Identifying Evidence for Implementing a Cloud Forensic Analysis Framework

Changwei Liu[#], Anoop Singhal[*], Duminda Wijesekera [#,*]

cliu6@gmu.edu, anoop.singhal@nist.gov, dwijesek@gmu.edu

[#] Department of Computer Science, George Mason University, Fairfax VA 22030 USA

[*] National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg MD 20899 USA

Abstract: Cloud computing provides several benefits to organizations such as increased flexibility, scalability and reduced cost. However, it provides several challenges for digital forensics and criminal investigation. Some of these challenges are the dependence of forensically valuable data on the deployment model, multiple virtual machines running on a single physical machine and multiple tenancies of clients. In this paper, we show what evidence from the cloud would be useful to construct the attack scenario by using a Prolog logic based forensic analysis tool. We propose to implement and design a forensic enabled cloud, which includes installing forensic tools in the cloud environment and logging all the activities from both the application layer and lower layers. Such an implementation can provide evidence for a Prolog based forensic tool, which can automate correlating the evidence from both the clients and the cloud service provider to construct attack steps and therefore re-create the attack scenarios on the cloud.

Keyword: Digital forensic analysis, cloud forensics, attack scenario, OpenStack

1. Introduction

Digital forensics is the application of science to identify, collect, examine, and analyze data while preserving information integrity and maintaining a strict chain of custody for the data during post incident examinations [1]. Being a component of digital forensics, network forensics analyzes network traffic in order to gather information from intrusion detection systems or logs to constitute legal evidence [2]. Considered as an emerging branch of network forensics, cloud forensics involves post-incident analysis of systems with distributed processing, multi-tenancy, virtualization and mobility of computations, which poses more challenges in identifying and preserving digital evidence, including [3]:

1. Dependence of forensically valuable data on the deployment model and methods. For example, customers of software as a service (SaaS) may have little or no control of the physical locations of their data.
2. Large volume in content and proprietary formats of data logs.
3. The diversity and the number of simultaneously operating virtual machines instances of a single physical machine isolated using virtualization. This takes extra efforts in segregating resources without breaching user confidentiality. In addition, weak registries in clouds make it easy for attackers to hide their traces.
4. Instances of servers running on virtual machines in the cloud monitored by hypervisors lack of warnings, procedures and tools for forensic investigation.

Although much research has progressed in digital forensics, the methods used in traditional digital forensics are inadequate for forensic investigation in clouds that have

been designed without much efforts dedicated for evidence retention and integrity. Recently, National Institute of Standards and Technology (NIST) and other researchers have published papers in cloud governance, security and risk assessment [4], and proposed implementing forensic-enabled clouds. For example, Dykstra et al. proposed implementing cloud to collect forensic data from operating system level underneath the virtual machines [2]. Zawod et al. provided complete, trustworthy, and forensic-enabled cloud architecture to collect logs for forensic analysis [3]. However, these implementations only focus on evidence acquaintance on Infrastructure-as-a-Service (IaaS) cloud deployment model. None of the work has discussed implementing forensic-enabled clouds that cover all deployment models. In this paper, we show what evidence can be used to construct corresponding attack scenarios in the cloud, and discuss how we may implement and automate the forensic analysis in the cloud using some example attacks with the objective of creating a deployment model.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 shows our experimental attacks in the cloud, and how we identify the evidence from the cloud to construct attack scenario by using a Prolog based tool. Section 4 shows how to use system call sequence to construct attack steps when other evidence is unavailable. We conclude the paper by discussing how we may implement and automate the forensic analysis in the cloud as our future work in Section 5.

2. Background and Related Work

We present the background and research related to digital and cloud forensics in this section.

2.1 Digital Forensics

Digital forensics uses scientifically accepted methods to collect, validate and preserve digital evidence derived from digital sources for the purpose of reconstruction of events found to be criminal or helping to anticipate unauthorized actions shown to be disruptive to planned operations [5]. Digital forensic investigators seek attack evidence from computers and networks. Typically, imaging tools are used to extract a computer's physical memory or disk sectors to a file, and then the investigators feed the file into data analysis tools to perform live or dead analysis. For network evidence, forensic investigators analyze network traffic and gather information from intrusion detection systems or logs to constitute legal evidence.

2.2 Cloud Forensics

NIST defined cloud model [6] uses three service deployment models: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). SaaS allows consumers to use the provider's applications running on a cloud infrastructure. PaaS allows consumers to deploy on the cloud consumer-created or acquired applications using programming languages, libraries, services and tools supported by the provider. IaaS provides consumers with the capability of provisioning processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software including operating systems and applications.

According to Ruan et al., cloud forensics is a subset of network forensics [3] that follows the main phases of network forensics with techniques tailored to cloud computing environments. For example, data acquisition is different in SaaS and IaaS, because the investigator will have to solely depend on cloud service provider in SaaS. With IaaS, the investigator can acquire the virtual machine image from customers.

