

Non-Intrusive Decentralized Attack Analyzer & Network Controller to Monitoring and Prevent Vulnerability in VM Network System

K. Senthil Raja¹, G. Sudhakar², Dr. S. Nithyanandam³

¹M.E/CSE, Ranganathan Engineering College, Coimbatore,

²AP/CSE, Ranganathan Engineering College, Coimbatore,

³Principal, PGP Engineering College & Technology, Namakkal.

Abstract

Cloud security is one of most important issues that have attracted a lot of research and development effort in past few years. Vulnerable virtual machines are an easy target and existence of such weak nodes in a network its entire security structure. Resource sharing nature of cloud favors the attacker, in that, compromised machines can be used to launch further devastating attacks. 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 decentralized 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.

1. 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.

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:

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

2. 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 caused 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.

3. System Overview And Models

3.1. 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 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 VMA simple network system. 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.

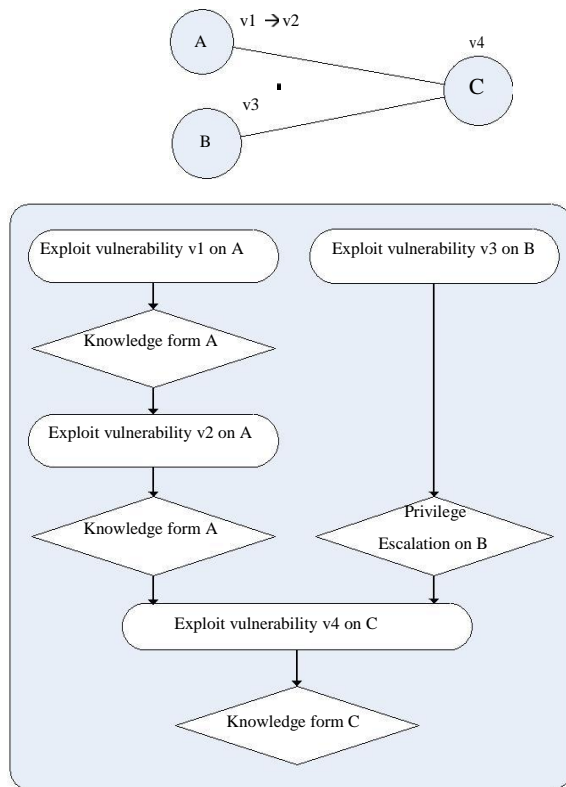


Figure 1. A simple attack graph example.

3.2. 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.

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 be exploited by an attacker if (s) he has exploited v1 and obtained the required privilege to exploit v2. VM B and C have 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 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.

4. System Design

4.1. 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. 2. The control center consists of a decentralized network controller, a VM profiler, and a decentralized attack analyzer.

