Chapter 7

IDENTIFYING EVIDENCE FOR CLOUD FORENSIC ANALYSIS

Changwei Liu, Anoop Singhal and Duminda Wijesekera

Abstract Cloud computing provides benefits such as increased flexibility, scalability and cost savings to enterprises. However, it introduces several challenges to digital forensic investigations. Current forensic analysis frameworks and tools are largely intended for off-line investigations and it is assumed that the logs are under investigator control. In cloud computing, however, evidence can be distributed across several machines, most of which would be outside the control of the investigator. Other challenges include the dependence of forensically-valuable data on the cloud deployment model, large volumes of data, proprietary data formats, multiple isolated virtual machine instances running on a single physical machine and inadequate tools for conducting cloud forensic investigations.

This research demonstrates that evidence from multiple sources can be used to reconstruct cloud attack scenarios. The sources include: (i) intrusion detection system and application software logs; (ii) cloud service API calls; and (iii) system calls from virtual machines. A forensic analysis framework for cloud computing environments is presented that considers logged data related to activities in the application layer as well as lower layers. A Prolog-based forensic analysis tool is used to automate the correlation of evidence from clients and the cloud service provider in order to reconstruct attack scenarios in a forensic investigation.

Keywords: Cloud forensics, attack scenarios, OpenStack

1. Introduction

Digital forensics involves the identification, collection, examination and analysis of data while preserving its integrity and maintaining strict chain of custody during post-incident investigations [9]. Network forensics is a component of digital forensics that primarily focuses on the analysis of network traffic and other data from intrusion detection systems

G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics XIII, IFIP AICT 511, pp. 111–130, 2017.
DOI: 10.1007/978-3-319-67208-3_7

and logs [14]. Cloud forensics is an emerging branch of network forensics, which involves post-incident analysis of systems with distributed processing, multi-tenancy, virtualization and mobility of computations. Ruan et al. [16] identify several challenges associated with cloud forensics. These include the dependence of forensically-valuable data on the cloud deployment model and methods, large volumes of data, proprietary data formats, large numbers of diverse, simultaneously-executing virtual machine instances, lack of monitoring and alerts by hypervisors that run virtual machines, and limited techniques and tools designed specifically for cloud forensic investigations.

The National Institute of Standards and Technology (NIST) [7] has published a cloud computing standards roadmap that emphasizes cloud governance, security and risk assessment. A key recommendation in the roadmap and by members of the digital forensics research community [14, 16] is the implementation of forensics-enabled clouds. However, most approaches focus on evidence gathering from infrastructure-as-a-service cloud model deployments. No formal approach currently exists for reconstructing attack scenarios based on evidence collected in virtualized cloud environments. This research demonstrates that evidence from multiple sources can be used to reconstruct cloud attack scenarios. The sources include: (i) intrusion detection system and application software logs; (ii) cloud service API calls; and (iii) system calls from virtual machines. A Prolog-based forensic analysis tool is used to automate the correlation of evidence from the three sources in order to reconstruct attack scenarios in cloud forensic investigations.

2. Background and Related Work

Cloud computing has three principal service deployments: (i) software-as-a-service (SaaS); (ii) platform-as-a-service (PaaS); and (iii) infrastructure-as-a-service (IaaS) [12]. A software-as-a-service model enables consumers to use service provider applications running on a cloud infrastructure. A platform-as-a-service model allows consumers to deploy their own applications or acquired applications using programming languages, libraries, services and tools supported by the service provider. An infrastructure-as-a-service model provides consumers with the ability to provision processing, storage, networks and other fundamental computing resources, including operating systems and applications.

Cloud forensics is a subset of network forensics that uses techniques tailored to cloud computing environments [16]. For example, data acquisition is different in the software-as-a-service and infrastructure-as-a-service models because an investigator has to depend entirely on the

cloud service provider in the case of a software-as-a-service model whereas an investigator can acquire virtual machine images from a customer in an infrastructure-as-a-service model.

Several techniques have been proposed to collect evidence from cloud environments, including remote data acquisition, management plane acquisition, live forensics and snapshot analysis [15]. Dykstra and Sherman [3] have retrieved volatile and non-volatile data from the Amazon EC2 cloud active user instance platform using traditional forensic tools such as EnCase and FTK. However, these tools do not validate the integrity of the collected data. Dykstra and Sherman [4] subsequently developed the FROST toolkit, which can be integrated within OpenStack to collect logs from the operating system that runs the virtual machines; this technique assumes that the cloud provider is trustworthy. Zawoad et al. [19] have designed a complete, trustworthy and forensics-enabled cloud.