2.3 Related Work

According to many researchers, data acquisition is a main issue in cloud forensics. Many methods have been proposed to collect evidence from clouds, which include remote data acquisition, management plane, live forensics and snapshot analysis [7]. Dykstra et al. successfully retrieved volatile and nonvolatile data from Amazon EC2 cloud's active user instance platform using forensic tools such as Guidance EnCase and Access Data FTK [8]. However, those tools do not validate data integrity. Researchers recommended and developed some toolkits to collect related logs from cloud infrastructure while assuring their integrity. Assuming the cloud provider is trustworthy, Dykstra et al. developed the FROST toolkit that can be integrated to OpenStack [9] to collect logs from the operating system level supporting the virtual machines [10]. Zawod et al. designed complete, trustworthy, and forensic-enabled cloud architecture for log collection to address this trust issue [11]. Hay et al. proposed live digital forensics analysis on clouds using virtual introspection, a process by which the state of a virtual machine (VM) is observed from either the hypervisor (VMM) or from some other virtual machine and presented a suite of virtual introspection tools developed for Xen (VIX tools) [12]. However, live forensic tools have not been incorporated and provided as a commercial service by the cloud service providers. Snapshot technologies enable customers to freeze a specific state of VM [13]. The snapshot images can be restored by loading them to a target VM for analysis, gaining information on the running state of a virtual machine that is supported by hypervisor vendors, including Xen, VMWare, ESX, Hyper-V, and cloud providers that support snapshot features.

In addition, many tools like Encase, the Sleuth Kit, SNORT, WireShark can collect

digital evidence from computers and networks. In order to reduce the investigators' time and effort in constructing attack steps, researchers proposed using rules to automate correlating evidence by finding the causality between items of evidence [14, 15]. Liu et al. integrated the tool with two databases, including a vulnerability database and an anti-forensic database, to ascertain the admissibility of evidence and explain missing evidence due to attackers' using anti-forensics [15]. These rule based forensic analysis frameworks have been proposed for network forensics, but have not been tested in a cloud environment.

3. Using Alerts and Logs to Construct Attack Scenario

In this section, we describe experimental attacks we launched on OpenStack [9] to identity evidence that can be used for cloud forensic analysis.

3.1 Experimental Environment Setup

OpenStack is a collection of python-based software projects that manage access to pooled storage, computing and network resources that reside in one or multiple machines of a cloud. This collection has six core projects: Neutron (Networking), Nova (Compute), Glance (Image Management), Swift (Object Storage), Cinder (Block Storage) and Keystone (Authorization and Authentication) [9]. OpenStack can be used to deploy three service models--SaaS, PaaS and IaaS, but is mostly deployed as IaaS.

"DevStack" is a series of extensible scripts that can invoke an OpenStack environment using the latest versions of the software. We deployed OpenStack "Juno" version as IaaS cloud on a single computer (with IP address 172.16.168.100) by running "Devstack" on an Ubuntu 14.04 Desktop. Authenticated users can access OpenStack services using the IP address 172.16.168.100 on their browser to access the control dashboard Horizon as shown in Figure 1.

