

Securing KVM-based Cloud Systems via Virtualization Introspection

Sheng-Wei Lee

Dept. of Management Information System
National Chengchi University
Taipei, Taiwan
100356010@nccu.edu.tw

Fang Yu

Dept. of Management Information System
National Chengchi University
Taipei, Taiwan
yuf@nccu.edu.tw

ABSTRACT

Linux Kernel Virtual Machine (KVM) is one of the most commonly deployed hypervisor drivers in the IaaS layer of cloud computing ecosystems. The hypervisor provides a full-virtualization environment that intends to virtualize as much hardware and systems as possible, including CPUs, network interfaces and chipsets. With KVM, heterogeneous operating systems can be installed in Virtual Machines (VMs) in an homogeneous environment. However, it has been shown that various breaches due to software defects may cause damages on such a cloud ecosystem. We propose a new Virtualization Introspection System (VIS) to protect the host as well as VMs running on a KVM-based cloud structure from malicious attacks. VIS detects and intercepts attacks from VMs by collecting their static and dynamic status. We then replay the attacks on VMs and leverage artificial intelligence techniques to derive effective decision rules with unsupervised learning nature. The preliminary result shows the promise of the presented approach against several modern attacks on CVE-based vulnerabilities.

1. Introduction

Cloud computing has become one of the most dominant computation platforms nowadays. While system vendors and public users are benefit from sharing resources in the cloud environment, security breaches that may cause worse damages of the cloud ecosystem than personal computers could be one of the major stunning blocks on this evolution road. One successful exploit of the cloud host may lead to the compromise of all guest VMs. It is hence essential and desirable to have effective and systematic mechanisms to enhance security of cloud computing platforms.

VMs are running in the cloud. One open-source solution that has been widely adopted is using Linux [26] Kernel-based Virtual Machine (KVM) as a hypervisor to drive each VM running on top of the cloud platforms. The role of KVM is to virtualize each device, such as mother boards, CPUs, RAM, hard drives, network interface cards, system

timer, and virtual networks. On the host, every VM is running as a KVM process. This enables the VM manager has rights to deploy debug tracing tools and VM device monitors, e.g., qemu-monitor, to check both static and dynamic status of VMs and take actions online on VMs when malicious or abnormal behaviors are observed. That is to say, VM managers should be able to not only protect hosts from potential attacks, but also provide a new security service to guest VM users, detecting whether the VMs have been compromised.

One can briefly summarize types of attacks on the cloud computing platforms as below: (1) attacking the hypervisor: attackers can gain host privileges via software defects and vulnerabilities of hosts from guests, (2) attacking guest VMs from another VM: attackers can crack another guest in the same cloud environment, and (3) attacking guest VMs from the outside: attacker can crack guest from outside of the cloud environment. In this work, we present a new virtualization introspection system (VIS) that monitors both static and dynamic status of guest VMs to detect and prevent the cloud ecosystem from potential malicious attacks. VIS is an aggressive, real-time monitoring mechanism with the aim of achieving efficiency, precision and transparency.

1.1 New attack threat in the cloud computing platforms

In 2011, a new attack on KVM-based cloud systems that leads to the controlling right taken from the hypervisor by a VM was first introduced by N. Elhage [21]. The attack is known as *cloudburst attack*. Elhage exploited the memory-leak bug (CVE-2011-1751) in the kernel KVM codes with Virtunoid to break into KVM hosts and take the full control of the KVM hypervisor. This bug makes KVM unable to check the virtualization of PCI devices, and makes it unclear if the device is unplugged. KVM fails to omit the users' request to unplug the device due to the software defect that causes the routine fails to support unplugged status. This means when a VM unplugs such a device, KVMs may not

clean up the memory and disconnect themselves, leaving some dangling pointers behind that hackers may take advantage to exploit the system. This cloudburst attack can be launched on cloud middleware such as OpenStack, Eucalyptus, RHEV, OpenECP [23] as well. Furthermore, traditional mechanisms that provide non-real-time protection, like SWADR, may not be useful in the face of virtunoid.

In this work, we replay this attack via running a VM in a closed environment, and monitor and traverse the changes of status of the VM's devices, and based on the collected data, we derive effective rules to detect abnormal behaviors of VMs and protect the cloud environment from being attacked by virtunoid.

1.2 Traditional attacks

In addition to new attacks, traditional attacks on personal computers can be launched to attack VMs in the cloud as well. One common way is using penetration-testing tools such as Social Engineering Toolkit (SET) to initialize a social engineering attack. For example, a website attack vector using a Java Applet method to cheat the victim, tricking him into clicking on a malicious link on the website or in an email that embeds a backdoor, e.g., building an ssh channel with a remote site. Other viable attack methods could include new JAVA vulnerabilities (e.g. CVE-2013-0422, CVE-2013-0431). This kind of attacks can be employed by a hacker via using an open source penetration testing tool, e.g., the project framework named *metasploit* [19], during the Oracle's patch delay.

