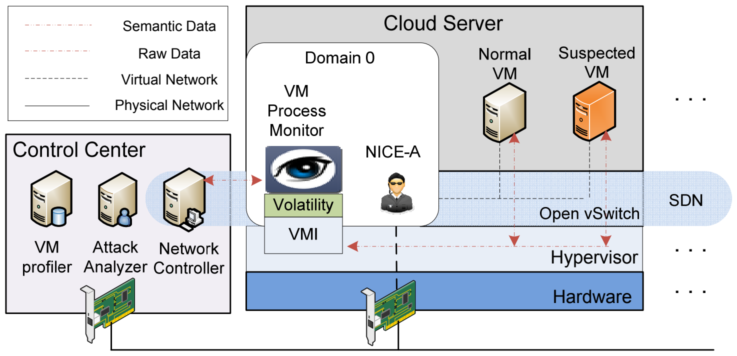


Figure 3. System Architecture.

As the attack graph lists all known vulnerabilities in the system and the connectivity information, one can get a whole picture of current security situation of the system. We can then see the possible threats and attacks by correlating detected events or activities with that depicted by the attack graph. Attack graph is thus helpful in identifying potential threats, possible attacks and known vulnerabilities in a cloud system. Once an event is recognized as a potential attack, attack graph tells important information about how the attacker can utilize that event the damage it can cause to other machines. With this information in hand we can apply specific countermeasures to mitigate a malicious event impact and take actions to prevent it from contaminating other virtual machines.

Fig. 1 shows a simple example of an attack graph. The left hand side of the figure shows a simple network topology with three VMs. VM A contains two vulnerabilities, $v1$ and $v2$. $v2$ can only exploited by an attacker if (s)he has exploited $v1$ and obtained the required privilege to exploit $v2$. VM B and C has $v3$ and $v4$ vulnerabilities respectively. The right hand side of the figure shows the generated attack graph corresponding to the network topology in the figure. Oval nodes represent attacker's action to exploit a vulnerability. Diamond nodes represent precondition of exploiting next vulnerability on the path and/or the consequence (post-condition) of an exploit. It shows that there are two possible attack paths to reach the goal which means to compromise VM C.

C. *Suspend-check-forwarding model*

The attack graph model enumerates all of possible threats in the system, and presents an analogy of a map to list different paths from source or attacker to destination or target. With the help of intrusion detection agent, we can identify where currently the attacker is on the attack graph. The alert raised by intrusion detection agent reveals that the traffic from a source to a destination with a certain protocol is suspicious. In order to detect the malicious payload, we propose a *suspend-check-forwarding* model to deter attacker's actions. Out system does so by detecting the infected process and stopping its communication with other processes in different VMs. We integrate the attack graph, VMI technology, and programmability feature of SDN to design our inspection model for tracking the available attack paths, monitoring the internal process of a guest VM, and suspending the traffic to a destination VM for further inspection.

The concept of this model is shown through Fig. 2. In order to verify the activities of internal process inside a possible victim and prevent the malicious code from further spreading out to infect other VMs, our system tracks the traffic that is generated from the port of the malicious process and whose IP address is the source of the alert message raised by a detection agent (step 1 and 2 in Fig. 2). After that, the system suspends the traffic on the receiving port of vulnerability on the destination VM after the suspicious node (step 3), and continually suspends the following interaction ports of vulnerabilities lying on the same attack path (step 4). Such suspended ports belong to applications susceptible to attacks. After suspending the traffic, monitoring on the infected VM begins to look for malicious processes (step 5). If a malicious process is detected, the traffic is blocked by the network controller in SDN, otherwise the traffic suspension is cleared and the receiving port on the destination VM is enabled (step 6).

IV. System Design

A. *System Architecture*

The proposed system is designed to work in a cloud virtual networking environment. It consists of a cluster of cloud servers and their interconnections. We assume that the latest virtualization solutions are deployed on cloud servers. The virtual environment can be classified as Privilege Domains, e.g., the dom0 of XEN Servers [7] and the host domain of KVM [21], and Unprivileged Domains, e.g., VMs. Cloud servers are interconnected through programmable networking switches, such as physical OpenFlow Switches (OFS) [22] and software-based Open vSwitches (OVS) [23] deployed in the Privilege Domains. In this work, we refer OFSs and OVSs and their controllers as to the Software Defined Network (SDN). The deployed security mechanism focuses on providing a non-intrusive approach to prevent attackers from exploring vulnerable VMs and use them as a stepping stone for further attacks.

The system architecture of our solution is illustrated in Fig. 3. The control center consists of a network controller, a VM profiler, and an attack analyzer.

A network intrusion detection engine NICE-A can be installed in either Dom0 or DomU of a XEN cloud server.