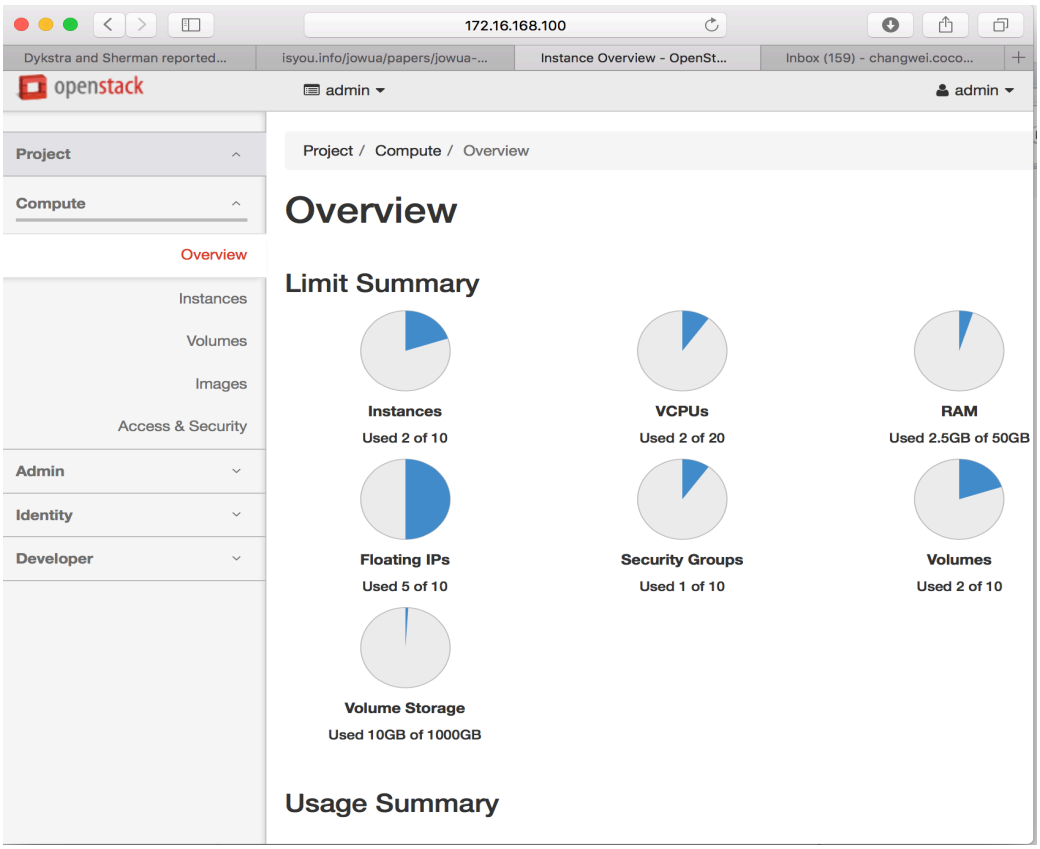

Figure 1. OpenStack web user interface--Horizon

We deployed two virtual machines (also called running instances), a webserver (named "WebServer" associated with IP address 172.16.168.226) and a fileserver (named "FileServer" associated with IP address 172.16.168.229) under the authenticated user "Admin" in our OpenStack cloud. In the "WebServer", we deployed an Apache webserver and a MySQL database, allowing users to query their data using the webserver. Authenticated users can access the "FileServer" by remotely using "ssh". In order to launch an attack, we also installed Kali (the penetration testing and ethical hacking Linux distribution tool [16]) in the same network (with IP address 172.16.168.173).

3.2 Example Attacks

We launched three attacks, a SQL injection attack, a DDoS attack, and a DoS attack towards the two VMs in our IaaS cloud.

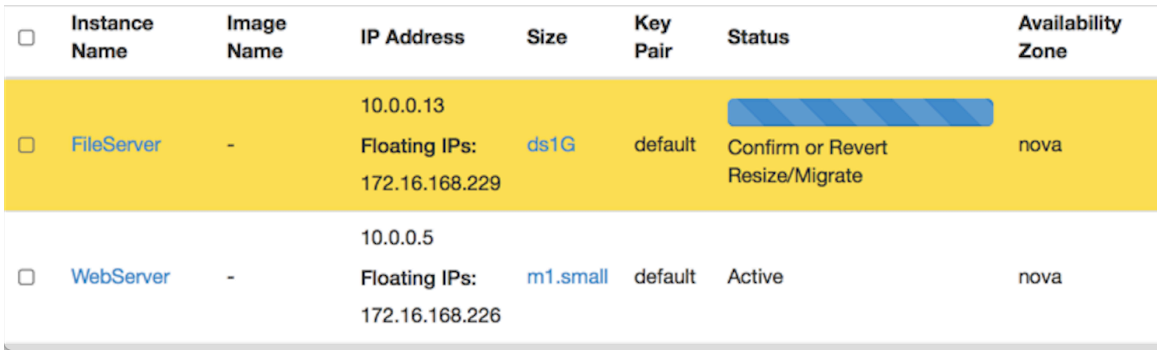

Figure 2. Resizing “FileServer”

Our SQL injection attack exploits un-sanitized user inputs (CWE89) in the “WebServer”. Our DDoS attack known as "TCP connection flood" used “nping” in Kali to flood the “FileServer” in order to prevent legitimate requests. While SQL injection and DDoS attacks can happen to any network including a cloud that has corresponding vulnerability, only IaaS privileged users can resize and delete a VM by launching DoS attacks that exploit the vulnerability “CVE-2015-3241”. According to NIST’s NVD, the vulnerability “CVE-2015-3241” that is in OpenStack Compute (Nova) versions 2015.1 through 2015.1.1, 2014.2.3 allows authenticated users to cause denial of services by resizing and then deleting an instance (VM). The process of resizing and deleting an instance in this way is also called instance migration. With “CVE-2015-3241”, the migration process does not terminate when an instance is deleted, so an authenticated user could bypass user quota enforcement to deplete all available disk space by repeatedly performing instance migration. Figure 2 shows the process of our resizing the file server from “ds512M” to “ds1G”, where we can see the instances’ availability zone is “nova”. We continued to resize and delete instances until Nova was so depleted that it could not accept any new instance.

3.3 Identifying Evidence to Reconstruct Attack Scenarios

In order to obtain evidence for forensic analysis, we configured the webserver and the SQL database in “WebServer” to log access and query history. We also installed Snort in “WebServer” and “FileServer” VMs and deployed WireShark in the host Ubuntu OS to monitor the network traffic. Snort was able to capture the SQL injection attack and generated alerts with appropriate rules. Also, WireShark was able to capture packets that formed the DDoS attack. Figure 3 lists some SNORT alerts and MySQL query log of the SQL injection attack, which shows the attack was done by using “ or ‘1’=‘1’ ” to bypass the SQL query condition check. The snapshot of packets captured by WireShark is listed in Figure 4, where we can see Kali Linux at 172.16.168.173 sent out numerous SYN packets to “FileServer” at 172.16.168.229, and the “FileServer” sent numerous SYN-ACK packets back to Kali Linux.

```
 [**] SQL Injection Attempt --1=1 [**]
08/16-14:37:27.818279 172.16.168.173:1715 -> 172.16.168.226:80
TCP TTL:128 TOS:0x0 ID:380 IpLen:20 DgmLen:48 DF
******S* Seq: 0xDEDBEABF  Ack: 0x0  Win: 0xFFFF  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK


160813 14:37:29 40 Connect
...
40 QuerySET GLOBAL general_log = 'ON' 40 Queryselect * from profiles where
name='Alice' AND password='alice' or '1'='1'
Gen_log 2: 130813 14:39:56
…
```