Hay and Nance [5] have conducted live digital forensic analyses on clouds with virtual introspection, a process that enables the hypervisor or any other virtual machine to observe the state of a chosen virtual machine. They also developed a suite of virtual introspection tools for Xen (VIX tools). At this time, live forensic tools have not been incorporated as a commercial service by cloud providers.

Snapshot technology enables cloud customers to freeze virtual machines in specific states [2]. A frozen snapshot image may be restored by loading it to a target virtual machine, following which information about the running state of the virtual machine can be obtained. Several hypervisors, including Xen, VMWare, ESX and Hyper-V, support snapshot features.

In order to reduce the time and effort involved in forensic investigations, researchers have proposed the use of rules to automate evidence correlation and attack reconstruction [10, 18]. Liu et al. [10] have integrated a Prolog rule-based tool with a vulnerability database and an anti-forensic database to ascertain the admissibility of evidence and explain missing evidence due to the use of anti-forensic tools. However, these rule-based forensic analysis frameworks have been developed for networks, not for cloud environments.

3. Attack Reconstruction

Liu et al. [10, 11] have described an application of the MulVAL logic-based network security analyzer [13] that uses rules representing generic attack techniques to ascertain the causality between different items of evidence collected from a compromised network to reconstruct the at-

tack steps. The rules, which are based on expert knowledge, are used as hypotheses by an investigator to link chains of evidence that are written in the form of Prolog predicates in order to create attack steps. Attack scenarios are reconstructed in the form of acyclic graphs as defined below [11].

Definition 1 (Logical Evidence Graph (LEG)): A logical evidence graph $LEG = (N_f, N_r, N_c, E, L, G)$ is a six-tuple where N_f, N_r and N_c are three disjoint sets of nodes in the graph (called fact, rule and consequence fact nodes, respectively), $E \subseteq ((N_f \cup N_c) \times N_r) \cup (N_r \times N_c)$ is the evidence, L is a mapping from nodes to labels and $G \subseteq N_c$ is a set of observed attack events.

Every rule node has one or more fact nodes or consequence fact nodes from prior attack steps as its parents and a consequence fact node as its only child. Node labels consist of instantiations of rules or sets of predicates specified as follows:

1. A node in N_f is an instantiation of predicates that codify system states, including access privileges, network topology and known vulnerabilities associated with host computers. The following predicates are used:
   - hasAccount(_principal, _host, _account), canAccessFile(_host, _user, _access, _path) and other predicates model access privileges.
   - attackerLocated(_host) and hacl(_src, _dst, _prot, _port) model network topology, including the attacker's location and network reachability information.
   - vulExists(_host, _vulID, _program) and vulProperty(_vulID, _range, _consequence) model node vulnerabilities.
2. A node in N_r describes a single rule of the form $p \leftarrow p_1 \wedge p_2 \cdots \wedge p_n$. The rule head p is an instantiation of a predicate from N_c, which is the child node of N_r in the logical evidence graph. The rule body comprises p_i $(i = 1..n)$, which are predicate instantiations of N_f from the current attack step and N_c from one or more prior attack steps that comprise the parent nodes of N_r.
3. A node in N_c represents the predicate that codifies the post-attack state as the consequence of an attack step. The two predicates execCode(_host, _user) and netAccess(_machine, _protocol, _port) are used to model the attacker's capability after an attack step. Valid instantiations of these predicates after an attack update valid instantiations of the three predicates listed in item 1 above.

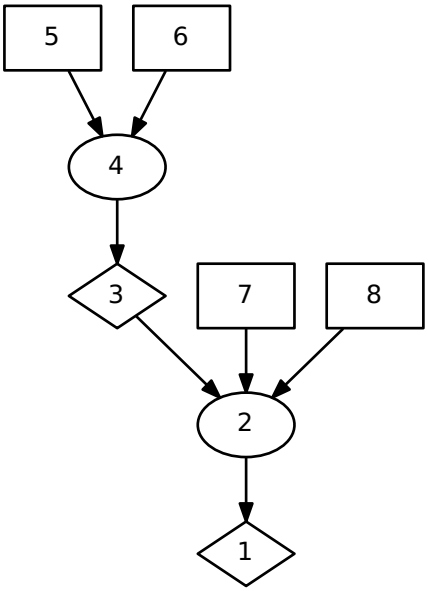

Figure 1. Example logical evidence graph.