In addition to a social engineering attack, another common attack is (distributed) deny of service (DoS/DDoS) attack. Hackers can compromise VMs in the clouds as jumpers to perform such attacks, making it hard to be tracked. To compromise VMs, one active attack is launched due to the Windows vulnerability, such as ms08_067_netapi or ms12-020. After compromising a victim's VM, the hacker can use a back door program, e.g., *meterpreter* [20] to set the payload - the command that the hacker would like the victim to execute. The payload can be continuously sending a large amount of requests to a server within a short period (DoS). On the other hand, it can also be spy-kind routines, e.g., taking a screenshot of the target or executing the "ps" command to list running processes in the target for further attacks. The *migrate* command can be used to transfer the payload to another running process to prevent the payload from being deactivated, and the *shell* command can be used to provide a command prompt in the meterpreter that you can add an account in the victim. Furthermore, you can use the *use-priv* command to upgrade account privileges to administrator, or the *run-persistence* command to inject an agent of the meterpreter in the victim

VM to ensure that after the target system reboots the backdoor remains open.

To check whether a VM is running a malicious program, we proposed in this work to replay the attack in the cloud environment, and then collect data of VMs during several events. Examples of such events include all VMs idling, a hacker launching an attack, a hacker who already has a backdoor, a hacker sending a keystroke, a hacker taking a screenshot, a hacker launching a pivoting attack, a victim been compromised, etc. VIS can then collect the data of each VM and simultaneously labeled the behavior with the event and VM's ID. After data been collected, VIS will run it through an analyzer for training, learning, and characterizing abnormal behaviors to distinguish between run-time status of VMs (running malicious code, idling etc.). Simultaneously, VIS will use a training model in analyzer to derive decision rules by using GHSOM. Based on these decision rules, VIS can detect and prevent attacks by termination and isolation functions.

VMs that directly attack a hypervisor protected by VIS's termination function will be destroyed or shutdown to prevent damages on the systems, such as the sensitive hypervisor data from theft or destroyed. For a malicious VM to attack another VM in the same cloud platforms, VIS can detect it by decision rules without using agent applications to collect VM's run-time status. Once a VM is detected running malicious programs, VIS will take corresponding responses such as using termination functions to shut down malicious VM or using isolation functions to migrate the malicious VM into a sandboxed host. With isolation, this VM can still work properly, but the VM is physically isolated and is unable to access VMs in the same cloud platform. The purpose is similar to a honeypot to trap further malicious behaviors, and let the administrator of VMs able to trace further hacking behaviors. Similarly, compromised VMs can be migrated to the sandboxed host to be further inspected. The VM manager or the owner may later decide which VM snapshot is clean, and then restore this VM snapshot.

2. RELATED WORK

2.1 Defense in the cloud systems

There are many research papers on the cloud environment working on defending from malicious programs or hackers such as the security risk evaluation of the cloud computing presented by Enisa [8] in detail. In Siebenlist [11], a number of security issues are discussed. There are also many interesting and worthwhile surveys about cloud security presented by Armbruest et al. [17]. However, all of these papers are discussing primarily attacks coming from guest machine users who turned said guest machine in the cloud computing environment into a malicious machine.

A cloud attacker may use a virtualization environment's vulnerabilities break into another VM when they detect the target machine in the cloud environment. In order to guard against above attack method, most approaches use Virtual Machine Monitor (VMM) isolation properties to secure VMs by leveraging different levels of virtual introspection [27]. Virtual introspection [25] allows users to observe a VM's state through this process. In previous papers surveyed, there are some useful approaches catching our attention. For example, SecVisor [3], Lares [4], among others, leverage virtualization to monitor the integrity of guest kernel code from a privileged virtual machine or from the VMM, also known as the hypervisor. Advanced Cloud Protection System (ACPS) [16] is one means of improving the security of cloud nodes. ACPS is an extension of the KvmSec [10] and KvmSma [16] which are also known as extensions of the Linux KVM [24]. ACPS is a protection system that is totally transparent to cloud environments and can also monitor both local and remote cloud components to protect the whole cloud system. But, none of these papers mentioned how to protect cloud platforms from cloudburst attacks and have no advance defense mechanisms, like migration, as presented in this paper.

A running VM employs the virtualized hardware and one of the VMs on the host is a process executed by KVM, and we use another physical host as a “SandBox-Host” [15] and execute the malicious program in a VM. In this way we decided to use “strace” and “qemu-monitor” to collect the dynamic data from VM and compare VM’s malicious behavior to models. Using “strace” this can be viewed as “black-box testing”, as the vector from which the VM can request the needed system call from the host. This system call when can be collected from the running VM for analysis. On the other hand, qemu-monitor can observe the integrity of virtualized hardware and determine if it is normal or being used inappropriately. The execution of a malicious program may come in several phases, such as vulnerability scanning, executing the attack, embed a backdoor, etc, so we separate these period and recording the data for analysis, much like TTAlyzer [31]. On the other hand, we classify the data collected by VIS from strace by period and role. Borrowing from [13], we use static analysis to disassemble the program and expose some suspicious “op code”, to use several different kinds of analysis models to analyze the malicious software and already known and classified normal software. Then unknown software is put into the analysis model. The analysis model then determines if the unknown software is malicious or normal [13]. We also use this method to do a dynamic analysis: VIS uses the statistical result from data collected by “strace” to find out

There are two ways to extract software behavior models that could provide alternative solutions for analyzing virtual machine behavior model. One is to retrieve properties of data values [1][19] such as constraints on legal values in the form of a Boolean expression. The second way would be to examine the properties of interaction patterns such as possible interaction sequences in the form of final state machines [18]. However, neither of these models account for their mutual interplay. Even when one takes both analysis models into consideration, it is hard to see their intricacies clearly.