Figure 3. The SNORT alert and the MySQL database log

No.	Time	Source	Destination	Protocol	Length	Info
217	10.405625326	172.16.168.173	172.16.168.229	TCP	74	34818 → 80 [SYN] Seq=0
218	10.405682554	172.16.168.173	172.16.168.229	TCP	74	44208 → 80 [SYN] Seq=0
219	10.405746104	172.16.168.173	172.16.168.229	TCP	74	38032 → 80 [SYN] Seq=0
220	10.408041819	172.16.168.173	172.16.168.229	TCP	74	34348 → 80 [SYN] Seq=0
221	10.408111539	172.16.168.173	172.16.168.229	TCP	74	38769 → 80 [SYN] Seq=0
222	10.408205849	172.16.168.173	172.16.168.229	TCP	74	36846 → 80 [SYN] Seq=0
223	10.408275950	172.16.168.173	172.16.168.229	TCP	74	35307 → 80 [SYN] Seq=0
224	10.408329211	172.16.168.229	172.16.168.173	TCP	60	80 → 41930 [RST, ACK]
225	10.408355690	172.16.168.229	172.16.168.173	TCP	60	80 → 44471 [RST, ACK]
226	10.408388686	172.16.168.173	172.16.168.229	TCP	74	35276 → 80 [SYN] Seq=0
227	10.408430802	172.16.168.229	172.16.168.173	TCP	60	80 → 45714 [RST, ACK]
228	10.408465024	172.16.168.229	172.16.168.173	TCP	60	80 → 35431 [RST, ACK]

Figure 4: A Snippet of packets caught by WireShark

```
/*The initial attack status and final attack status*/
attackerLocated(internet).
attackGoal(serviceDown(fileServer,user)).
attackGoal(execCode(database,user)).

/* The network topology and computer configuration*/
/* "_" means any port */
hacl(internet, webServer, tcp, 80).
hacl(internet, fileServer, tcp, _).
directAccess(webServer,database,modify,user).

/* The evidence found in webServer */
vulExists(webServer, 'SQLInjection', httpd).
vulProperty('SQLInjection', remoteExploit, privEscalation).
networkServiceInfo(webServer , httpd, tcp , 80 , user).

/* The evidence captured by WireShark*/
vulExists(fileServer,'DDoS', httpd).
vulProperty('DDoS', remoteExploit, privEscalation).
networkServiceInfo(fileServer, httpd, tcp, _, user).
```

Figure 5. Prolog predicates for SQL injection and DDoS attack evidence

We used our Prolog based tool presented in [15] to automate the process of correlating items of evidence to generate attack scenarios. To do so, we converted the above evidence and the cloud configuration to corresponding Prolog predicates as the input file in Figure 5. Our Prolog based tool uses rules to correlate these items of evidence represented by Prolog predicates to construct attack paths. The constructed attack paths are shown in Figure 6, and the notation of all nodes is listed in Table 1. In this graph model, an attack status obtained from the attacked system is represented by a diamond. All computer configuration, network topology and software vulnerabilities used to launch an attack are represented by boxes and a rule used to correlate attack status before and after an attack step is represented by an ellipse [15]. The software vulnerability exploited to launch an attack is obtained by forensic investigators' judgment on the evidence collected from the attacked system. Two attack paths are shown in Figure 6. The left path

(8⟶6⟶5⟶4⟶3⟶2⟶1) represents the SQL injection attack that used the web server vulnerability to maliciously obtain the information from the MySQL database. The right path (8⟶15⟶14⟶13⟶12) represents the DDoS attack to bring down the "FileServer".

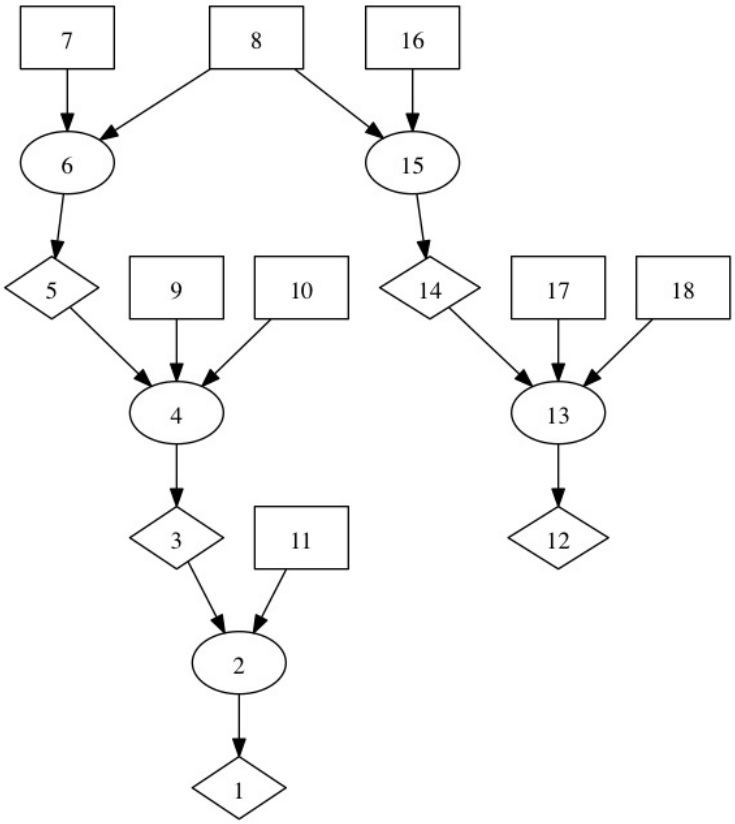

Figure 6. The attack path constructed for SQL injection and DDoS attacks

SNORT and WireShark failed in capturing our DoS attack on the "FileServer" that exploited the "CVE-2015-3241" vulnerability on OpenStack Nova service. Because OpenStack service application programming interface (API) logs provide information about users' operations on the running instances, we used the OpenStack service API logs as evidence. Figure 7 lists a snippet of Nova API logs that are related to our instance migration of the DoS attack, where the commands in bold font show the instance