Figure 1 shows an example logical evidence graph; Table 1 describes the nodes in Figure 1. In Figure 1, fact, rule and consequence fact nodes are represented as boxes, ellipses and diamonds, respectively. Consequence fact nodes (Nodes 1 and 3) codify the attack status obtained from event logs and other forensic tools that record the postconditions of attack steps. Fact nodes (Nodes 5, 6, 7 and 8) include network topology (Nodes 5 and 6), computer configuration (Node 7) and software vulnerabilities obtained by analyzing evidence captured by forensic tools (Node 8). Rule nodes (Nodes 2 and 4) represent rules that change the attack status using attack steps. These rules, which are based on expert knowledge, are used to link chains of evidence as consequences of attack steps. Linking a chain of evidence using a rule creates an investigator's hypothesis of an attack step given the evidence.

4. Reconstructing Attack Scenarios

This section demonstrates how three experimental attacks launched on a private cloud are reconstructed using evidence from the cloud.

4.1 Experimental Setup

OpenStack was used to create a private cloud. OpenStack is a collection of Python-based software projects that manage access to pooled

Table 1. Descriptions of the nodes in Figure 1.

Node	Notation
1	execCode(workStation1, user)
2	THROUGH 3 (remote exploit of a server program)
3	netAccess(workStation1, tcp, 4040)
4	THROUGH 8 (direct network access)
5	hacl(internet, workStation1, tcp, 4040)
6	attackerLocated(internet)
7	networkServiceInfo(workStation1, httpd, tcp, 4040, user)
8	vulExists(workStation1, 'CVE-2009-1918', httpd, remoteExploit, privEscalation)

storage and computing and network resources that reside in one or more machines corresponding to a cloud. The collection has six core projects: (i) Neutron (networking); (ii) Nova (computing); (iii) Glance (image management); (iv) Swift (object storage); (v) Cinder (block storage); and (vi) Keystone (authentication and authorization). OpenStack can be used to deploy software-as-a-service, platform-as-a-service and infrastructure-as-a-service cloud models; however, it is mostly deployed as an infrastructure-as-a-service cloud.

DevStack is a series of extensible scripts that can invoke an OpenStack environment quickly. DevStack was used to deploy a private infrastructure-as-a-service cloud with a version of Juno on an Ubuntu computer that was accessed from IP address `172.16.168.100`. An authenticated user can manage OpenStack services by entering the IP address `172.16.168.100` on a browser to access the cloud control dashboard Horizon as shown in Figure 2.

Two virtual machine instances were deployed in the private cloud, a web server named WebServer with IP address `172.16.168.226` and a file server named FileServer with IP address `172.16.168.229`. The instances were managed by an authenticated user named `admin`. WebServer was an Apache server with a MySQL database that enabled SQL queries to be issued via web applications. Also, SSH was set up on FileServer to enable authenticated users to access it remotely. The Kali ethical hacking Linux distribution tool was set up in the same network at IP address `172.16.168.173` in order to launch attacks.

