Digital Investigation 10 (2013) S87–S95

Contents lists available at SciVerse ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform

Josiah Dykstra*, Alan T. Sherman

Cyber Defense Lab, Department of CSEE, University of Maryland, Baltimore County (UMBC), 1000 Hilltop Circle, Baltimore, MD 21250, United States

Keywords:
OpenStack
Cloud computing
Digital forensics
Cloud forensics
FROST

A B S T R A C T

We describe the design, implementation, and evaluation of FROST—three new forensic tools for the OpenStack cloud platform. Our implementation for the OpenStack cloud platform supports an Infrastructure-as-a-Service (IaaS) cloud and provides trustworthy forensic acquisition of virtual disks, API logs, and guest firewall logs. Unlike traditional acquisition tools, FROST works at the cloud management plane rather than interacting with the operating system inside the guest virtual machines, thereby requiring no trust in the guest machine. We assume trust in the cloud provider, but FROST overcomes non-trivial challenges of remote evidence integrity by storing log data in hash trees and returning evidence with cryptographic hashes. Our tools are user-driven, allowing customers, forensic examiners, and law enforcement to conduct investigations without necessitating interaction with the cloud provider. We demonstrate how FROST's new features enable forensic investigators to obtain forensically-sound data from OpenStack clouds independent of provider interaction. Our preliminary evaluation indicates the ability of our approach to scale in a dynamic cloud environment. The design supports an extensible set of forensic objectives, including the future addition of other data preservation, discovery, real-time monitoring, metrics, auditing, and acquisition capabilities.

1. Introduction

Today, cloud computing environments lack trustworthy capabilities for the cloud customer or forensic investigator to perform incident response and forensic investigation. Consequently, customers of public cloud services are at the mercy of their cloud provider to assist in an investigation. Law enforcement relies on the cumbersome and time-consuming search warrant process to obtain cloud data, and requires the cloud provider to execute each search on behalf of the requester. In 2012, we concluded that the management plane is an attractive solution for user-driven forensic capabilities since it provides access to forensic data without needing to trust the guest virtual machine (VM) or the hypervisor and without needing assistance from the cloud provider. Storing and acquiring trustworthy evidence from a third party provider is non-trivial. This paper describes our design and implementation of a management plane forensic toolkit in a private instantiation of the OpenStack cloud platform, which we call Forensic OpenStack Tools (FROST).

FROST provides the first forensic capabilities integrated with OpenStack, and to our knowledge the first to be built into any Infrastructure-as-a-Service (IaaS) cloud platform. Throughout the paper we use the NIST definition of *cloud computing* as a model for on-demand access to a pool of resources "that can be rapidly provisioned and released with minimal management effort or service provider interaction" (National Institute of Standards and Technology, 2011). Our forensic extensions allow for efficient, trustworthy, and user-

* Corresponding author.
E-mail addresses: dykstra@umbc.edu (J. Dykstra), sherman@umbc.edu (A.T. Sherman).

1742-2876/$ – see front matter
http://dx.doi.org/10.1016/j.diin.2013.06.010

driven incident response and forensic acquisition in a cloud environment, which match the cloud characteristic of being user-driven.

This work implements practical tools on the theoretical foundations we established (Dykstra and Sherman, 2012). FROST collects data at the cloud provider, from the host operating system level (outside the guest virtual machines), and makes that data available within the *management plane*. The management plane, exposed through a website and application programming interface (API), is how users of OpenStack control the cloud and where they start and stop virtual machines. Because the user collecting forensic data does not communicate with a virtual machine, the forensic data are preserved against compromised or untrustworthy virtual machines.