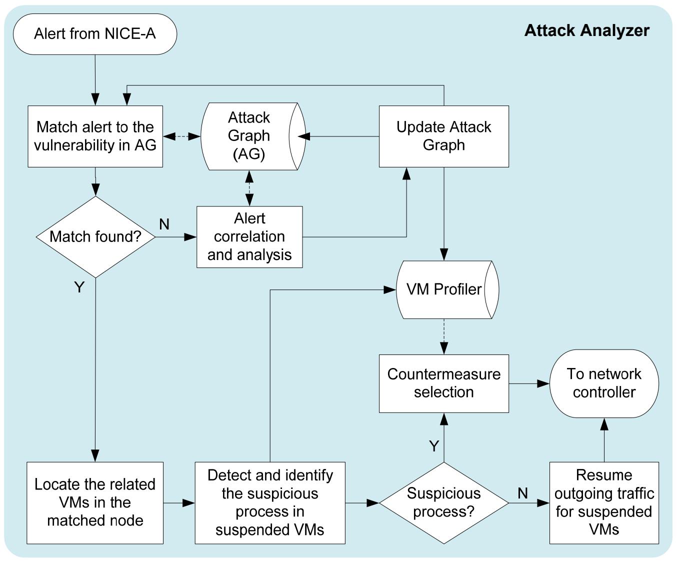

Attack Analyzer
Alert from NICE-A
Match alert to the vulnerability in AG
Attack Graph (AG)
Update Attack Graph
Match found?
N
Alert correlation and analysis
Y
VM Profiler
Countermeasure selection
To network controller
Y
Locate the related VMs in the matched node
Detect and identify the suspicious process in suspended VMs
Suspicious process?
N
Resume outgoing traffic for suspended VMs

Figure 4. Workflow of Attack Analyzer.

Its job is to capture and filter malicious traffic. Alerts from NICE-A are sent to control center upon detection of anomalous traffic. After receiving an alert, attack analyzer evaluates the severity of the alert based on the attack graph. It then initiates countermeasures through the network controller after deciding what countermeasure strategies to take.

As described in [6], countermeasures initiated by the attack analyzer are based on the evaluation results from the cost-benefit analysis of the effectiveness of countermeasures. The network controller initiates countermeasure actions by reconfiguring virtual or physical OpenFlow switches. We must note that the alert detection quality of NICE-A depends on the implementation of NICE-A that uses Snort. We do not focus on the detection accuracy of Snort in this paper. Dom0 consists of VM Process Monitor which uses VMI to monitor running processes on VMs.

B. Attack Analyzer

Attack Analyzer is a centralized information process center to process the security-related information and has the whole picture of the security status of the monitoring cloud server cluster. The major tasks of this component include collecting and processing information about the identified alerts, suspicious traffic and suspected processes from each VM Process Monitor, selecting the best countermeasure based on the knowledge of current attacks and system status, and sending the commands to the Network Controller for countering or mitigating the attack. These functionalities are realized by four subcomponents: Attack Graph analysis model, countermeasure selection, and VM profiler.

Figure 4 shows the workflow in the attack analyzer component. Alert received from NICE-A is checked against the vulnerability in the attack graph (*AG*). If a match is found, i.e. the vulnerability corresponding to the alert already exists in the attack graph then it can be regarded as a known attack matching with signature in the alert message. If no alert matches in the AG, then alert correlation and analysis is performed and AG is updated. However, for a matching alert, Attack analyzer locates the VM in the matched node based on the destination IP address in the alert. The VM Process Monitor then performs the inspection action on the corresponding VM using VMI to detect and identify the suspicious process with reference to the VM profiler. If a process is identified as suspicious, a selected countermeasure is applied by the network controller based on the severity of evaluation results. If the inspected process is found to be harmful, appropriate countermeasure is applied by the network controller, otherwise the outgoing traffic is resumed from the suspended VM.

C. Attack Graph

To keep track of all possible attack in the cloud, AA maintains an attack graph analysis model to analyze vulnerabilities and their relationship from all monitored VMs. The related tasks to the attack graph include constructing and updating the attack graph when a vulnerability is patched or the network topology is changed, correlating alerts with attack graph, predicting attacks, managing the VM Profiler, and selecting the optimal countermeasure. The attack graph is pre-constructed based on the following information:

- *Cloud system information*: it is collected from each PD. The information includes the number of VMs in each cloud server, the running services on each VM, and VM's Virtual Interfaces (VIFs) information.
- *Virtual network topology and configuration information*: NC collects this information, that includes virtual network topology, host connectivity, VM connectivity, every VM's IP address, MAC address, port information, and traffic flow information.
- *Vulnerability information*: it is generated by both on-demand vulnerability scanning and regular penetration testing using the well-known vulnerability databases, such as Open Source Vulnerability Database (OSVDB)[24], Common Vulnerabilities and Exposures List (CVE)[25], NIST National Vulnerability Database (NVD) [26], etc. Such scanning can be initiated by the NC and VM process monitor.

Many alert correlation techniques have been proposed [27][28][29] to reduce the false detection rate. In our framework, alert correlations and analysis are also handled by Attack Analyzer in the control center. This component has two major functions: (1) correlate alerts and integrate them into the attack graph model, (2) provide threat information or countermeasure to Network Controller for virtual network reconfiguration or further inspection.

D. Network Controller

Network controller is the main component to conduct the VM oriented (high level) countermeasure on the suspicious and malicious traffic based on the decision from Attack Analyzer. Network controller is the key component to support programmable network using OpenFlow protocol [30]. In our framework, each cloud server has a software switch, i.e., implemented by using Open vSwitch (OVS) [23] as the edge switch to handle all of traffic to and from VMs. The communication between cloud servers (i.e., physical servers) is handled by OFS. Both OVS and OFS are controlled by the Network Controller, allowing the controller to set security/filtering rules on both OVS and OFS. Network Controller is also responsible for collecting network information of current OpenFlow network, and provides inputs to Attack Analyzer to construct attack graphs.