"bd1dac18-1ce2-44b5-93ee-967fec640ff3" representing the "FileServer" VM (as shown in Table 2, which is obtained by running "nova list" in the Ubuntu host system.) has been resized using commands "mv" (move) and "mkdir" (create new directory) operated by the user "admin". We aggregated the related Nova API calls as evidence to form the input file with the corresponding attack status and the system configuration (Figure 8). By running our Prolog based tool [15] on the input file, we obtained the attack scenario as shown in Figure 9 with the notation of all nodes in Table 3. The figure shows the attack path that used the control dashboard "Horizon" exploiting the "CVE-2015-3241" vulnerability.

Table 1. Notation of all nodes in Figure 6

Node Number	Notation of the Node
1	execCode(database,user)
2	THROUGH 7 (Attack by compromised computer)
3	execCode(webServer,user)
4	THROUGH 3 (remote exploit of a server program)
5	netAccess(webServer,tcp,80)
6	THROUGH 9 (direct network access)
7	hacl(internet,webServer,tcp,80)
8	attackerLocated(internet)
9	networkServiceInfo(webServer,httpd,tcp,80,user)
10	vulExists(webServer,'SQLInjection',httpd,remoteExploit,privEscalation)
11	directAccess(webServer,database,modify,user)
12	execCode(fileServer,user)
13	THROUGH 3 (remote exploit of a server program)
14	netAccess(fileServer,tcp,_)
15	THROUGH 9 (direct network access)
16	hacl(internet,fileServer,tcp,_)
17	networkServiceInfo(fileServer,httpd,tcp,_,user)
18	vulExists(fileServer,'DDoS',httpd,remoteExploit,privEscalation)

2016-09-18 07:52:00.237 DEBUG oslo_concurrency.processutils [req-f79c7911-04ed-4a0c-adbe-0ae0a487c0f7 admin admin] Running cmd (subprocess): **mv /opt/stack/data/nova/instances/bd1dac18-1c e2-44b5-93ee-967fec640ff3 /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3_resize from (pid=41737) execute /usr/local/lib/python2.7/dist-packages/oslo_concurrency/processutils.py:344**

2016-09-18 07:52:00.253 DEBUG oslo_concurrency.processutils [req-f79c7911-04ed-4a0c-adbe-0ae0a487c0f7 admin admin] CMD **"mv /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3 /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3_resize"** returned: 0 in 0.016s from (pid=41737) execute /usr/local/lib/python2.7/dist-packages/oslo_concurrency/processutils.py:374

2016-09-18 07:52:00.254 DEBUG oslo_concurrency.processutils [req-f79c7911-04ed-4a0c-adbe-0ae0a487c0f7 admin admin] Running cmd (subprocess): **mkdir –p /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3** from (pid=41737) execute /usr/local/lib/python2.7/dist-packages/oslo_concurrency/processutils.py:344

2016-09-18 07:52:00.271 DEBUG oslo_concurrency.processutils [req-f79c7911-04ed-4a0c-adbe-0ae0a487c0f7 admin admin] CMD **"mkdir –p /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3"** returned: 0 in 0.017s from (pid=41737) execute /usr/local/lib/python2.7/dist-packages/oslo_concurrency/processutils.py:374

Figure 7. Nova API Call Logs

Table 2. The VM instance IDs, names and IPs

ID	Name	…	Networks
bd1dac18-1ce2-44b5-93ee-967fec640ff3	FileServer		private=10.0.0.13, 172.16.168.229
c01d5e66-c20d-4544-867b-d3e2b70bfc60	WebServer		private=10.0.0.5, 172.16.168.226

/* the initial and final attack status*/
attackerLocated(controlDashboard).
attackGoal(execCode(nova,admin)).

/* the fileserver VM could be reached from control dashboard*/
hacl(controlDashboard, fileServer, http, _).

/* the evidence of attack using 'CVE-2015-3241' that uses RESTful service*/
vulExists(nova,'CVE-2015-3241', 'REST').
vulProperty('CVE-2015-3241', remoteExploit, privEscalation).
networkServiceInfo(nova, 'REST', http, _, admin).

Figure 8. The input file for attack using "CVE-2015-3241"

Figure 6 and Figure 9 cannot be grouped together, because attackers were in different locations. In addition, in Figure 9, the attack happened in the cloud compute service instead of a VM, although the attacker launched the attack from a VM. This is because all VMs share the same compute service in our cloud.