Consider a cloud customer, Alice, whose provider uses OpenStack with FROST. Alice wants to investigate an incident of suspiciously-high bandwidth usage from her cloud-hosted webserver. While her webserver logs web requests inside of its VM, Alice can get a more complete pictures of activity by obtaining a record of management activity and metadata about her VM. Alice uses FROST to retrieve firewall logs, Nova Compute Service API logs, and a virtual hard drive image of the suspicious machine and then provides this evidence to the authorities. The firewall logs may show an attacker scanning Alice's virtual machine before hacking it. API logs may contain evidence of unauthorized attempts to stop the virtual machine. The disk image may contain evidence of what an attacker did once he obtained access. This is strong forensic evidence about the potential crime that can be used in court. Alice can obtain this evidence using either the web management plane or the OpenStack API. FROST ensures the forensic integrity of the evidence that Alice gathers. Without FROST, this evidence would only be available with assistance from Alice's cloud provider.

OpenStack (2012b) is an open-source cloud computing platform, conceived as a joint project between the National Aeronautics and Space Administration (NASA) and Rackspace. OpenStack users include many large organizations such as Intel, Argonne National Laboratory, AT&T, Rackspace, and Deutsche Telekom. The cloud platform comprises six primary modular components: Nova, the compute platform and cloud controller; Swift, the object storage system; Glance, the service for managing disk images; Keystone, the identity service; Horizon, the web-based dashboard for managing OpenStack services; and Quantum, network services for virtual devices. OpenStack is a complex software package, with over 600,000 lines of code and 415 active developers (OpenStack, 2012a). It is a widely used platform for private cloud instances, but it is also compatible with commercial cloud offerings. OpenStack has APIs compatible with Amazon EC2 and S3.

Without loss of generality, our approach makes the following assumptions. First, the user-driven forensic capabilities are applicable in situations where a cooperative cloud customer is involved in the investigation. That is, if a malicious customer uses the cloud to commit a crime, the cloud provider will still be required to assist law enforcement in the investigation. Second, the proposed solution assumes a trusted cloud provider and cloud infrastructure. Evidence from our forensic tools could be manipulated unless the underlying layers of the cloud infrastructure, such as the host operating system and hardware, have integrity. We assume that the hardware, host operating system, hypervisor, and cloud employees are trusted, but we do not assume trust in the guest machine. Third, we do not consider legal issues associated with the process or product of cloud-based forensic data acquisition; Dykstra and Riehl (2012) previously explored those issues.

Our contributions are:

- Description of the architecture, design goals, and implementation of user-driven forensic acquisition of virtual disks, API logs, and firewall logs from the management plane of OpenStack.
- An algorithm for storing and retrieving log data with integrity in a hash tree that logically segregates the data of each cloud user in his or her own subtree.
- Preliminary informal Evaluation results showing that the proposed solution satisfies technological and legal requirements for acceptance in court and scales appropriately for a cloud environment.

The rest of the paper is organized as follows. Section 2 reviews previous and related work. Section 3 describes the requirements, specifications, and capabilities of FROST. Section 4 explains the architecture of our solution. Section 5 discusses the design. Section 6 explains our API and management console implementations based on the architecture. Section 7 evaluates our solution. Section 8 discusses advantages, limitations, and trust assumptions. Section 9 concludes the work.

2. Previous and related work

We survey previous and related work in remote forensic acquisition, forensic data collected by providers, and methods for storing content on untrusted platforms.

Data acquisition is a key issue when investigating cloud-based incidents (Dykstra and Sherman, 2011a,b; Ruan et al., 2011; Taylor et al., 2011). Research to date has focused on explaining this issue but has failed to produce practical tools to support remote forensic acquisition. Dykstra and Sherman (2012) illustrated how to use existing tools like Guidance EnCase to acquire forensic data remotely over the Internet, but explained why the data may be untrustworthy. Martini and Choo (2012) proposed a conceptual framework for preservation and collection of forensic data from cloud computing but did not implement any capabilities. In some cases, such as in Amazon Web Services, it is possible to retrieve an image of the virtual disk of a virtual machine. There is, however, no mechanism to obtain a hash of the image on the provider's system to validate the integrity of the image after download.