E. VM Profiler

VM Profiler keeps tracking the security-related status of each VM. These profiles are necessary for the Attack Analyzer to identity suspicious events. We use three lists to record the security status of the processes for VMs in the cloud.

- *Frequently Compromised Process List (FCP)*: FCP is a list of processes related to well-known vulnerabilities in CVE, NVD, and OSVDB because these vulnerabilities are easy to be compromised by zero-day attacks, for example, IExplorer.exe, Acrobat.exe, WinRAR.exe, WINWORD.exe and so on. FCP is a public list for all VMs.
- *Blacklist (BL)*: BL is a list of malicious processes that have been identified by a PI from a VM. The process in BL is not allowed to establish a communication channel to other VMs in the cloud.
- *Whitelist (WL)*: WL is a list of processes that have not been identified as suspicious.

F. VM Process Monitor

Detecting the hidden malicious process is a very important task for securing the system. For the malicious process detection, traditional methods primarily rely on the detection and monitoring agent installed in the protected system. However, such systems have drawbacks like system integrity is not preserved, detection and monitor agent is easy to be attacked, and the correctness of the detection result cannot be guaranteed. To address these problems, placing the detection and monitor agent out of the protected VM is reasonable.

Virtualization Technology equips VM with three properties: isolation, encapsulation, and privilege. Due to these properties, it is easy to deploy the detection and monitoring agent in the virtualization platform. In this article, we focus on XEN virtualization platform only. We place the detection and monitor agent out of the protected VM, use VMI technology and semantic reconstruction to traverse thread dispatched database, and to reconstruct the complete process list of kernel and compare the user-level process with the kernel-level process using cross-view technology to identify the hidden process.

Figure 5 shows VM Process Monitor. We develop five submodules in this monitor: security console, daemon, VMI, semantic reconstruction, extractor and executor. Security console is an interactive console for administrators to setup the VM to be monitored and configure the monitor strategies and rules. These configuration messages will be sent to the daemon

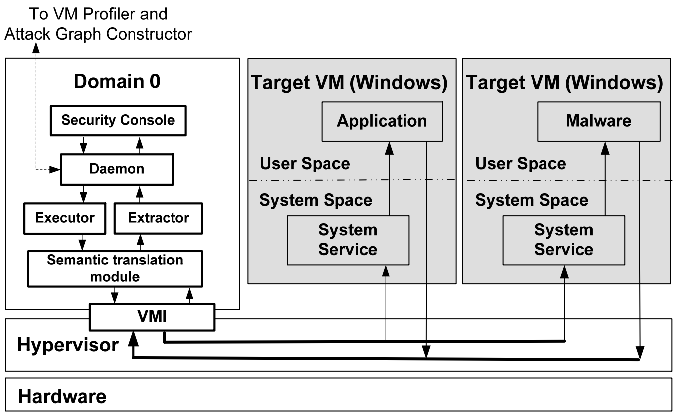


Figure 5. VM Process Monitor.

module with a command and displayed on the console to notify the administrators or serve as a log. Daemon, extractor and executor are all supporting functions for VMI and reconstruction modules to pass the messages and commands between VMM and VM Process Monitor.

VMI module is responsible for mapping the addresses between the application process' virtual address space in the VM and the actual physical address space through two-layer mappings. The first layer translates the Guest Virtual Address (GVA) to the Guest Physical Address (GPA). The second layer translates the GPA to the Host Physical Address (HPA). By reading the GVA from the outside of VM, VMI is able to read the content of the memory information in the VM through the translation from VM's GVA to VM's HPA.

In a virtualized platform, the virtual machine monitor (VMM) or hypervisor can only read VM's internal virtual hardware information, which is a large volume of unreadable raw data. It lacks meaningful semantic view of internal structures in the guest operating system and it is also hard to directly map the current running status of the system. This situation is often referred as "sematic gap" problem. The semantic translation module is the one for addressing the sematic gap issue. It is responsible for reconstructing the kernel-level process of a VM and recovering the VM's virtual hardware information from the information captured by the VMI.

For example, in order to retrieve the socket and network connection information of processes in a MS Windows VM, the VM process monitor locates the _ADDRESS_OBJECT and _TCPT_OBJECT data structures in VM's memory, which is pointed by _AddrObjTable and _TCBTable in PE (Portable and Executable) header of the kernel module tcpip.sys. The VM process monitor traverses the linked list of _ADDRESS_OBJECT and _TCPT_OBJECT data structures to identify all the in-use sockets and active network connections, and most importantly, the processes they belong to.

V. COUNTERMEASURE STRATEGIES

We consider the countermeasure strategies at both network level and host level. The network level countermeasures

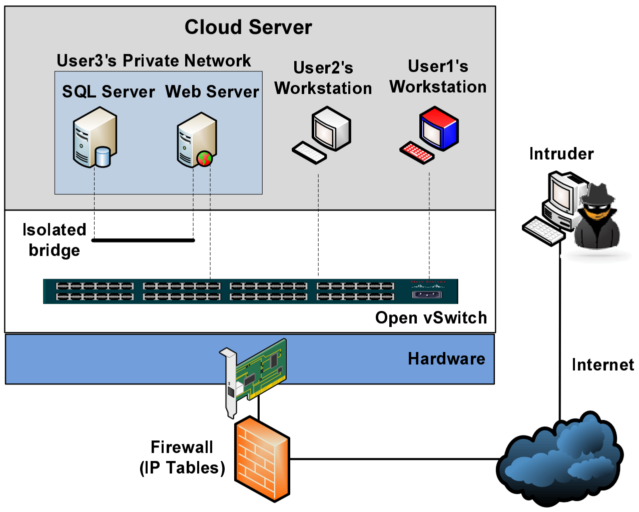


Figure 6. Network topology for the case study.

primarily rely on network reconfiguration strategies through SDN. It contains countermeasures such as traffic isolation, deep packet inspection (DPI), MAC address rewrite, IP address rewrite, network topology change, port blocking, and rate-limiting, as well as traffic drop, redirect, and suspend. In this work, we use port blocking and traffic suspension as network level countermeasure strategies for the proof of concept.