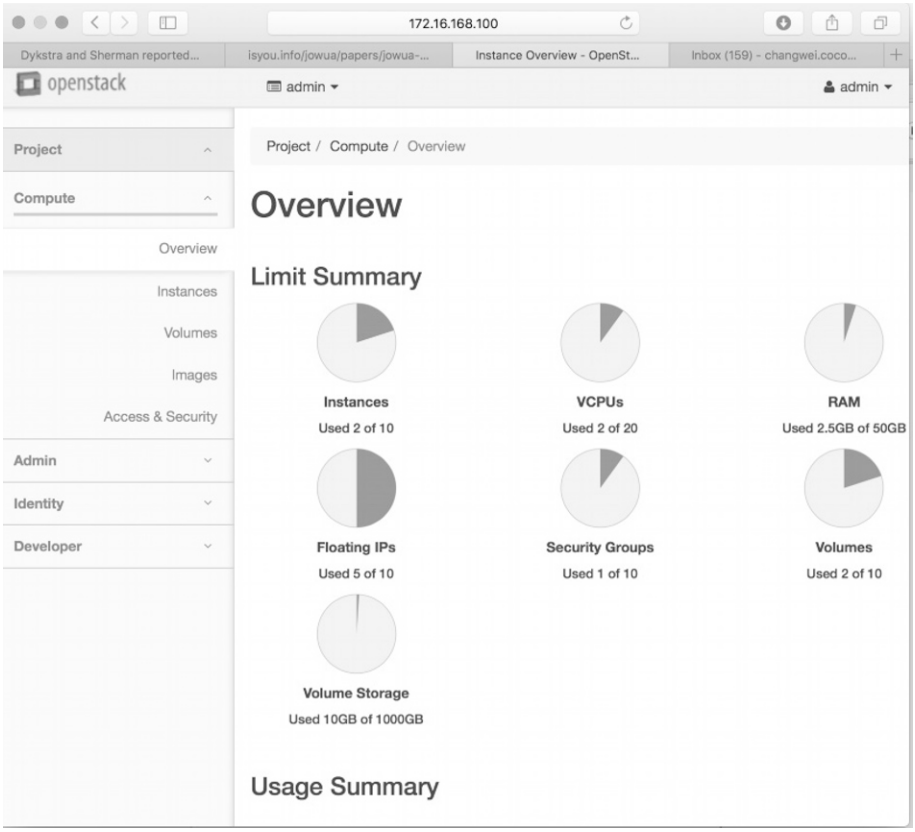

Figure 2. OpenStack web user interface (Horizon).

4.2 Experimental Attacks

A SQL injection attack, distributed denial-of-service (DDoS) attack and denial-of-service (DoS) attack were launched at the two virtual machines in the infrastructure-as-a-service cloud. The SQL injection attack exploited an unsanitized user input (CWE89 vulnerability) to the web server. The DDoS attack involved a TCP connection flood that used `nping` in Kali to prevent legitimate requests from reaching the file server. The SQL injection and DDoS attacks could target any network (including a cloud) that has the associated vulnerabilities. However, only privileged users in the infrastructure-as-a-service cloud can resize and delete a virtual machine by launching the DoS attack that exploits vulnerability CVE-2015-3241 in OpenStack Nova versions 2015.1 through 2015.1.1 and 2014.2.3 and earlier. The process of resizing and deleting an instance in this way is called instance migration. The migration process does not

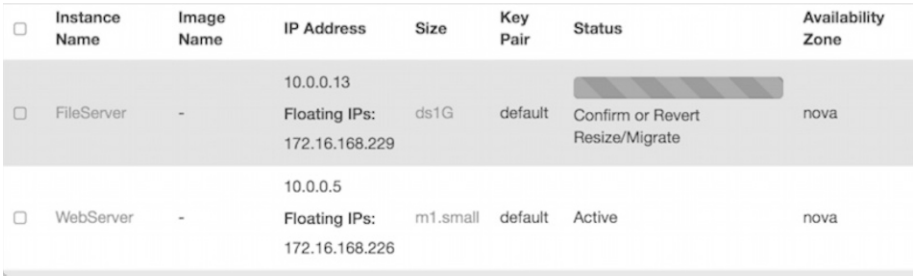

Figure 3. Resizing the file server.

terminate when an instance is deleted by exploiting CVE-2015-3241, so an authenticated user could bypass the user quota enforcement mechanism to deplete all the available disk space by repeatedly performing instance migration.

Figure 3 shows the resizing of the file server from `ds512M` to `ds1G` where the availability zone of the instances is Nova. Instances were resized and deleted until Nova was so depleted that it could not accept any new instances.

4.3 Collecting Evidence for Reconstruction

In order to obtain evidence for forensic analysis, WebServer and the SQL database in WebServer were configured to log accesses and query history. Also, Snort was installed on the virtual machines in WebServer and FileServer while Wireshark was deployed in the Ubuntu host machine to monitor network traffic. Snort was configured to capture the SQL injection attack, which generated alerts based on the pre-set rules while Wireshark was configured to capture packets associated with the DDoS and DoS attacks.

Figure 4 lists example Snort alerts and MySQL query logs for the SQL injection attack. Note that the attack was launched using `or '1'='1'` to bypass the SQL query syntax check.

Figure 5 shows a snapshot of the packets captured by Wireshark. Kali Linux at IP address `172.16.168.173` sent numerous SYN packets to FileServer at IP address `172.16.168.229` and FileServer sent numerous SYN-ACK packets back to Kali Linux.