3. VIS architecture

In this section, we will introduce how the monitor in VIS dump, store and compare the VM runtime state data, and how it judge the abnormal VM runtime state data. And the architecture of VIS as below: Figure 3.1

Figure 3.1 VIS Architecture

Introspection Engine:

The introspection engine consists of the introspection modules, behavior DB, policy DB, behavior analyzer, behavior checker, monitor and controller. The following are the main components of VIS :

- (1) Logging and storage component: monitor, Behavior DB, Policy DB
- (2) Analysis component: Behavior Analyzer, behavior DB, Policy DB
- (3) Introspection components: Introspection Modules, Behavior Analyzer, Behavior Checker
- (4) Control components: Controller, Virtualization API

We use “strace” and “qemu-monitor” to monitor the runtime state of VMs. However, “strace” and “qemu-monitor” monitor the VMs in different ways. In our case, we use “strace” to monitor what kinds of system calls that VMs use in a given runtime, and use “qemu-monitor” to monitor the hardware state of the VMs. By comparing the states of each VM, any abnormal hardware state or any unusual use of system calls will be apparent and we may regard them as malicious attacks from VMs and react accordingly

Introspection Modules

We store many various security rules in introspection modules. Every module can import to the behavior checker as an independent python module, loadable dynamically or executed stand-alone.

Policy Database

This component stores the security policy for introspection module for behavior checkers to compare.

Behavior Database

The behavior DB stores runtime states of all VMs. The program running on the VM will determine the system call and subsequent classification. For example, I usually classify the data by attack method. After attack method, the role of the VM will be used, i.e. hacker, victim, and normal. Then they will be classified by time period, the attack method employed or program used, and system call. Thus, the classification is done in the following order: Attack Method → Role → Period → Program → System call.

Behavior Analyzer

The behavior analyzer will analyze the data stored in the behavior database to build the introspection modules. This will be mentioned in Section Implementation.

Behavior Checker

The behavior checker will import the introspection modules from the policy DB. The introspection rules are

stored in the Introspection module to enable the behavior checker to do the comparison immediately to determine which VM is executing the malicious program.

When a predefined attack is identified, the monitor will pass the domain action message to the controller, such as “domain destroy” or “domain shutdown”, as the response to prevent the attack. In the example domain [action] message could look like “SVM-01 [destroy]”.

Monitor

The responsibility of the monitor is collecting data from the VM, such as the run-time system calls of VM and “qemu-monitor” output from the VM, then they are categorized in the order of “Attack Method → Role → Period → Program → System call” and stored in the behavior database , and from there can be visualized as a bar graph.

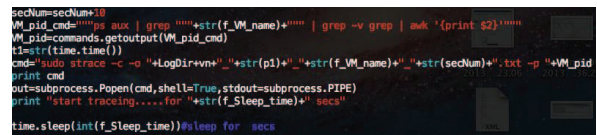
Controller

The controller mainly receives commands from the behavior checker. The behavior checker determine which domain (VM) which action the controller will take, such as allow, destroy, shutdown, migrate etc. The controller will pass the message to the cloud middleware and record the message in cloud middleware’s database (unrelated to the policy DB of VIS) so that the cloud middleware can record the whole state of the cloud. Additionally, cloud middleware can directly use libvirt, virsh to control the compromised VM.

4. Implementation

4.1. Monitoring

To monitor the run-time status of a VM, First, VIS must determine the VM's run-time system call using “strace” to collect VM’s underlying system calls of KVM from Host, as Figure 4.1.1



```
secNum=secNum+1
VM_pid_cmd="ps aux | grep '^'+str(f_VM_name)+'^' | grep -v grep | awk '{print $2}'"
VM_pid=commands.getoutput(VM_pid_cmd)
time.sleep(time.time())
cmd="sudo strace -c -o "+LogDir+vmn+"_"+str(p1)+"_"+str(f_VM_name)+"_"+str(secNum)+".txt -p "+VM_pid
print cmd
out=subprocess.Popen(cmd,shell=True,stdout=subprocess.PIPE)
print "start tracing....for "+str(f_Sleep_time)+" secs"
time.sleep(int(f_Sleep_time))#loop for secs
```

Figure 4.1.1 The strace command

Second, to monitor the VM’s run-time virtualized device, VIS uses “qemu-monitor” to check simulated device of VMs, as Figure 4.1.2

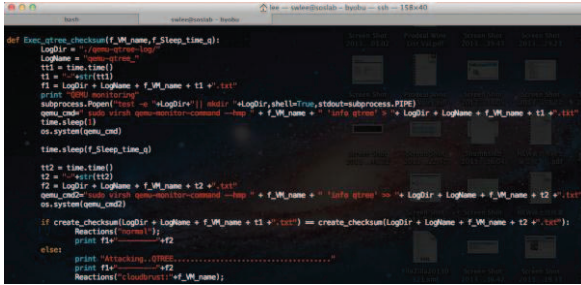


Figure 4.1.2 The qemu-monitor command

As mentioned above, VIS uses strace to monitor the KVM process of the VM, simultaneously, using qemu-monitor to monitor virtualized QEMU devices of the VMs, as Figure 4.1.3

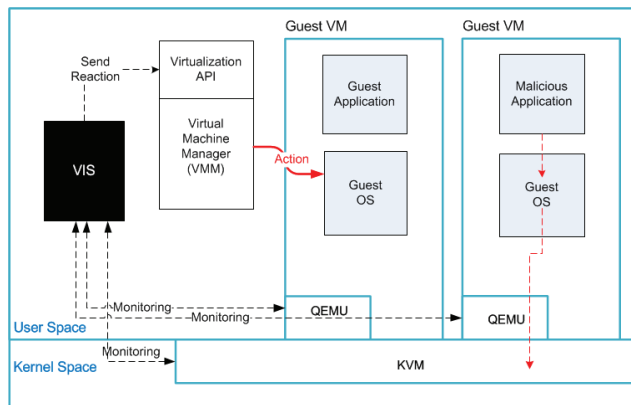


Figure 4.1.3 Monitoring VMs status

4.2 Termination-Shutdown of VMs

Every abnormal behavior is regarded as malicious, thus enabling VIS to intercept a cloudburst attack. For example, if the VM uses a system call abnormally, VIS may use virtualization API libvirt or a virtualization management user interface such as virsh to intercept the attack and shutdown the malicious VM.