As for the host-level countermeasure strategies, our system mainly involves two levels of actions: VM-level network reconfiguration and process-level network reconfiguration. Virtual switch, i.e., the OVS in a XEN system, are the main component in the virtual networking of a cloud system for the VM connectivity. A VM in the XEN environment is connected to a virtual bridge in the OVS through the Virtual Interfaces (VIFs) attaching to the VM. VMs on different bridges are isolated at layer-2. Even the traffic between VMs on the same bridge is under the control of the virtual switch. Our framework utilizes this layer-2 traffic management capability to propose a VM-level network reconfiguration strategy. The VM-level reconfiguration strategy can either disable the suspicious VM's VIF to isolate it from other VMs or force the suspicious traffic redirect to an in-line mode intrusion detection system for further deep-packet inspection.

Process-level network reconfiguration provides a fine-grained and scrutinized control over the network connections. With the processes and sockets information of a VM reported from VM Process Monitor, the Network Controller is able to make OVS to isolate the traffic issued by the inspected processes or to redirect the traffic to an in-line mode intrusion detection system for further inspection before delivering to the destination. The advantage of the process-level network reconfiguration is that all other network services remain untouched while the activity of the suspicious process is monitored or isolated.

In summary, the host level countermeasure strategies include:

1) *VMIsp*: VM-level Inspection put a suspected VM under the inspection of a in-line mode intrusion prevention system (IPS).
2) *VMIso*: VM-level Isolation disables all network traffic to and from a suspected VM.
3) *PrIsp*: Process-level Inspection redirects the traffic of a suspicious process in the VM to a in-line mode IPS.
4) *PrIso*: Process-level Isolation prevents the suspicious process in a suspected VM from communicating with other VMs while the network services from other processes stay unaffected.

VI. Evaluation

A. *Case Study for Security Performance Analysis*

To demonstrate the security performance of our framework, we created an attack scenario to evaluate our network intrusion detection system with attack graph model and non-intrusive process-based monitoring system with VMI technology.

TABLE I
VULNERABILITIES IN THE TEST NETWORK.

Owner	Host	Vulnerability	CVE ID
User1	Workstation	Internet Explorer	CVE2009-1918
User2	Workstation	none	none
User3	Web Server	Apache HTTP service	CVE2006-3747
User3	Database Server	MySQL database service	CVE2009-2446

Figure 6 shows the test network topology consisting of three users. User1 and User2 acquire a VM for their workstations respectively. They are all connect to the common virtual network trough OVS. User3 creates a private network to host a database server which can not be accessed directly from external network and a web server which can be accessed from internet through firewall and virtual network through OVS. Attacker is assumed to be outside the network and has access to the network through internet. The target for attacker is to get root access on the database server. Table I lists the vulnerabilities present on the VMs inside the test network.

Attack graph for the test network is shown in Fig. 7. The original attack graph contains only one attack path which is the path on the right side from node 1 to node 16. Node 1 in attack graph represents the attacker. Node 16 represents the situation where attacker obtains root privilege on database server in the private network of User3 and allows to execute any code on the server which is the attacker's goal. After node 4 has been exploited, it can lead to node 6 and allows attacker to remotely execute malicious code on user VM. The execution of malicious script allows the attacker network access to the database server through tcp on port 3306, as denoted by node 8. From here, the attacker can exploit another vulnerability on node 16 and can gain root access to the database server. Another possible exploitation sequence the attacker can follow is to go for exploitation of vulnerability denoted by node 12 which can lead the attacker to node 13 allowing permission to execute code on the apache webserver.

The attack path on the left side is dynamically created by the attack graph constructor when the vulnerability on the node