A Prolog-based forensic tool [10, 11] was used to automate the process of correlating items of evidence to reconstruct the SQL injection and DDoS attacks. This was accomplished by coding the evidence and the cloud configuration as Prolog predicates to create the input file shown

```
[**] 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 4. Example Snort alerts and MySQL query logs.

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 Win=2920…
218	10.405682554	172.16.168.173	172.16.168.229	TCP	74	44208 → 80 [SYN] Seq=0 Win=2920…
219	10.405746104	172.16.168.173	172.16.168.229	TCP	74	38032 → 80 [SYN] Seq=0 Win=2920…
220	10.408041819	172.16.168.173	172.16.168.229	TCP	74	34348 → 80 [SYN] Seq=0 Win=2920…
221	10.408111539	172.16.168.173	172.16.168.229	TCP	74	38769 → 80 [SYN] Seq=0 Win=2920…
222	10.408205849	172.16.168.173	172.16.168.229	TCP	74	36846 → 80 [SYN] Seq=0 Win=2920…
223	10.408275950	172.16.168.173	172.16.168.229	TCP	74	35307 → 80 [SYN] Seq=0 Win=2920…
224	10.408329211	172.16.168.229	172.16.168.173	TCP	60	80 → 41930 [RST, ACK] Seq=1 Ack…
225	10.408355690	172.16.168.229	172.16.168.173	TCP	60	80 → 44471 [RST, ACK] Seq=1 Ack…
226	10.408388686	172.16.168.173	172.16.168.229	TCP	74	35276 → 80 [SYN] Seq=0 Win=2920…
227	10.408430802	172.16.168.229	172.16.168.173	TCP	60	80 → 45714 [RST, ACK] Seq=1 Ack…
228	10.408465024	172.16.168.229	172.16.168.173	TCP	60	80 → 35431 [RST, ACK] Seq=1 Ack…
229	10.408494300	172.16.168.173	172.16.168.229	TCP	74	45076 → 80 [SYN] Seq=0 Win=2920…
230	10.408557887	172.16.168.229	172.16.168.173	TCP	60	80 → 35247 [RST, ACK] Seq=1 Ack…
231	10.408583684	172.16.168.229	172.16.168.173	TCP	60	80 → 36321 [RST, ACK] Seq=1 Ack…
232	10.408616274	172.16.168.173	172.16.168.229	TCP	74	35152 → 80 [SYN] Seq=0 Win=2920…

Figure 5. Snapshot of packets captured by Wireshark.

in Figure 6. At runtime, the input file instantiated the rules to create the attack paths shown in Figure 7.

Table 2 describes the notation used in Figure 7, which shows two attack paths. The attack path on the left $[7, 8] \rightarrow 6 \rightarrow [5, 9, 10] \rightarrow 4 \rightarrow [3, 11] \rightarrow 2 \rightarrow 1$ corresponds to the SQL injection attack on the web server that exploited the CWE89 vulnerability to steal user data. The attack path on the right $[8, 16] \rightarrow 15 \rightarrow [14, 17, 18] \rightarrow 13 \rightarrow 12$ corresponds to the DDoS attack on FileServer.

However, Snort and Wireshark failed to capture the DoS attack on FileServer that exploited the CVE-2015-3241 vulnerability in the OpenStack Nova service. Fortunately, the OpenStack Nova API logs, which record information about user operations on running instances, provided evidence related to the DoS attack on FileServer.

Figure 8 shows a snapshot of the Nova API logs pertaining to the instance migration caused by the DoS attack. The commands in bold font show that instance `bd1dac18-1ce2-44b5-93ee-967fec640ff3` representing the FileServer virtual machine was resized via the commands

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

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

//Evidence found in WebServer
vulExists(webServer, 'SQLInjection', httpd).
vulProperty('SQLInjection', remoteExploit, privEscalation).
networkServiceInfo(webServer, httpd, tcp, 80, user).

//Evidence captured by Wireshark
vulExists(fileServer, 'DDoS', httpd).
vulProperty('DDoS', remoteExploit, privEscalation).
networkServiceInfo(fileServer, httpd, tcp, _, user).
```

Figure 6. Prolog predicates for the SQL injection and DDoS attacks.

`mv` (move) and `mkdir` (create new directory) issued by user `admin`. Table 3 shows that the instance ID was obtained by executing the `nova list` command on the Ubuntu host computer.

To combine the attack status and cloud system configuration, the related Nova API calls were manually aggregated and encoded as Prolog evidence predicates. This yielded the input file shown in Figure 9.

Running the Prolog-based forensic analysis tool on this input file produced the logical evidence graph shown in Figure 1, but with different node notation (shown in Table 4). The logical evidence graph shows an attack path that exploited the vulnerability CVE-2015-3241 and used the control dashboard Horizon to launch a DoS attack on the cloud.

Figure 7, which represents the SQL injection and DDoS attacks, and Figure 1, which represents the DoS attack, cannot be grouped together because the attacks originated from different locations. In addition, the DoS attack was on the Nova service instead of on a virtual machine, although it was launched from a virtual machine.