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

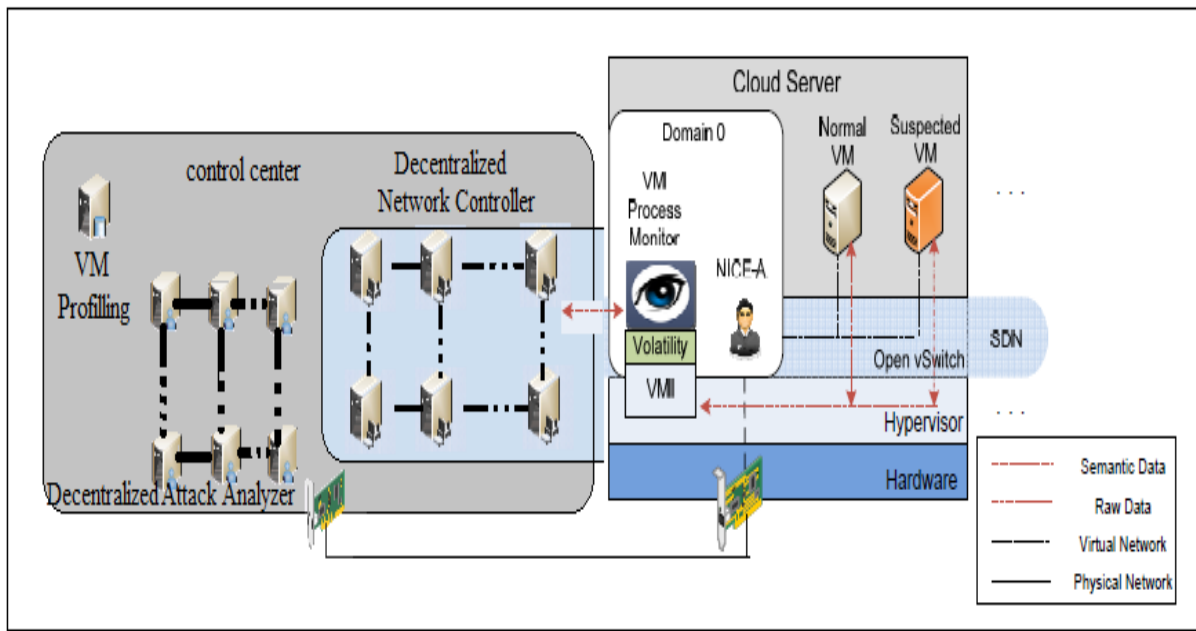


Figure 2: System Architecture

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, decentralized attack analyzer evaluates the severity of the alert based on the attack graph. It then initiates countermeasures through the decentralized network controller after deciding what countermeasure strategies to take.

As described in [6], countermeasures initiated by the decentralized attack analyzer are based on the evaluation results from the cost benefit analysis of the effectiveness of countermeasures. The decentralized 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.

4.2. Decentralized Attack Analyzer

Attack Analyzer is a decentralized 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 Decentralized Network Controller for countering or mitigating the attack. These functionalities are realized by four subcomponents: Attack Graph analysis model, countermeasure selection, and VM profiler.

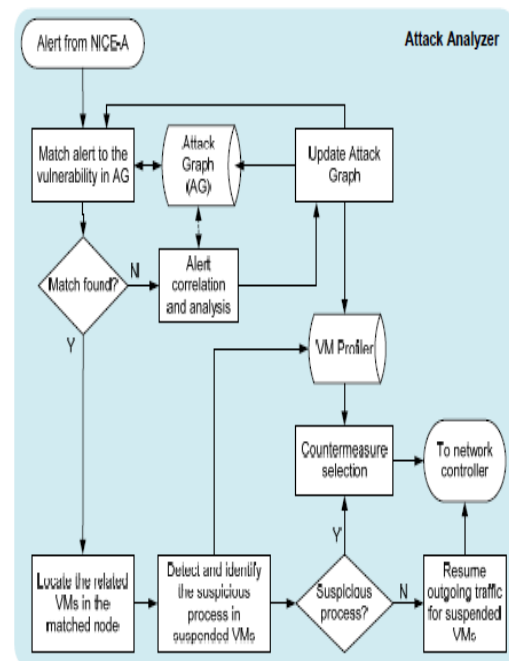


Figure 3: workflow of attack analyzer

Figure 3 shows the workflow in the decentralized 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, decentralized 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 decentralized network controller based on the severity of evaluation results. If the inspected process is found to be harmful, appropriate countermeasure is applied by the decentralized network controller, otherwise the outgoing traffic is resumed from the suspended VM.

4.3. 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 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 frame-work, alert correlations and analysis are also handled by Decentralized 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 Decentralized Network Controller for virtual network reconfiguration or further inspection.

4.4. Decentralized Network Controller

Decentralized 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 decentralized Attack Analyzer. Decentralized 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 Decentralized Network Controller, allowing the controller to set security/filtering rules on both OVS and OFS. Decentralized Network Controller is also responsible for collecting network information of current OpenFlow network, and provides inputs to decentralized Attack Analyzer to construct attack graphs.

4.5. VM Profiler

VM Profiler keeps tracking the security-related status of each VM. These profiles are necessary for the decentralized Attack Analyzer to identify 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.

4.6. 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 pre-served, 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 4 shows VM Process Monitor. We develop five sub-modules 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

To VM profiler and attack graph constructor

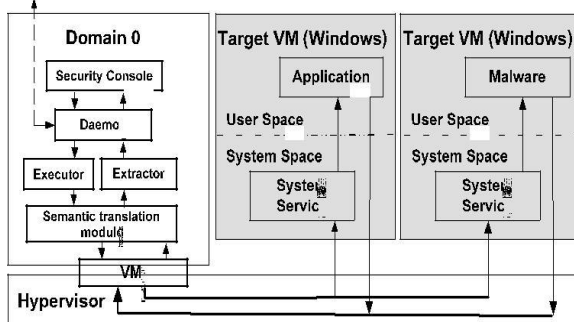


Figure 4. VM Process Monitor.

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.

5. Countermeasure Strategies

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

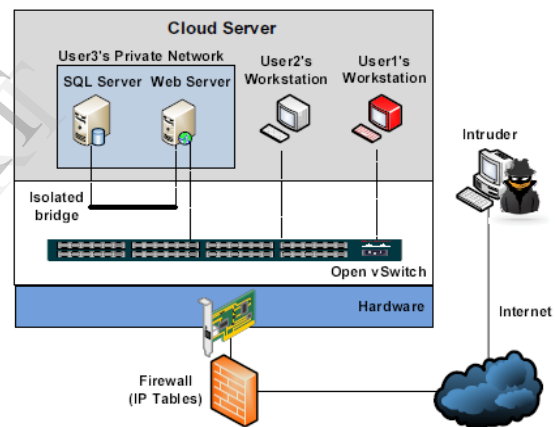


Figure 5. Network topology for case study

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, is 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 Decentralized 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.