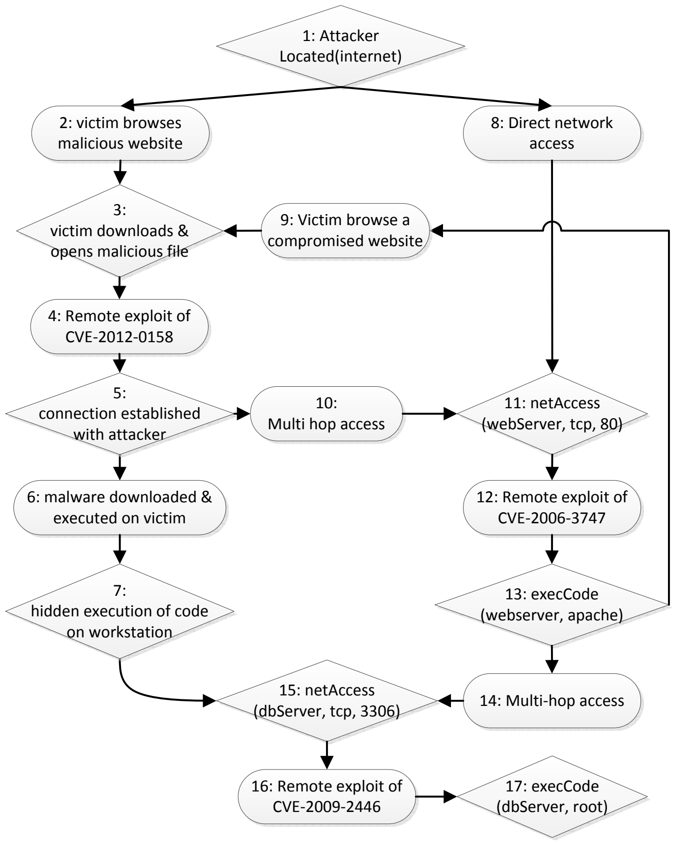

1: Attacker Located(internet)
2: victim browses malicious website
8: Direct network access
3: victim downloads & opens malicious file
9: Victim browse a compromised website
4: Remote exploit of CVE-2012-0158
5: connection established with attacker
10: Multi hop access
11: netAccess (webServer, tcp, 80)
6: malware downloaded & executed on victim
12: Remote exploit of CVE-2006-3747
7: hidden execution of code on workstation
13: execCode (webserver, apache)
15: netAccess (dbServer, tcp, 3306)
14: Multi-hop access
16: Remote exploit of CVE-2009-2446
17: execCode (dbServer, root)

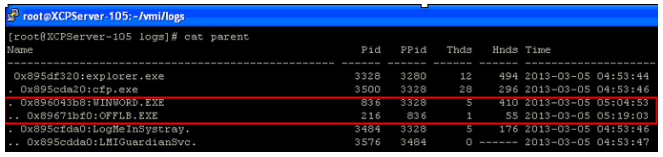
Figure 7. Attack Graph for the test network.

4 (CVE-2012-0158) is detected by NICE-A, which means a user in the virtual network has downloaded a malicious file containing the vulnerability of MSCOMCTL.OCX from attacker's website. Whenever the victim opens the downloaded malicious file,a hidden connection is established to the remote attacker. In order to detect if user on a VM has executed the file, we need to enable the suspicious process monitoring and detection module.

For better understanding we grouped attacker's actions as follows:

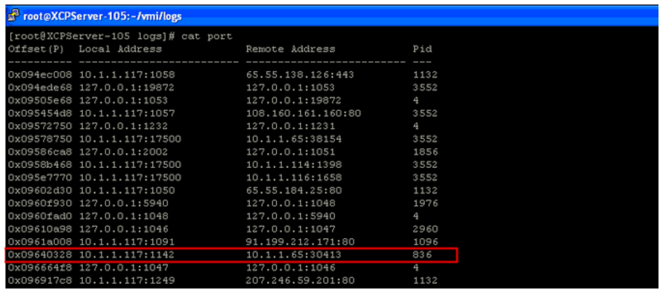
1) Attack Preparation: Attacker places a malicious file on a website and waits for a novice user to download it. The malicious file is capable of connecting back to the attacker to provide a shell control, if it is opened by the victim.

2) Exploitation: When a cloud user downloads the malicious DOC file, NICE-A raises an alert to report CVE-2012-0158 vulnerability. At this stage, the receiver VM is not exploited by the remote attacker, until a user opens the file with MS Office 2007. When the file is opened, the embedded shellcode establishes a connection, under the process name WINWORD.EXE, to the remote attacker. Although the connection can be detected by NICE-A, we cannot say it's a malicious behavior. Now, the suspicious process detector (SPD) in the VM Process Monitor is activated to monitor the process using VMI technology. The SPD detects an unknown process created by WINWORD.EXE and tries to make a



```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat parent
Name                                              Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ -------------------
 0x895df320:explorer.exe                          3328   3280     12    494 2013-03-05 04:53:44
. 0x895cda20:cfp.exe                              3500   3328     28    296 2013-03-05 04:53:46
. 0x896043b8:WINWORD.EXE                           836   3328      5    410 2013-03-05 05:04:53
.. 0x89671bf0:OFFLB.EXE                            216    836      1     55 2013-03-05 05:19:03
. 0x895cfda0:LogMeInSystray.                      3484   3328      5    176 2013-03-05 04:53:46
.. 0x895cdda0:LMIGuardianSvc.                     3576   3484      0 ------ 2013-03-05 04:53:47
```

Figure 8. Unknown process dependence with WINWORD.EXE.



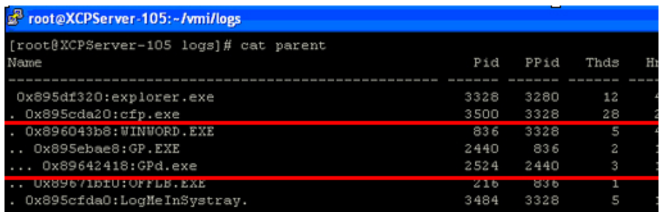
```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat port
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x094ec008 10.1.1.117:1058           65.55.138.126:443         1132
0x094ede68 127.0.0.1:19872           127.0.0.1:1053            3552
0x09505e68 127.0.0.1:1053            127.0.0.1:19872           4
0x095454d8 10.1.1.117:1057           108.160.161.160:80        3552
0x09572750 127.0.0.1:1232            127.0.0.1:1231            4
0x09578750 10.1.1.117:17500          10.1.1.65:38154           3552
0x09586ca8 127.0.0.1:2002            127.0.0.1:1051            1856
0x0958b468 10.1.1.117:17500          10.1.1.114:1398           3552
0x095e7770 10.1.1.117:17500          10.1.1.116:1658           3552
0x09602d30 10.1.1.117:1050           65.55.184.25:80           1132
0x0960f930 127.0.0.1:5940            127.0.0.1:1048            1976
0x0960fad0 127.0.0.1:1048            127.0.0.1:5940            4
0x09610a98 127.0.0.1:1046            127.0.0.1:1047            2960
0x0961a008 10.1.1.117:1091           91.199.212.171:80         1096
0x09640328 10.1.1.117:1142           10.1.1.65:30413           836
0x096664f8 127.0.0.1:1047            127.0.0.1:1046            4
0x096917c8 10.1.1.117:1249           207.246.59.201:80         1132
```