If a cloudburst attack is detected, the monitor will pass "domain [shutdown]" or "domain [destroy]" as a message to shutdown the malicious VM. But, the VM image will not be deleted immediately. Instead, it will be offline-migrated to sandboxed host.

4.3 Isolation-Migration of VMs and Redirect iptables

If the attacker is using known traditional attacks, such as using the Metasploit Framework to find other VMs that can be attacked, then VIS will online migrate the VM to the sandboxed host, as in Figure 4.1.3 VIS can monitor the running VM in real time.

When a traditional attack is detected: When the monitor detects a traditional attack, the monitor will use online migration to migrate malicious VMs to specific sandboxed hosts as Figure 4.3.1 by resetting the iptables of the

malicious VM, the malicious VM is physically separated from the cloud, but continues running, making it possible to trace the attacker and has no effect on the original cloud. Because the relative setting has been changed, the malicious VM cannot perpetrate any attacks on the cloud infrastructure or other VMs.

Migration: The first step is to translate the memory of the compromised or malicious VMs to the sandboxed host, then hosts on the both side will check whether the translation is complete. Next, the original host translates the VM's IP to the sandboxed host. In order to sever the connection between compromised/malicious VM to original host, the original host will cut the connection, while configuring the compromised/malicious VM's VLAN and the related "iptables" configuration after the checking the migration is complete (see Figure 4.3.1). In this way, the compromised/malicious VM will not discover it is been migrated. We use this way to track attackers and intercept potential attacks, as Figure 4.1.3

For example, before the VIS migrates the VM to the “SandBox-Host”, it will be connected to the sandbox-host by following command:

```
“virsh -c qemu+ssh://soslab@SandBox-  
Host:999/system”
```

The command “virsh” is a virtualization support management user interface, and the “-c” option is used to specify the URI parameter with which to connect to the hypervisor, in our case we use the “qemu+ssh” to connect with “sandbox-host” and the account is “soslab” that hostname is “SandBox-Host” on port 999.

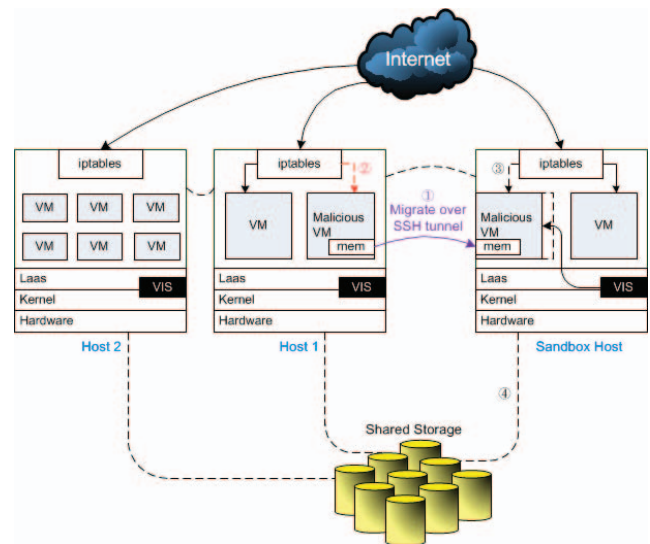
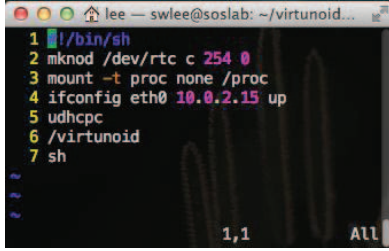


Figure 4.3.1 migrate the VM into sandbox Host

5. EVALUATION

5.1 DETECT CLOUDBURST ATTACK

Virtunoid as an exploit is an example of a cloudburst attack. The exploit will create `/dev/rtc` files and mount `/proc`, then start up the `eth0` interface to execute malicious shell code. In this paper, we rebuild the malicious code in the guest VM. At first, we compiled `virtunoid.c` to build the files `virtunoid` and `initrd.gz`, as Figure 5.1.1



```

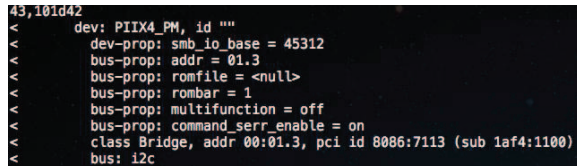
1 |! /bin/sh
2 |mkmod -dev/rtc c 254 0
3 |mount -t proc none /proc
4 |ifconfig eth0 10.0.2.15 up
5 |udhcpc
6 |/virtunoid
7 |sh

```

Figure 5.1.1 Rebuild the attack VM

We set up the malicious `initrd.gz` and updated the `grub2` menu, then reboot the VM. The exploit will trigger the attack at boot. The VM Linux kernel will call the `init` program then run the script from `initrd.gz` and exploit the host. The Cloudburst's VM executes the exploit of Cloudburst called `virtunoid`. The attack will be launched after VM boots up. The purpose of this experiment is to observe Cloudburst's VM with another VM's system call, with output of `qemu-monitor's` result for each being different. Cloudburst's VM will execute special system calls, and before running these calls, it will "unplug" the `PII4` chip (See figure 5.1.2, below). We rebuild the Cloudburst attack in the VM.