6. Evaluation

6.1. 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.

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 through OVS

Table I lists the vulnerabilities present on the VMs inside the test network.

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

User3 creates a private network to host a database server which cannot 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. Attack graph for the test network is shown in Fig. 6. 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 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.

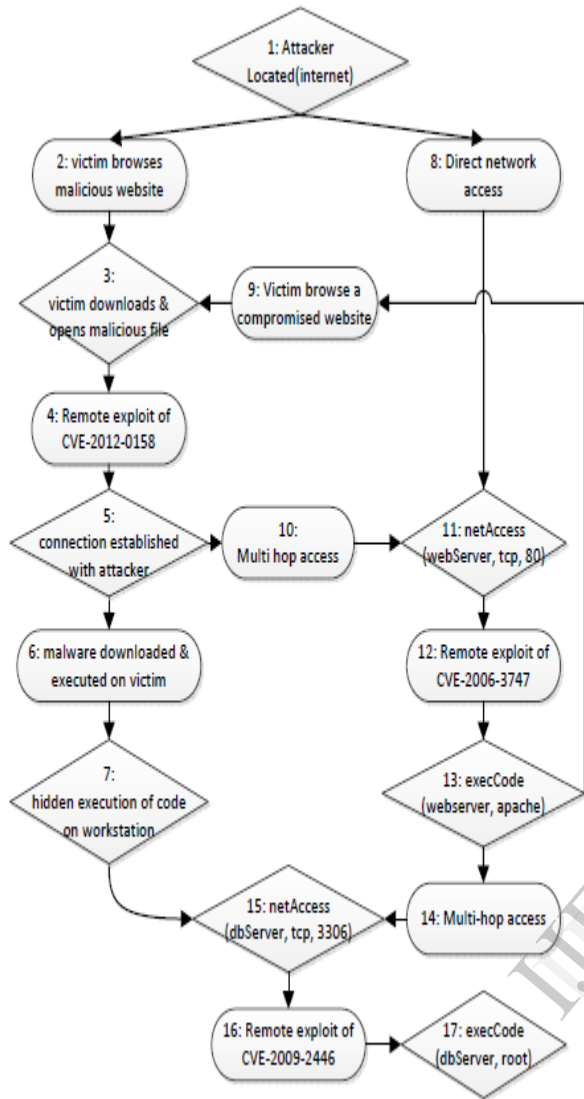


Figure 7. Attack Graph for the test network.

```

root@XCPServer-105:~/vml/logs
[root@XCPServer-105 logs]# net parent
Name                               Pid  Ppid  tbrs  nnds  Time
-----
0x355d2120: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
0x89671b40:OFFLB.EXE                 216   836   1     55  2013-03-05 05:19:03
0x895cfa00:lgis IrSystem?            3904  3328   5    176  2013-03-05 04:53:46
0x895cda00:LogMeInSystem?            3576  3484   0     ----  2013-03-05 04:53:47
  
```

Figure 7. Unknown process dependence with WINWORD.EXE.

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. Exploitation: When a cloud user downloads the malicious DOC files, 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 connection to a remote host. Figure 7 shows the process dependency for WINWORD.EXE detected by SPD. Figure 8 further details out the network connections established by WINWORD.EXE

```

root@XCPServer-105:~/vml/logs
[root@XCPServer-105 logs]# cat port
Offset (P)  Local Address      Remote Address      Pid
-----
0x094ee008  10.1.1.117:1058    65.55.138.126:443   1132
0x094ee0e8  127.0.0.1:19272   127.0.0.1:1083      3582
0x09505e68  127.0.0.1:1083    127.0.0.1:19872     4
0x095454d8  10.1.1.117:1057   108.160.161.160:80   3552
0x09572750  127.0.0.1:1232    127.0.0.1:1531      4
0x09578750  10.1.1.117:17500  10.1.1.65:38154     3552
0x09586cae  127.0.0.1:2002    127.0.0.1:1091      1896
0x0958e4e8  10.1.1.117:17500  10.1.1.114:2938     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
0x0960e930  127.0.0.1:5940    127.0.0.1:1048      1976
0x0960fda0  127.0.0.1:1048    127.0.0.1:5940      4
0x09610a68  127.0.0.1:1048    127.0.0.1:1047      2980
0x0961a008  10.1.1.117:1091   91.199.212.171:80   1096
0x09640328  10.1.1.117:1142   10.1.1.65:30413     836
0x09666448  127.0.0.1:1047    127.0.0.1:1046      4
0x096917e8  10.1.1.117:1249   207.246.89.201:80   1132
  
```

Figure 8: Network connections created by WINWORD.EXE in Fig. 7.