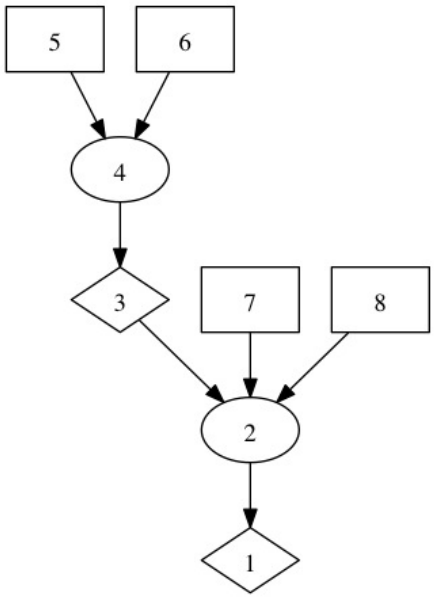

Figure 9. The attack path constructed for the DoS attack

Table 3. The notation of all nodes in Figure 9

Node Number	Notation of the Node
1	execCode(nova,admin)
2	THROUGH 3 (remote exploit of a server program)
3	netAccess(nova,http,_)
4	THROUGH 9 (direct network access)
5	hacl(controlDashboard,nova,http,_)
6	attackerLocated(controlDashboard)
7	networkServiceInfo(nova,'REST',http,_,admin)
8	vulExists(nova,'CVE-2015-3241','REST',remoteExploit,privEscalation)

4. Using System Call Invocations for Evidence Analysis

Because system calls allow user level processes to request kernel level services including access to storage operations, memory or network access, and process management, system calls sequence is often used for intrusion detection and forensics [17]. When evidence or expert knowledge is unavailable to recognize the interaction between user level processes to kernel level services as a known attack, forensic investigators analyze the system calls to ascertain program behavior. According to [18], it is rare or

unlikely to have an attack path, in which every attack step is a zero-day attack. As such, we use system calls to construct the missing attack steps only when other evidence is not available.

There are several popular mechanisms to trace the system calls in a cloud based VM: (1) use "ptrace" command to set up system call interception and modification by modifying a software application, (2) use "strace" command to log system calls and signals, (3) use auditing facilities within the kernel, (4) modify the system call table and write system call wrappers to log the corresponding system calls, (5) intercept the system call within the hypervisor [19]. Because OpenStack supports different hypervisors, including Xen, QEMU, KVM, LXC, Hyper-V and UML, there is no a generic solution to intercept the system call within the hypervisor. Thus, we use methods 2 and 4 to log relevant system calls.

```
Sep 25 00:15:49 FileServer sshd[829]: Server listening on 0.0.0.0 port 22.
Sep 25 00:15:49 FileServer sshd[829]: Server listening on :: port 22.
Sep 25 00:28:15 FileServer sshd[1162]: Accepted password for coco from 172.16.168.173 port
44842 ssh2
Sep 25 00:28:16 FileServer sshd[1162]: pam_unix(sshd:session): session opened for user coco by
(uid=0)
```

Figure 10. The authentication log for sshd

Now we show how to use system call sequences to construct an attack step by using an attack example. In this experimental attack launched from our Kali Linux, we, as the attacker, used ssh to log into "FileServer" by using stolen credentials from a legitimate user named "coco". In order to simulate the stealthy attack without triggering IDS alerts, we assumed that the attacker could use social engineering attacks, such as shoulder surfing, to obtain the legitimate user's (username, password) pair to log into the "FileServer" using ssh. The corresponding sshd log from "/var/log/auth.log" in "FileServer" is listed in Figure

10, where user “coco” was listed to log in “FileServer” from “172.16.168.173” that actually belonged to the attacker, which indicates that the attacker stole user coco’s credentials.

A process typically comprises of many system calls, of which only some generally are important to ascertain a process’ behavior (we use the ones presented in [18]. These system calls are listed in the second column of Table 4). Figure 11 is a snippet of important system calls captured from the attack of using coco’s stolen credentials to modify a file in “FileServer” (due to space limitations, we list a part of captured system calls). By analyzing these system calls, we notice that the “write/read” system calls (in bold font) indicate that the attacker used “vi test.txt”(“vi” is a text editor) command to modify “test.txt” file. In the “write/read” system call, the first argument is the file descriptor where the process reads or writes, the second argument represents the content in the buffer, the third argument represents how many bytes the system call will write/read, and “=1/<any number greater than 1>” indicates that the system call executed successfully.

Table 4: Important System Calls

Tasks	System Calls
Process modifies file	write, pwrite64,rename, mkdir, linkat, link, symlinkat, symlink, fchmodat, fchmod, chmod, fchownat, mount
Process uses but does not modify file	stat64,lstat6e,fsat64, open, read, pread64, execve, mmap2, mprotect, linkat, link, symlinkat, symlink
Process uses and modifies file	open, rename, mount, mmap2, mprotect
Process creation or termination	vfork, fork, kill
Process creation	Clone

write(9, "v", 1) = 1
read(11, "v", 16384) = 1
write(3, "\0\0\0\20\331\255\275\264c\2173)z2j\32\255
n\2007d\366m\21\316\2648\240\207\31\211"..., 36) = 36
read(3,
"\0\0\0\20\240\253\341\227\321xU\305\347\226\246\361\316\242S=\30\341QT\231\n\343\314\34
3\307\f\361"..., 16384) = 36
write(9, "i", 1) = 1