Figure 9. Network connections created by WINWORD.EXE in Fig. 8.

connection to a remote host. Figure 8 shows the process dependency for WINWORD.EXE detected by SPD. Figure 9 further details out the network connections established by WINWORD.EXE.

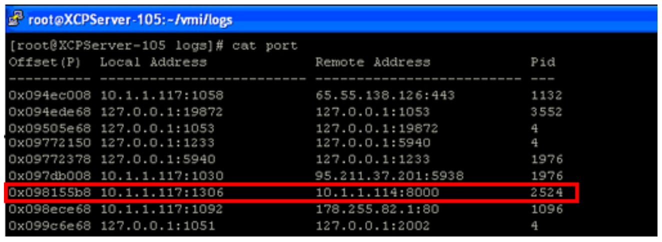
3) Persistent Control: To be able to control the compromised VM, attacker takes help of a malware. For instance, malwares like GP.EXE and FU.EXE are used to manage processes remotely and also allow hiding of processes to avoid detection by the user. In our test case, attacker can be anywhere including internal and external network, (s)he also can change the location (attacker's IP address) to get hold of command and control anytime. The attacker then transfers these two files stealthy through the reverse connection created by the victim. After GP.EXE is executed on victim VM, it will extract and install a malicious daemon GPd on the victim and sets up a connection to the attacker. The GPd modifies victim's auto run registry to attach itself to autostart script, which guarantees persistent control by the attacker. The attacker now can fully control and manage the processes on the victim VM. Since SPD has been enabled, it can detect an unknown process (GP.EXE) which is created by WINWORD.EXE, as shown in Fig. 10. Even though GP.EXE is closed, SPD still be able to detect unknown process GPD.EXE depending on WINWORD.EXE, as shown in Fig. 11. We got the experiment result of Fig. 8-11 with support from volatility project [31].

4) Hidden Control: Hiding processes and their dependency is a common strategy for attacker to obfuscate the detection. To



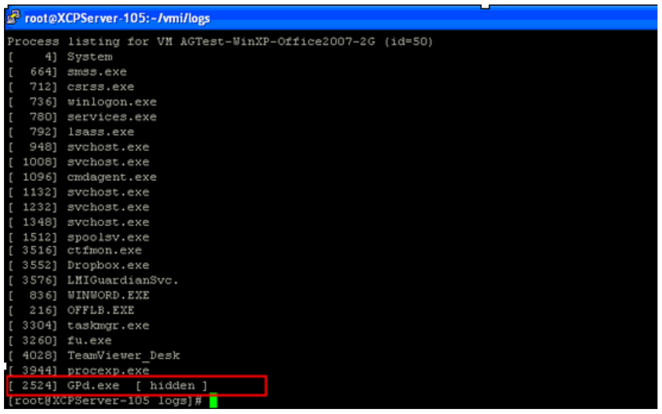
```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat parent
Name                                              Pid   PPid   Thds   H
-------------------------------------------------- ------ ------ ------ ---
 0x895df320:explorer.exe                          3328   3280     12
. 0x895cda20:cfp.exe                              3500   3328     28
. 0x896043b8:WINWORD.EXE                           836   3328      5
.. 0x895ebae8:GP.EXE                              2440    836      2
... 0x89642418:GPd.exe                            2524   2440      3
.. 0x89671bf0:OFFLB.EXE                            216    836      1
. 0x895cfda0:LogMeInSystray.                      3484   3328      5
```

Figure 10. Unknown process dependency.



```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat port
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x094ec008 10.1.1.117:1058           65.55.138.126:443         1132
0x094ede68 127.0.0.1:19872           127.0.0.1:1053            3552
0x09505e68 127.0.0.1:1053            127.0.0.1:19872           4
0x09772150 127.0.0.1:1233            127.0.0.1:5940            4
0x09772378 127.0.0.1:5940            127.0.0.1:1233            1976
0x097db008 10.1.1.117:1030           95.211.37.201:5938        1976
0x098155b8 10.1.1.117:1306           10.1.1.114:8000           2524
0x098ece68 10.1.1.117:1092           178.255.82.1:80           1096
0x099c6e68 127.0.0.1:1051            127.0.0.1:2002            4
```