We monitor the run-time states and hardware integrity of the VMs on the host side, and collect important `qemu-monitor` data of the guest machine; such as CPU registers, virtual-to-physical memory mappings, active virtual memory mappings, `qdev` device model list (name "`PIIX4`", as Figure 5.1.2, bus `PCI`, desc "`ISA bridge`", normally inaccessible to users), device tree, and system calls.



```

43,101d42
< dev: PIIX4_PM, id ""
< dev-prop: smb_io_base = 45312
< bus-prop: addr = 01.3
< bus-prop: romfile = <null>
< bus-prop: rombar = 1
< bus-prop: multifunction = off
< bus-prop: command_serr_enable = on
< class Bridge, addr 00:01.3, pci id 8086:7113 (sub 1af4:1100)
< bus: 12c

```

Figure 5.1.2 `PIIX4` chip being "unplugged"

5.2 Detect Traditional Attacks

We demonstrate six different kinds of attack, in each attack we divided several different periods, the periods are different is because we used different attack exploit and payload. The following table I shown each different attacks/vulnerabilities has how much periods, and we will describe two different kind of attacks/vulnerabilities.

TABLE I
SIX DIFFERENT KIND OF ATTACKS/VULNERABILITIES AND PERIODS

Attacks/Vulnerabilities	Periods
CVE-2013-0422	9
CVE-2013-0431	9
Dos-hping3	3
MS12-020	3
SET-Web-Java-Applet	9
SE Firefox xpi	9

For example, in CVE-2012-0422 it is a java vulnerability, it can be attack by using a exploit named "`Java_jre17_jmxbean`" in metasploit and the payload can be set up as meterpreter, by using the meterpreter as payload from first period in our experiment is each VM do nothing, and there have nine different periods as following describe, period 1 is hacker VM setup attack, period 2 is hacker VM using exploit to attack victim VM, after hacker VM compromised victim VM, hack VM was able to use meterpreter to do malicious in victim VM, e.g., taking a screenshot of the target's screenshot or executing the "`ps`" command to list running processes in the target for "`migrate`" command. The *migrate* command can be used to transfer the payload to another running process to prevent the payload from being deactivated, and the *shell* command can be used to provide a command prompt in the meterpreter that you can add an account in the victim.

In this experiment we recorded the system call distribution of three VMs (hacker VM, victim VM, normal VM) by every ten second with six different kind of attacks (in the form of 973 files) analyzed via Growing-Hierarchical Self-Organizing Maps (GHSOM). The result of three classified result is given in the Table II.

TABLE II
THREE CLASSIFIED RULES IN THE GHSOM ANALYZE RESULT

52	'SET-Web-Java-Applet_kill_BT5R3_Base_40.txt' 'CVE-2013-0431_shell_BT5R3_Base_20.txt' 'CVE-2013-0422_keylogrecorder_BT5R3_Base_20.txt' 'MS12020_attacking_BT5R3_Base_10.txt' 'CVE-2013-0431_migrate_BT5R3_Base_20.txt'
51	'CVE-2013-0422_attacking_BT5R3_Base_10.txt' 'CVE-2013-0422_attacking_BT5R3_Base_20.txt' 'CVE-2013-0422_attacking_BT5R3_Base_70.txt' 'CVE-2013-0422_attacking_BT5R3_Base_80.txt' 'CVE-2013-0422_attacking_BT5R3_Base_30.txt' 'CVE-2013-0431_attacking_BT5R3_Base_30.txt' 'CVE-2013-0431_attacking_BT5R3_Base_10.txt' 'CVE-2013-0431_attacking_BT5R3_Base_20.txt' 'CVE-2013-0422_attacking_BT5R3_Base_40.txt' 'CVE-2013-0422_attacking_BT5R3_Base_60.txt' 'CVE-2013-0422_attacking_BT5R3_Base_50.txt'
50	'CVE-2013-0422_screenshot_Victim_10.txt' 'CVE-2013-0422_ps_Victim_10.txt'

```
'CVE-2013-0431_ps_Victim_10.txt'
'CVE-2013-0422_sysinfo_Victim_10.txt'
'SET-Web-Java-Applet_shell_Victim_10.txt'
```

In the Table II (No. 51) consists only of data collected from the hacker's VM during DoS attacks. Data collected from the hacker's VM during the "attack" period has been sorted into items of this class so that VIS can use the mean of these classes to calculate the run-time status of each VM for reference to prevent similar DoS attacks.

Unfortunately, this figure is not very clear, so we'll examine data from the "hacker's VM" during the period when it perpetrates malicious behavior in summary. In the Figure 5.2.2, each system call is displayed as a box.

Unfortunately, figure 5.2.1 is not easily understood at first glance. In the figure below, we've taken each class and colored it based on frequency of usage, arranged in the same pattern as figure 5.2.2. White indicates the most frequently used classes, gray less used, with black being the least used.