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.

```

root@XCPServer-105:~/vml/logs
[root@XCPServer-105 logs]# cat parent
Name                               Pid  PPid  Thds  H
-----
0x895d4f320: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
0x89671b40:OFFLB.EXE                 216   836    1
0x895cfa00:LogMeInSystem?            3484  3328    5
  
```

Figure 9. Unknown process dependency

The attacker then transfers these two files stealthily 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 auto start script, which guarantees persistent control by the attacker.


```

root@XCPServer-105:~/mlls
[root@XCPServer-105 logs]# cat port
Offset(P) Local Address Remote Address Pid
-----
0x001e008 10.1.1.117:1058 65.55.198.126:443 1132
0x001e0e8 127.0.0.1:10872 127.0.0.1:1053 3552
0x00505e8 127.0.0.1:1053 127.0.0.1:19872 4
0x00772150 127.0.0.1:1233 127.0.0.1:5940 4
0x00772378 127.0.0.1:5940 127.0.0.1:1233 1976
0x007db008 10.1.1.117:1030 95.211.37.201:5938 1976
0x008155b8 10.1.1.117:1306 10.1.1.114:8000 2524
0x008e2e68 10.1.1.117:1092 178.255.82.1190 1096
0x009e6e68 127.0.0.1:1051 127.0.0.1:2002 4
    
```

Figure 10. Unknown process connection.

```

root@XCPServer-105:~/mlls
Process listing for VM AGTest-WinXP-Office2007-2G (id=50)
[ 4] System
[ 664] smss.exe
[ 712] csrss.exe
[ 730] winlogon.exe
[ 780] services.exe
[ 792] lsass.exe
[ 848] svchost.exe
[ 1000] svchost.exe
[ 1096] cmdagent.exe
[ 1132] svchost.exe
[ 1232] svchost.exe
[ 1348] svchost.exe
[ 1512] spoolsv.exe
[ 2516] ctmon.exe
[ 3552] Dropbox.exe
[ 3576] LMIGuardianSvc.
[ 836] WINWORD.EXE
[ 216] OFFICE.EXE
[ 3904] taskmgr.exe
[ 3260] ru.exe
[ 4028] TeamViewer_host
[ 3244] process.exe
[ 2524] GPD.exe [ hidden ]
root@XCPServer-105 logs]#
    
```

Figure 11 Detected Hidden Process.

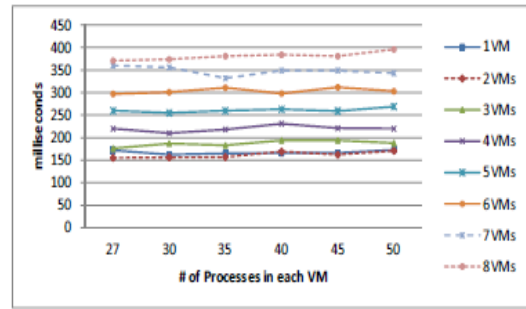


Figure 12. VMI Performance Evaluation.

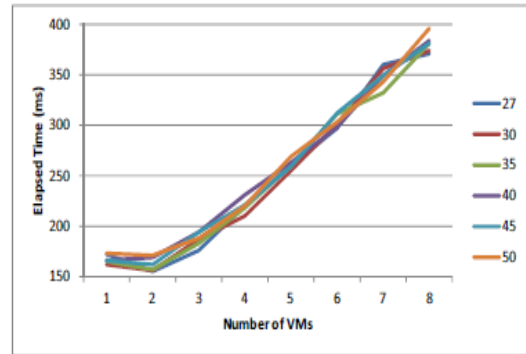


Figure 13. VMI Performance Evaluation.

7. Conclusion

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. 9. Even though GP.EXE is closed, SPD still be able to detect unknown process GPD.EXE depending on WINWORD.EXE, as shown in Fig. 10. We got the experiment result of Fig. 7-10 with support from volatility project [31].

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-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.

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

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. 13.

To measure the fine-grained performance impact of our suspicious process detector, we used UnixBench benchmark [32]. The results are shown in Fig. 14. 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.

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%
Excel 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 14. SPD Overhead measurement using unixbench benchmark.

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.

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 re-construction," *ACM Transaction on Information and System Security*, 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 Virtualization." [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: Lightweight VM introspection to support cybersecurity 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 (OSVDB)," <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/> "Unixbench – a Unix benchmark suite." [Online]. Available: <http://www.tux.org/pub/tux/niemi/unixbench/>