Figure 11. Unknown process connection.



```
root@XCPServer-105:~/vmi/logs
Process listing for VM AGTest-WinXP-Office2007-2G (id=50)
[    4] System
[  664] smss.exe
[  712] csrss.exe
[  736] winlogon.exe
[  780] services.exe
[  792] lsass.exe
[  948] svchost.exe
[ 1008] svchost.exe
[ 1096] cmdagent.exe
[ 1132] svchost.exe
[ 1232] svchost.exe
[ 1348] svchost.exe
[ 1512] spoolsv.exe
[ 3516] ctfmon.exe
[ 3552] Dropbox.exe
[ 3576] LMIGuardianSvc.
[  836] WINWORD.EXE
[  216] OFFLB.EXE
[ 3304] taskmgr.exe
[ 3260] fu.exe
[ 4028] TeamViewer_Desk
[ 3944] procexp.exe
[ 2524] GPd.exe  [ hidden ]
[root@XCPServer-105 logs]#
```

Figure 12. Detected Hidden Process.

make the control hidden from the attacker, FU.EXE is a good tool for attacker to hide the malicious processes. For instance, the GP daemon process (GPd) can be removed from the process chain list to hide the dependence with WINWORD.EXE, so that the malicious daemon can be executed stealthy from user and administration tools, as shown in Fig. 10. To prevent attackers to hide themselves using this technology, we develop the SPD with ability to detect the hidden processes and their communication, as shown in Fig. 12.

5) Countermeasures: In different stages of alert, cloud controller can make the decision of whether to take countermeasure or which countermeasure should be taken according to the alert level. If SPD does not detect any clue that the vulnerability related to the first alert is exploited, Cloud IPS can take the alert as false positive.

B. System Performance

We performed the evaluation on a cloud server with Intel quad-core Xeon 2.4 GHz CPU and 32G memory. For the non-intrusive suspicious process detection and monitoring system, we created eight VMs running Windows XP SP3. In addition to the default processes in Windows XP SP3 (27 processes), we launched multiple instance of NOTEPAD.EXE to increase the number of process to be detected in each VM for different tests. All tests were performed on different number of VMs (from 1VM to 8VMs) but the same configuration. Six different tests (with different number of process) were performed for each test run. Every test was performed 100 times with the same number of process for all VMs in that test run. The average time elapsed for each test is shown in Fig. 13.

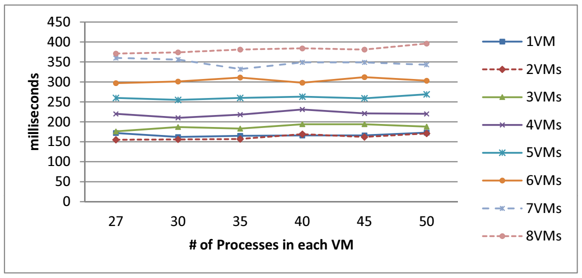


Figure 13. VMI Performance Evaluation.

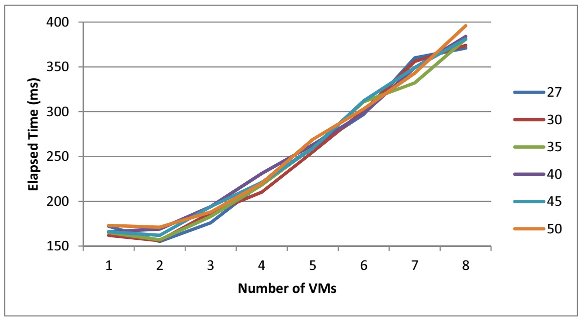


Figure 14. VMI Performance Evaluation.

As we can see from Fig. 13, the average process time for each test run approaching to a horizontal line which means the detection time is independent from the number of process. However, the average process time is proportional to the number of VM to be monitor simultaneously, as shown in Fig. 14.

To measure the fine-grained performance impact of our suspicious process detector, we used UnixBench benchmark [32]. The results are shown in Fig. 15. The worst-case overhead of our system is 9.25%, while the overheads in most other cases are below 10%. The overall system performance overhead is 3.6% which is a small amount.

VII. Conclusion and Future Work

The system we proposed in this paper integrates a network based intrusion detection system to monitor and detect the traffic in the virtual network and a non-intrusive host based suspicious process monitor and detection system using "out-

System Benchmarks Index Values	Unit	w/o SPD	w SPD	overhead
Dhrystone 2 using register variables	lps	956.5	910.8	4.78%
Double-Precision Whetstone	MWIPS	395.7	394.3	0.35%
Execl Throughput	lps	298.4	281.5	5.66%
File Copy 1024 bufsize 2000 maxblocks	KBps	1299.7	1236.6	4.85%
File Copy 256 bufsize 500 maxblocks	KBps	774.9	774.8	0.01%
File Copy 4096 bufsize 8000 maxblocks	KBps	2025	1837.6	9.25%
Pipe Throughput	lps	615.4	578.3	6.03%
Pipe-based Context Switching	lps	294.4	274.3	6.83%
Process Creation	lps	204.3	193.1	5.48%
Shell Scripts (1 concurrent)	lpm	740.5	715.5	3.38%
Shell Scripts (8 concurrent)	lpm	2217.1	2176.3	1.84%
System Call Overhead	lps	422.4	403	4.59%
System Benchmarks Index Score		647	623.7	3.60%

Figure 15. SPD Overhead measurement using Unixbench benchmark.

of-box" VMI technology. Moreover, the host-based intrusion detection is based on VM introspection techniques that do not need the implement special codes in users' VMs.