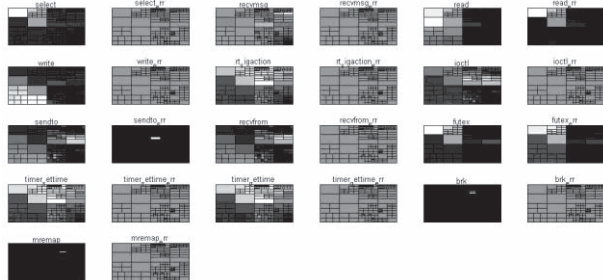


Figure 5.2.2 Detail summary in system call

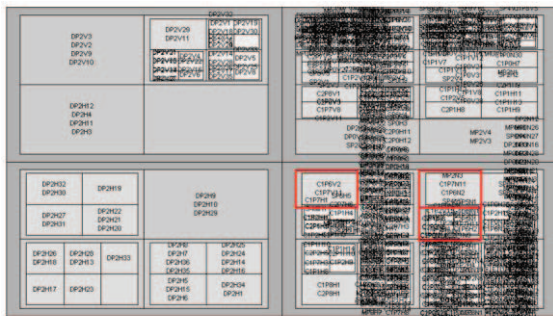


Figure 5.2.3 Class No. 50, 51, 52

After analysis and classification, we can clearly see in the summary which class is dominated by the "hacker's VM" running malicious programs. This has been highlighted in red in the Figure 5.2.3. On the other hand, as in Figure 5.2.4 below, you can see how each file is classified. This figure (Figure 5.2.4) is Class No. 50.

```
CVE-2013-0422_ps_Victim_10.txt
CVE-2013-0422_screenshot_Victim_10.txt
CVE-2013-0422_sysinfo_Victim_10.txt
CVE-2013-0431_ps_Victim_10.txt
SET-Web-Java-Applet_shell_Victim_10.txt
```

Figure 5.2.4 Classified data of Class 50

Figure 5.2.4 is data from the victim once it has been compromised by the hacker and already under the influence of malicious programs. This has been classified as Class No. 50 of the GHSOM analysis. In this case, we can set the reaction rule notify the customer that the VM been compromised.

Then we use the mean of the GHSOM analysis (as Figure 5.2.5) to calculate each class of data set to determine the decision rules for VIS.

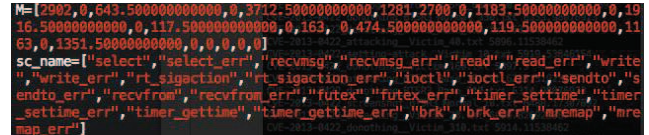


Figure 5.2.5 Mean of Class No. 50

Figure 5.2.5 shows the mean GHSOM calculation result of Class No. 50 of a compromised victim's VM. VIS can compare this data with the VM's running status, determine if the VM has been compromised, and either migrate it or recover it from a snapshot.

$$\frac{\sum |X - M|}{\text{Sum}X}$$

Figure 5.2.6 The formula to calculate run-time data with the mean

Figure 5.2.6 VIS shows the formula used to calculate the distance between a VM's run-time status and the mean of the attack. If the VM's run-time status is farther the expected status, then the VM is not under attack.

```
CVE-2013-0422_sysinfo_Victim_10.txt 181.692307692
CVE-2013-0422_screenshot_Victim_10.txt 157.153846154
CVE-2013-0422_ps_Victim_10.txt 163.307692308
SET-Web-Java-Applet_shell_Victim_10.txt 172.230769231
CVE-2013-0431_ps_Victim_10.txt 168.269230769
50Mean.txt (END)
```

Figure 5.2.7 Mean of Class No.50

Figures 5.2.7 shows how VIS can choose decision rules to determine the tolerance between the mean and VM running state.

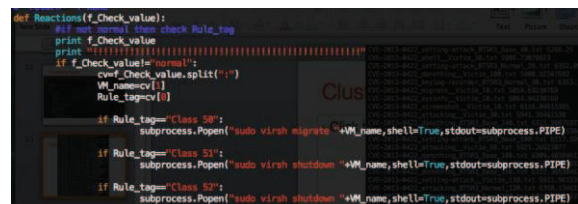


Figure 5.2.8 Reaction rules of each class

To detect malicious behaviors, we record system calls of VMs every 10 seconds, and compute the distance from expected malicious means according to the formula shown

in Figure 5.2.6. If it is within the “mean” or tolerance value of the class (e.g., Figure 5.2.7), its “Rule_tag” will be labeled with that class, e.g., “Class 52”. According to “Rule_tag”, VIS takes appropriate action on the target VM as shown in Figure 5.2.8

5.3 Deriving Detection Rules

In this section, we will explain how to derive rules from GHSOM results. We also discuss the defense rules used to defend against attacks from the hacker VM, as well as the recovery rules that we can use to recover compromised VMs. The tables bellow show attacks by period.

Detection rules

As you can see in the Table I we list six different kind of vulnerabilities number and exploits, for example, CVE-2013-0422 is a vulnerability of JAVA which can be attack by hacker with using a exploit from metasploit, in this case the hacker can use exploit which named “java_jre17_jmxben”, and vulnerability number CVE-2013-0431 can be attack by exploit named “java_jre17_jmxben”, vulnerability number MS12-020 can be attack by exploit named “ms12_020_maxchannelids”. On the other hand, some exploit of social engineering attack can be used without vulnerabilities, such as SET-Web-Java-Applet and SE_Firefox_xpi, hacker can use it to make a malicious link to deceive victim click the malicious link, even pass through the victim’s firewall and anti-virus applications.