read(11, "i", 16384) **= 1**
write(3,
"\0\0\0\20\177\352\313\332\373yjM\3416l\230\215\10\220p\252g\375\365\1\f\335\361\r\273\374\
357"..., 36) = 36
read(3,
"\0\0\0\20\27\334?\201x\300\16\356\346,\0379\32\220{\372)\366\4\v\1=\347\263\311\250k\353"...
, 16384) = 36
write(9, " ", 1) = 1
read(11, " ", 16384) = 1
write(3,
"\0\0\0\20`i\321\344\220\313\322\254S\252o\201\225;6v\243\205\10gs^\253\237\325\375\332v"...
, 36) = 36
read(3, "\0\0\0\20\5\27k;\254\301\24\n\\ZN\267\260\336\323`\323\32\345\2b\226-\271|[B\21"...,
16384) = 36
write(9, "t", 1) **= 1**
read(11, "t", 16384) **= 1**
read(3,
"\0\0\0\20\325\261\7\254\211(\201\331\272\344[\355\200\\u4\357G\347\232\276:\201\376\342\20
2\201."..., 16384) = 36
write(3,
"\0\0\0\20\320\254#\312\211_\3022\n\227u\16I\372\202\347\37\252T\257\220\210E\343\222\342\
24S"..., 36) = 36
write(9, "e", 1) **= 1**
read(11, "e", 16384) **= 1**
write(3, "\0\0\0\20\334n}4\375Q\212o\353\375\262\342\316\334w-
F\213\303\277t\312\245\16\266\255B|"..., 36) = 36
read(3, "\0\0\0\20\274\376\7J\214L\314OL\1c\22\364-gvJ%\21\344J<,h\363\261\36\10"..., 16384)
= 36
write(9, "\t", 1) **= 1**
read(11, "st.txt ", 16384) **= 7**
…

Figure 11. Traces of "Read" and "Write" system calls

//The initial attack status
attackerLocated(internet).
// the attacker was able to log into "FileServer" by using stolen credentials
attackGoal(logInService(fileserver, tcp,22)
attackGoal(princinpalCompromised(user))
//InCompetent user
InCompetent(user).

//The attack status obtained from analyzing system call sequence
attackGoal(canAccessFile(fileServer,user,modify,_)).
//The user could login fileserver by using ssh protocol
networkServiceInfo(fileServer , sshd, tcp, 22, _).
//the user who has the account on "FileServer" has the privilege to modify a file
localFileProtection(fileServer,user,modify,_).

Figure 12. Input file for the attack of modifying a file with stolen credentials

We converted the program behavior learned from the system call sequence in Figure 11, that is the attacker's using a text editor to convert "test.txt" file, to Prolog Predicate "canAccessFile(fileServer,user,modify,_)" (This predicate means that the attacker as the user can modify the file located at "_" representing the home directory of the user). With the evidence obtained from the log in Figure 10 that the attacker with stolen credentials (represented by predicates "attackGoal(princinpalCompromised(user))", "InCompetent(user)" and "attackerLocated(internet)") logged into the "FileServer" by using ssh (represented by Predicate "attackGoal(logInService(fileserver, tcp,22)"), and the fact user "coco", who has an account on "FileServer", has the privilege to modify a file (the corresponding predicate is "localFileProtection(fileServer,user,modify,_)"), we formed the input file as Figure 12 to use our Prolog based tool. The constructed attack path is shown in Figure 13, and the notation of all nodes is in Table 5. In Figure 13, the attack step (3, 4, 7) $\longrightarrow$ 2$\longrightarrow$1 has two pre-conditions represented by Node 4 and Node 7. Node 4 is obtained from the fact that the "FileServer" can be accessed by using ssh with protocol tcp from port 22. Node 7 is obtained from ssh authentication log in Figure 10 that indicates the user's credentials have been stolen by the attacker. Without the evidence obtained from the system call sequence (Node 1), the attack step (3, 4, 7) $\longrightarrow$ 2$\longrightarrow$1 would not have been established.

Notice the two rule nodes (Node 5 and Node 2) in Figure 13 do not have any rule description because of the obvious correlation between Node 6 and Node 4 (if the network provides the service of using ssh to log into a fileserver by using tcp at port 22, the user including the attacker could log into the fileserver with stolen credentials), nodes (3,4,7) and Node 1 (if a user is allowed to have the privilege of modifying a file in fileserver, the

attacker with the stolen credentials from the user could access the file and modify it).

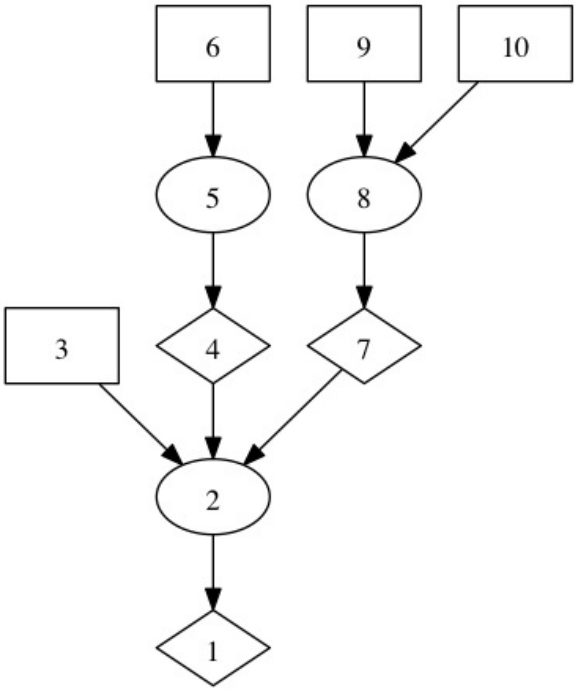

Figure 13. The attack step constructed by using evidence obtained from system calls