5. Using System Calls for Evidence Analysis

Because system calls enable low user-level processes to request kernel level services such as storage operations, memory and network access, and process management, they are often used for intrusion detection and

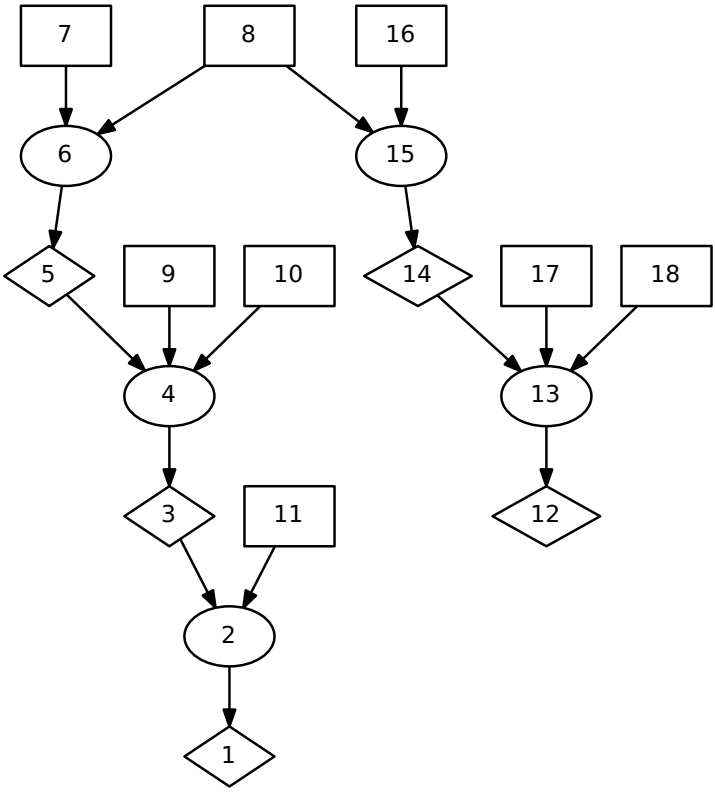

Figure 7. Attack path reconstruction for the SQL injection and DDoS attacks.

```
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/bd1dac18-1ce2-44b5-93ee-967fec640ff3 /opt/stack/data/nova/instan
ces/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/ bd1d
ac18-1ce2-44b5-93ee-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 8. Nova API call logs.

Table 2. Descriptions of the nodes in Figure 7.

Node	Notation
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)

Table 3. Virtual machine instances, names and IP addresses.

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`

forensic analysis [6]. When evidence cannot be obtained from forensic tools or system services to help recognize a known attack, system calls can be used to ascertain system behavior. Because it would be extremely rare to have an attack path in which every attack step is a zero-day

```
//Initial and final attack status
attackerLocated(controlDashboard).
attackGoal(execCode(nova, admin)).

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

//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 9. Input file for the attack using CVE-2015-3241.

Table 4. Descriptions of nodes in the DoS attack.

Node	Notation
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)

attack [17], system calls can help reconstruct the missing attack steps when other evidence is not available.

Five popular mechanisms are available to trace the system calls in a cloud-based virtual machine: (i) `ptrace` command that sets up system call interception and modification by modifying a software application; (ii) `strace` command that logs system calls and signals; (iii) auditing facilities within the kernel; (iv) system call table modification and the use of system call data writing wrappers to log the corresponding system calls; and (v) system call interception within a hypervisor [1]. Because OpenStack supports several hypervisors, including Xen, QEMU, KVM, LXC, Hyper-V and UML, no generic solution for intercepting system calls within a hypervisor exists. Hence, the `strace` command and system

```
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. SSH authentication log.

Table 5. 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

call table modification with system call data writing wrappers may be used to log relevant system calls.

An example attack launched from Kali Linux is used to demonstrate how system call sequences are used in attack reconstruction. In this attack, SSH was used to log into FileServer by supplying stolen credentials from a legitimate user named `coco`. In order to simulate the stealthy attack without triggering intrusion detection sytem alerts, the attacker was assumed to use shoulder surfing to obtain the (username, password) credentials. Figure 10 shows the SSH log from `/var/log/auth.log` in FileServer. The log entry shows that `coco` logged into FileServer from `172.16.168.173`, which actually belonged to the attacker, indicating that the attacker stole the credentials belonging to `coco`.

A process typically issues many system calls; however, only some of the calls are important for ascertaining process behavior. The important system calls [17] are listed in the second column of Table 5.

Figure 11 shows the important system calls captured from the attack. The `read` and `write` calls (in bold font) indicate that the attacker

write(9, "v", 1) = 1
read(11, "v", 16384) = 1
write(3, "\0\0\0\20\331\255\275\264c\2173)z2j\32\255n\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\343\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\20ri\321\344\220\313\322\254S\252o\201\225; 6v\243\205\10gsˆ
\253\237\325\375\332v" ..., 36) = 36
read(3, "\0\0\0\20\5\27k; \254\301\24\n\\ZN\267\260\336\323ı\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\202\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.

opened and modified a file named `test.txt`. In a `read` or `write` call, the first argument is the file descriptor where the process reads/writes data, the second is the buffer contents, the third is the number of bytes read/written by the system call; and = 1 or any number greater than 1 indicates that the system call was executed successfully.

The program behavior and the opening and modifying of a legitimate user's file were expressed in the form of the Prolog predicate: canAccessFile(fileServer, user, modify, _). This predicate states that the attacker as a legitimate user can modify the file located at _, which represents the home directory of the legitimate user. Using the evi-