In the Table I, rule 11 can be used to detect a DoS-hping3 attack, and rule 18_2 is used for others, including CVE-2013-0422, CVE-2013-0411, MS12-020, SET-Web-Java-Applet and SE_Firefox_xpi. Rule 18_2 allows VIS to detect attacks just as they begin against the Victim VM. Rule 23 can additionally be used to detect Java vulnerabilities CVE-2013-0422 and CVE-2013-0431.

TABLE III
RULES OF PERIOD OF ATTACKS FOR HACKER

Attacks/ Vulnerabilities	Period			
	Period 1	Period 2	Period 3	Period 4
CVE-2013-0422	Setting Attack	Attacking	Screenshot	Sysinfo
	Rule 18_2, 23	Rule 18_2, 23		
CVE-2013-0431	Rule 18_2	Rule 18_2, 23	Rule 18_2	Rule 18_2
Dos-hping3	Rule 11(12)			
MS12-020		Rule 18_2		
SET-Web-Java-Applet	Rule18_2	Rule 18_2		Rule 18_2
SE_Firefox_xpi		Rule 18_2		

Table III classifies data by attack, provides rules used for detection, and what data was acquired through the attack.

TABLE IV
MEAN AND UPPER BOUND OF RULES FOR HACKER VM

Rules	Bound	System call usage			
		<i>futex</i>	<i>futex_err</i>	<i>timer_settime</i>	<i>timer_settime_err</i>
rule_11	627.69	0	2768.75	469.5	0
rule_23	81.52	66.5	10.53	1050	83.33
rule_18_2	81.47	82	12.66	884.78	12.51

As you can see, the bound of rule 11 is 627.69, bond of rule 23 is 81.523437 and bound of rule 18_2 is 81.470703.

Recovery rules

We can use rules rule 47_2, 33, and 65_4 (see Table V) to detect attacks, such as CVE-2013-0422, CVE-2013-0431 and DoS-hping3. However, we can safely assume that not all users of cloud systems have the knowledge needed to recover data or otherwise cleanup their VM after it has been compromised. We have therefore decided to provide a recovery function "qemu-img" to detect compromised VMs.

TABLE V
MEAN AND UPPER BOUND OF RULES FOR VICTIM VM

Rules	Bound	System call usage			
		<i>futex</i>	<i>futex_err</i>	<i>timer_settime</i>	<i>timer_settime_err</i>
rule_47_2	28.23	1303	72	2056	0
rule_33	2059.9	32762	2078	1907	0
rule_65_4	44.31	1719	77.5	2086	0

Table V shows the upper bound of rule 47 as 28.29916, rule 33 as 2059.8958, and rule 65 as 44.3125.

5.4 Testing: Detection of attacks

In this section, we will show the result of tests of each rule. This section will demonstrate how attacks progress on VMs, how they are compromised, as well as demonstrate VIS in action.

Detection of malicious VMs

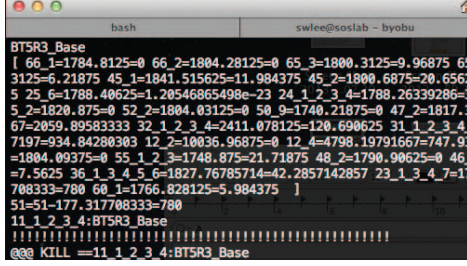


Figure 5.4.1 Attacks detected by rule 11

Figure 5.4.1 shows rule 11 detecting an attack and VIS sending the command to the controller to migrate the hacker VM (BT5R4_Base). Table IV shows rule 18_2 detecting attacks. VIS sends the command to shutdown the hacker VM (BT5R4_Base).

TABLE VI
RULES AND DETECTABLE ATTACKS

Rules	Exploits/Vulnerabilities					
	<i>CVE-2013-0422 JAVA</i>	<i>CVE-2013-0431 JAVA</i>	<i>DoS-hping3</i>	<i>MS12-020</i>	<i>SET-Web-Java-Applet</i>	<i>SE-Firefox-xpi</i>
Rule 11			v			
Rule 23	v	v				
Rule 18_2	v	v		v	v	v

Table VI shows that 18_2 is a special rule - able to detect almost every type of attack. The only exception is when the hacker VM boots into text mode, which prompts a false positive and is shutdown. Rule 23 also has a false positive, causing a shutdown during web browsing.

Detection of compromised VMs

Table VII shows rule 33 detecting a compromised VM. VIS will send a command to the controller to use the snapshot function to recover data from the victim VM.

TABLE VII
RULES AND DETECTABLE ATTACKS

Rules	Exploits/Vulnerabilities					
	<i>CVE-2013-0422(JAVA)</i>	<i>CVE-2013-0431(JAVA)</i>	<i>DoS-hping3</i>	<i>MS12-020</i>	<i>SET-Web-Java-Applet</i>	<i>SE-Firefox-xpi</i>
Rule 65_4	v	v				
Rule 47_2		v				
Rule 33			v			

We see in Table V that not every compromised VM can be detected - more experiments are needed to increase detection efficiency. However, rule 65_4 can detect the two

Java related attacks, Rule 47_2 can detect one form attack. If a VM under DoS attack is detectable under rule 33, then a change in the virtual network settings can stop a hacker VM's attack on the victim VM.