Table 5. The notation of all nodes in Figure 13

Node Number	Notation of Node
1	canAccessFile(fileserver,user, modify,_)
2	THROUGH 23()
3	localFileProtection(fileserver,user,modify,_)
4	logInService(fileserver,tcp,22)
5	THROUGH 18 ()
6	networkServiceInfo(fileserver,sshd,tcp,22,user)
7	princinpalCompromised(user)
8	THROUGH 16(password sniffing)
9	inCompetent(user)
10	attackerLocated(internet)

5. Conclusion and Future work

The use of cloud computing can increase the flexibility and efficiency of organizations or enterprises. However, clouds present significant challenges to forensics,

including customers' lack of control of the physical locations of their data, the large volume of data logs and the prevalence of proprietary formats. To solve the above problems, we plan to implement a forensic-enabled cloud by exploring what evidence could be useful for cloud forensic analysis.

Our example attacks show evidence from three resources could help investigators to construct attack scenarios, which include (1) evidence from IDS and application software logging, (2) cloud service API calls, and (3) system calls from VMs. To acquire the evidence from the three resources, the forensic-enabled cloud should have three extensions, which can (1) retrieve IDS and software service logging; (2) store and secure OpenStack service API call logs, firewall logs and snapshots for running instances; (3) obtain system calls when the evidence from (1) and (2) is missing. Our future plan addresses implementing forensic-enabled cloud with the above extensions and resolving the corresponding problems including assuring the data integrity, reducing the large volume and formalizing the proprietary of forensic data stored in the cloud.

DISCLAIMER

References

[1] Kent K, Chevalier S, Grance T and Dang H. "Guide to integrating forensic techniques into incident response". 2006. p. 800e86. NIST Special Publication.

[2] Gary Palmer. "A Road Map for Digital Forensic Research". Report from DFRWS 2001,

First Digital Forensic Research Workshop, Utica, New York, August 7 – 8, 2001, Page(s) 27–30.

[3] Ruan, Keyun, Joe Carthy, Tahar Kechadi, and Mark Crosbie. "Cloud forensics." In IFIP International Conference on Digital Forensics, pp. 35-46. Springer Berlin Heidelberg, 201.

[4] M. Hogan, F. Liu, A. Sokol,J. Tong. "NIST cloud computing standards roadmap." NIST Special Publication 35 (2011).

[5] A. Jaquith, "Security Metrics: Replacing Fear, Uncertainty, and Doubt", Addison Wesley, Mar 26, 2007.

[6] P. Mell and T. Grance. "NIST definition of cloud computing". National Institute of Standards and Technology. October 7, 2009.

[7] Pichan, Ameer, Mihai Lazarescu, and Sie Teng Soh. "Cloud forensics: technical challenges, solutions and comparative analysis." Digital Investigation 13 (2015): 38-57.

[8] J. Dykstra and A.T. Sherman, "Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques," in Proc. of the 12th Annual Digital Forensics Research Conference (DFRWS'12), Washington, DC, USA, Digital Investigation, vol. 9, August 2012, pp. 90–98.

[9] OpenStack Open Source Cloud Computing Software. Retrieved from https://www.openstack.org.

[10] J. Dykstra and A. Sherman, "Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform," Digital Investigation, vol. 10, no. Supplement, p. S87–S95, 2013.

[11] S. Zawoad and R. Hasan, "FECloud: A Trustworthy Forensics-Enabled Cloud Architecture," Proc. 11th Ann. Int'l Fed. Info. Processing WG 11.9 Int'l Conf. Digital Forensics, 2015, pp. 271–285.

[12] Hay, Brian, and Kara Nance. "Forensics examination of volatile system data using virtual introspection." ACM SIGOPS Operating Systems Review 42.3 (2008): 74-82.

[13] D. Birk, C. Wegener. "Technical issues of forensic investigations in cloud computing environments". In 6th International workshop on systematic approaches to digital forensic engineering–IEEE/SADFE 2011, Oakland, CA, USA; 2011, p. 1–10.

[14]] W. Wang, E.D. Thomas, "A graph based approach toward network forensics analysis", ACM Transactions on Information and Systems Security 12 (1) 2008.

[15] C. Liu, A. Singhal, D. Wijesekera. "A Logic Based Network Forensics Model for Evidence Analysis". IFIP Int. Conf. Digital Forensics 2015.

[16] Kali Linux--Penetration Testing and Ethical Hacking Linux Distribution. Retrieved from https://www.kali.org.

[17] Hofmeyr, Steven A., Stephanie Forrest, and Anil Somayaji. "Intrusion detection using sequences of system calls." Journal of computer security 6, no. 3 (1998): 151-180.

[18] X. Sun, J. Dai, A. Singhal, P. Liu and J. Yen, “Towards Probabilistic Identification of Zero-day Attack Paths”, Accepted for IEEE Conference on Communication and Network Security, Philadelphia, October 17th – 19th, 2016.

[19] F. Beck and O. Festor. "Syscall interception in xen hypervisor." (2009): 19.