When hardening network security, hosts cannot be kept apart. Our IDS framework takes care of network security and VM Process Monitor accounts for the security of the host machines. The major benefit for using our framework is to gain the ability to select optimal countermeasures and making the virtual system attack resilient by deploying these countermeasures swiftly. Furthermore, agility of our defense mechanism can be greatly improved with the use of SDN approach and provide an adaptive defense-in-depth approach.

From the case study for the security analysis and system performance we conducted in the evaluation section, it shows that our system is able to capture the malicious traffic and detect the suspicious process related to the alert.

In order to increase the detection accuracy of intrusion and presence of malware in the cloud, we need develop a more sophisticated malware analysis and detection system for our framework in the future to cover different types of VMs, operation systems, vulnerabilities, and attacks. Additionally, our proposed solution suffers from scalability issues since generation of attack graph is complex. We will investigate the decentralized network attack analysis model and/or collaborative approach to address the scalability issues of our framework.

Acknowledgment

The authors would like to thank ONR Young Investigator Program (YIP) award to support the research as part of the secure mobile cloud computing project. The authors would also like to thank the anonymous reviewers for their constructive comments and suggestions to improve the research quality of this work.

References

[1] W. Jansen, "Cloud hooks: Security and privacy issues in cloud computing," in *2011 44th Hawaii International Conference on System Sciences (HICSS)*, Jan. 2011, pp. 1 –10.

[2] H. Qian and D. Medhi, "Server operational cost optimization for cloud computing service providers over a time horizon," in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, ser. HotICE'11, 2011.

[3] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proc. of the 10th Annual Network and Distributed Systems Security Symposium*, 2003.

[4] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction," *ACM Transaction on Information and System Securirty*, vol. 13, no. 2, pp. 12:1–12:28, Mar. 2010.

[5] X. Jiang and X. Wang, "Out-of-the-Box monitoring of VM-Based high-interaction honeypots," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, C. Kruegel, R. Lippmann, and A. Clark, Eds. Springer Berlin Heidelberg, Jan. 2007, no. 4637, pp. 198–218.

[6] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "NICE: network intrusion detection and countermeasure selection in virtual network systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 4, pp. 198–211, Jul. 2013.

[7] "Citrix XenServer." [Online]. Available: http://www.citrix.com/xenserver

[8] "VMware NSX Network Virualization." [Online]. Available: http://www.vmware.com/products/nsx/

[9] P. Salvador, A. Nogueira, U. Franca, and R. Valadas, "Framework for zombie detection using neural networks," in *Fourth International Conference on Internet Monitoring and Protection, 2009. ICIMP '09*, 2009, pp. 14–20.

[10] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Antfarm: Tracking processes in a virtual machine environment," in *Proceedings of the USENIX Annual Technical Conference*, 2006, pp. 1–4.

[11] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "VMM-based hidden process detection and identification using lycosid," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '08. New York, NY, USA: ACM, 2008, pp. 91–100.

[12] C. Benninger, S. Neville, Y. Yazir, C. Matthews, and Y. Coady, "Maitland: Lighter-weight VM introspection to support cyber-security in the cloud," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 471 –478.

[13] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process out-grafting: an efficient "out-of-VM" approach for fine-grained process execution monitoring," in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 363–374.

[14] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *2002 IEEE Symposium on Security and Privacy, 2002. Proceedings*. IEEE, 2002, pp. 273– 284.

[15] O. M. Sheyner, "Scenario graphs and attack graphs," Ph.D. dissertation, Carnegie Mellon University, 2004.

[16] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*, ser. CCS '02. New York, NY, USA: ACM, 2002, pp. 217–224.

[17] S. Jajodia, "Topological analysis of network attack vulnerability," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, ser. ASIACCS '07. New York, NY, USA: ACM, 2007, pp. 2–2.

[18] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: a logic-based network security analyzer," in *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*. Berkeley, CA, USA: USENIX Association, 2005, pp. 8–8.

[19] H. Qian, D. Medhi, and T. Trivedi, "A hierarchical model to evaluate quality of experience of online services hosted by cloud computing," in *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2011, pp. 105–112.

[20] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 336–345.

[21] "Kernel based Virtual Machine (KVM)." [Online]. Available: http://www.linux-kvm.org/

[22] "OpenFlow project," http://openflow.org/. [Online]. Available: http://openflow.org/

[23] "Open vSwitch project," http://openvswitch.org/. [Online]. Available: http://openvswitch.org/

[24] "Open source vulnerability database (OVSDB)," http://osvdb.org/.

[25] Mitre Corporation, "Common vulnerabilities and exposures, CVE," http://cve.mitre.org/.

[26] NIST, "National vulnerability database, NVD," http://nvd.nist.gov.

[27] L. Wang, A. Liu, and S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts," *Computer Communications*, vol. 29, no. 15, pp. 2917–2933, Sep. 2006.

[28] S. Roschke, F. Cheng, and C. Meinel, "A new alert correlation algorithm based on attack graph," in *Computational Intelligence in Security for Information Systems*, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 6694, pp. 58–67.

[29] S. Roschke, F. Cheng, and C. Meinel, "A flexible and efficient alert correlation platform for distributed IDS," in *2010 4th International Conference on Network and System Security (NSS)*. IEEE, Sep. 2010, pp. 24–31.

[30] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[31] "Volatility." [Online]. Available: https://code.google.com/volatility/

[32] "Unixbench – a Unix benchmark suite." [Online]. Available: http://www.tux.org/pub/tux/niemi/unixbench/