6. CONCLUSION

We propose VIS, a virtualization introspection system for KVM-based cloud platforms. This system can monitor both static and dynamic VM status, replay and classify various attacks to determine which VMs are attacking the Hypervisor, and determine which VMs are attacking other VMs. Additionally, VIS can also detect compromised VMs, as well as perform termination and online migration.

VIS is limited to protection based on established rules: it needs to collect more attack patterns. Additionally, the rules are derived by heuristics, have false positives and negatives, and require more sophisticated analysis, such as system call sequences.

References

- [1] A. Biermann and J. Feldman. On the synthesis of finite state machines from samples of their behavior. *IEEE Transactions on Computer*, 21:592-597, 1972
- [2] A. H. Sung, J. Xu, P. Chavez, S. Mukkamala. "Static Analyzer of Vicious Executables (SAVE)" *Proceedings of the 20th Annual Computer Security Applications Conference*, p.326-334, 2004
- [3] A. Seshadri, M. Luk, N. Qu, and A. Perrig. "SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes." In *16th SOSP*, Oct 2007, pp. 335-350
- [4] B.D. Payne, M. Carbone, M. Sharif, and W. Lee. "Lares: An Architecture for Secure Active Monitoring Using Virtualization." In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland 2008)*, May 2008, pp.233-247
- [5] D. Lo and S.-C. Khoo. Mining patterns and rules for software specification discovery. *Proc. VLDB Endow.*, 1(2):1609-1616, Aug. 2008
- [6] D. Lorenzoli, L. Mariani, and M. Pezz'e. Automatic generation of software behavioral models. In *Proceedings of the 30th international conference on Software engineering, ICSE '08*, pages 501-510, New York, NY, USA, 2008. ACM.
- [7] D. Zissis and D. Lekkas. "Addressing Cloud Computing Security Issues." *Future Generation Computer Systems*(28:3) March 2012, pp.583-592
- [8] ENISA. (2009, Feb) "Cloud computing: benefits, risks and recommendations for information security." Available: <http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computingrisk-assessment> [Jul. 10, 2010].
- [9] F. Bellard. "Qemu, a Fast and Portable Dynamic Translator," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, pp. 41-41
- [10] F. Lombardi and R. Di Pietro. "KvmSec: A Security Extension for Linux Kernel Virtual Machines." In *SAC '09: Proceedings of the 2009 ACM symposium on applied Computing*, ACM, New York, NY, USA, 2009. pp. 2029-34.

- [11] F. Siebenlist. "Challenges and opportunities for virtualized security in the clouds," Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, 2009, pages 1-2.
- [12] G.H. Kim and E.H. Spafford. "The Design and Implementation of Tripwire: A File System Integrity Checker." In CCS '94: Proceedings of the 2nd ACM conference on computer and communications security. ACM, New York, NY, USA, 1994. pp. 18-29
- [13] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas. "Opcode Sequences as Representation of Executables for Data-mining-based Unknown Malware Detection." Information Sciences, 2011.
- [14] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, L. Lo Iacono. "All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces." In Proceedings of the 3rd ACM workshop on Cloud computing security workshop (CCSW), Chicago, IL, USA, 17–21 October 2011. pp.3-14.
- [15] K. Rieck, T. Holz, C. Willems, P. Duessel, and P. Laskov. "Learning and Classification of Malware Behavior", Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), 2008.
- [16] Lombardi and Di Pietro. "Secure virtualization for cloud computing." Journal of Network and Computer Applications(34:4), July 2011, pp.1113-1122.
- [17] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.
- [18] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering, 27(2):99–123, 2001.
- [19] Metasploit. Metasploit. <http://www.metasploit/>, 2013.
- [20] Metasploit. Meterpreter. http://www.offensive-security.com/metasploit-unleashed/Meterpreter_Basics, 2013.
- [21] N. Elhage. Virtunoid: A KVM Guest -> Host privilege escalation exploit. In DEFCON, 2011.Black Hat USA 2011, August, 2008.
- [22] O. Cicchello and S. C. Kremer. Inducing grammars from sparse data sets: a survey of algorithms and results. Journal of Machine Learning Research, 4:603–632, 2003.
- [23] Openecp. Openecp. <http://www.openecp.org>, 2010.
- [24] RedHat. Libvirt. <http://libvirt.org>, 2007.
- [25] S. Bahram, X. Jiang, Z. Wang, M. Grace, J. Li, D. Srinivasan, J. Rhee, and D. Xu. DKSM: Subverting Virtual Machine Introspection for Fun and Profit, Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems Pages 82-91.
- [26] S.-W. Lee, D.-B. Tsai, A Guide to Having Fun with the Next Generation Linux, Ubuntu, ISBN: 9867199979, GrandTech Press,Taipei, Taiwan, Dec.2006.
- [27] S.-W. Lee, F. Yu, Virtualization Introspection System on KVM-based Cloud Computing Platforms, The 18th Conference on Information Management and Praticce, Taipei, Taiwan, Dec. 2012.
- [28] T. Foster, Y. Zhao, I. Raicu, S. Lu. "Cloud Computing Resource Management through a Grid Middleware: A Case Study with DIET and Eucalyptus." Cloud Computing, IEEE International Conference on, 2009. pp. 151-154.
- [29] T. Ormandy. "An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments." In CanSecWest, 2007.
- [30] T. Ristenpart, E. Tromer, H. Shacham, and Stefan Savage. "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds." In CCS '09: Proceedings of the 16th ACM conference on Computer, 2009, pp.199-212.
- [31] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A Tool for Analyzing Malware." 15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR), 2006.