```
//Initial attack status
attackerLocated(internet).
//Attacker was able to log into FileServer using stolen credentials
attackGoal(logInService(fileserver, tcp, 22).
attackGoal(principalCompromised(user)).
//Incompetent user
inCompetent(user).

//Attack status obtained by analyzing system call sequence
attackGoal(canAccessFile(fileServer, user, modify, _)).
//User could log into FileServer using the SSH protocol
networkServiceInfo(fileServer, sshd, tcp, 22, _).
//User who has the account on FileServer has file modification privileges
localFileProtection(fileServer, user, modify, _).
```

Figure 12. Input file for modifying a file with stolen credentials.

dence obtained from the log in Figure 10, which shows that the attacker with stolen credentials (expressed by the predicates: (i) attackGoal(principalCompromised(user)); (ii) inCompetent(user); and (iii) attackerLocated(internet)) logged into FileServer using SSH (expressed by the predicate attackGoal(principalCompromised(user))), and the fact that user `coco` with an account on FileServer had the privileges to modify files (expressed by the predicate localFileProtection(fileServer, user, modify, _)), the input file shown in Figure 12 was created for the Prolog-based tool.

Figure 13 shows the reconstructed attack paths and Table 6 shows the associated node notation. The attack path [3, 4, 7] $\rightarrow$ 2 $\rightarrow$ 1 has three pre-conditions, which are represented by Nodes 3, 4 and 7. Node 3 expresses the fact that files in FileServer can be modified by FileServer users. Node 4 is obtained from the fact that FileServer can be accessed using SSH via TCP on port 22. Node 7 is obtained from the SSH authentication log in Figure 10, which indicates that the user's credentials were stolen by the attacker. Note that, without the evidence obtained from the system call sequence (Node 1), the attack path [3, 4, 7] $\rightarrow$ 2 $\rightarrow$ 1 would not have been established.

The two rule nodes (Node 5 and Node 2) in Figure 13 do not have rule descriptions because of the obvious correlation between Node 6 and Node 4 (if the network provides the SSH service for logging into FileServer via TCP on port 22, then any user or attacker with stolen credentials could log into FileServer); and Nodes 3, 4 and 7 collectively and Node 1 (if a user has privileges to modify a file in FileServer, then the attacker who has stolen a user's credentials could modify the file).

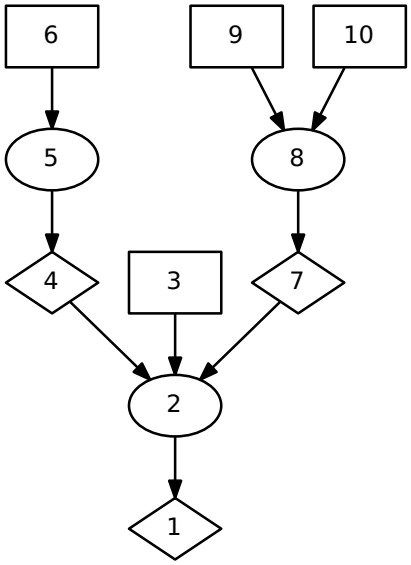

Figure 13. Attack path reconstruction using evidence obtained from system calls.

Table 6. Descriptions of the nodes in Figure 13.

Node	Notation
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	principalCompromised(user)
8	THROUGH 16 (password sniffing)
9	inCompetent(user)
10	attackerLocated(internet)

6. Conclusions

Cloud computing increases the efficiency and flexibility of enterprise operations. However, clouds present significant challenges to digital forensics. One challenge is the lack of customer control over the physical

locations of data. Other challenges include the dependence of forensically-relevant data on the cloud deployment model, large volumes of data, proprietary data formats, multiple isolated virtual machine instances running on a single physical machine, and inadequate tools for conducting cloud forensic investigations.

This research has demonstrated that evidence from multiple sources can be used to reconstruct cloud attack scenarios. The sources include intrusion detection system and application software logs, cloud service API calls and system calls from virtual machines. To acquire evidence from the sources, a forensics-enabled cloud should support: (i) logging and retrieval of intrusion detection system and software service data; (ii) secure storage and retrieval of OpenStack service API call logs, firewall logs and snapshots of running instances; and (iii) storage and retrieval of system calls, especially when the first two sources are unavailable. The Prolog-based forensic analysis presented in this chapter demonstrates the effectiveness and utility of automating the correlation of evidence from multiple sources to reconstruct attack scenarios in digital forensic investigations.

Future research will implement extensions to the forensics-enabled cloud to preserve data integrity, reduce data volume and manage the diversity of digital forensic data stored in the cloud.

References

[1] F. Beck and O. Festor, Syscall Interception in Xen Hypervisor, Technical Report no. 9999, INRIA Nancy – Grand Est, Villers-les-Nancy, France, 2009.

[2] D. Birk and C. Wegener, Technical issues of forensic investigations in cloud computing environments, *Proceedings of the Sixth International Workshop on Systematic Approaches to Digital Forensic Engineering*, 2011.

[3] J. Dykstra and A. Sherman, Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust and techniques, *Digital Investigation*, vol. 9(S), pp. S90–S98, 2012.

[4] J. Dykstra and A. Sherman, Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform, *Digital Investigation*, vol. 10(S), pp. S87–S95, 2013.

[5] B. Hay and K. Nance, Forensic examination of volatile system data using virtual introspection, *ACM SIGOPS Operating Systems Review*, vol. 42(3), pp. 74–82, 2008.

[6] S. Hofmeyr, S. Forrest and A. Somayaji, Intrusion detection using sequences of system calls, *Journal of Computer Security*, vol. 6(3), pp. 151–180, 1998.

[7] M. Hogan, F. Liu, A. Sokol and J. Tong, NIST Cloud Computing Standards Roadmap, NIST Special Publication 500-291, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[8] A. Jaquith, *Security Metrics: Replacing Fear, Uncertainty and Doubt*, Pearson Education, Boston, Massachusetts, 2007.

[9] K. Kent, S. Chevalier, T. Grance and H. Dang, Guide to Integrating Forensic Techniques into Incident Response, NIST Special Publication 800-86, National Institute of Standards and Technology, Gaithersburg, Maryland, 2006.

[10] C. Liu, A. Singhal and D. Wijesekera, A logic-based network forensic model for evidence analysis, in *Advances in Digital Forensics XI*, G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 129–145, 2015.

[11] C. Liu, A. Singhal and D. Wijesekara, A probabilistic network forensic model for evidence analysis, in *Advances in Digital Forensics XII*, G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 189–210, 2016.

[12] P. Mell and T. Grance, NIST Definition of Cloud Computing, NIST Special Publication 800-145, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[13] X. Ou, S. Govindavajhala and A. Appel, MulVAL: A logic-based network security analyzer, *Proceedings of the Fourteenth USENIX Security Symposium*, 2005.

[14] G. Palmer, A Road Map for Digital Forensic Research, DFRWS Technical Report, DTR-T001-01 Final, Air Force Research Laboratory, Rome, New York, 2001.

[15] A. Pichan, M. Lazarescu and S. Soh, Cloud forensics: Technical challenges, solutions and comparative analysis, *Digital Investigation*, vol. 13, pp. 38–57, 2015.

[16] K. Ruan, J. Carthy, T. Kechadi and M. Crosbie, Cloud forensics, in *Advances in Digital Forensics V*, G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 35–46, 2011.

[17] X. Sun, J. Dai, P. Liu, A. Singhal and J. Yen, Towards probabilistic identification of zero-day attack paths, *Proceedings of the IEEE Conference on Communications and Network Security*, pp. 64–72, 2016.

[18] W. Wang and T. Daniels, A graph based approach toward network forensic analysis, *ACM Transactions on Information and Systems Security*, vol. 12(1), article no. 4, 2008.

[19] S. Zawoad and R. Hasan, A trustworthy cloud forensics environment, in *Advances in Digital Forensics XI*, G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 271–285, 2015.