In addition to disk images, forensic investigators use metadata and system logs to reconstruct an event. Generating metadata and system logs are usually standard practices in the operation of a system, rather than forensic-specific tasks. Nevertheless, the logs are useful in an investigation and easily gathered. Consumers of cloud

services have few tools available for accessing low-level logs to the cloud infrastructure. Cloud providers and researchers encourage application-level logging (Marty, 2011); Google, Amazon, and Microsoft allow customers to log accesses to stored objects (Google, 2012; Amazon Web Services, 2011; Microsoft, 2012). Cloud customers are usually responsible for their own monitoring, metrics, and auditing inside the customer's VM. Amazon CloudWatch is a monitoring service for EC2, but its metrics are very coarse (Amazon Web Services, 2013); detailed monitoring for network utilization reports the number of bytes sent or received on all network interfaces by an instance at one-minute frequency. To our knowledge, no cloud provider makes available customer accessible API call audit logs or VM firewall logs. That is, a customer has no way to know if, when, and from what IP address his or her credentials were used to make API calls.

Data integrity is a critical component of the forensic process. Given the popular discussions about data security in the cloud, researchers and commercial vendors have been dedicated to addressing data privacy. Other authors have developed proposals for ensuring integrity on untrusted machines, such as third-party servers. Clarke (2005) proposed a method for validating the integrity of untrusted data using hash trees and a small fixed-sized trusted state. This method differs from our method because it does not check the integrity of subsets of the data. SUNDR (Secure untrusted data repository) (Li et al., 2004) is a filesystem for storing data securely on untrusted servers. However, SUNDR requires that each filesystem user is able to see file modifications by all other clients. In our solution, we want each cloud consumer to be independent from the other customers.

Other research has focused on storing content securely on untrusted servers, which could then produce trustworthy forensic data, even from third-party cloud providers. Haber et al. (2008) explored in depth the redaction of subdocuments from signed original data, while preserving the cryptographic link of integrity between the two datasets. Haber posited that audit logs can be considered an append-only database, and that an audit report is essentially a database query with certain entries redacted. Haber's proposed redactable signature algorithm is precisely applicable to the cloud logs we will encounter, though it must take into account a constantly changing dataset.

The dissertations of Crosby (2009) and Kundu (2010) bear striking similarity to our goals despite different motivations. Crosby proposed history tree tamper-evident logs, and suggested that they could "increase the trust in software service and 'cloud computing.'" Kundu was interested in authenticating subsets of signed data objects without leaking structural information about the data structures. Our work was influenced by these designs. We assume that the logger is trusted, and we use our enhanced logging mechanism simply for efficient log storage, retrieval, and integrity validation.

3. Requirements, specifications, and capabilities

We describe the requirements, specifications, and capabilities for FROST. We identify the stakeholders and use cases that will help determine the tool requirements. We also discuss the accepted legal and forensic community requirements, and how we will meet them.

Cloud-based crimes take two general forms that determine the stakeholders who would use FROST. One form is a crime committed against the cloud-based resources of an innocent victim who is cooperative in an investigation. The other is a crime committed by an uncooperative party using the cloud as an instrument of a crime. In the first case, the legitimate cloud customer and/or law enforcement will use FROST. In the second case, law enforcement or the provider will use FROST. In both cases the requirement is to minimize interaction with personnel at the cloud provider. The cloud provider deploys FROST, but has no other responsibilities (subject to the assumptions in Section 1).

3.1. Scientific, technical, and legal requirements

There is no single, authoritative source for requirements development of new forensic tools. Our solution, however, is informed by accepted practices and written guidance. The Scientific Working Group on Digital Evidence (SWGDE) (2006) asserts that "Digital Evidence submitted for examination should be maintained in such a way that the integrity of the data is preserved. The commonly accepted method to achieve this is to use a hashing function." On the requirements for acquisition the National Institute for Standards and Technology (2004) says "The two critical measurable attributes of the acquisition process are completeness and accuracy. Completeness measures if the all the data was acquired, and accuracy measures if the data was correctly acquired." Integrity and completeness of the data will be of foremost importance.

The cloud environment dictates the technical requirements. Any digital forensic tools for cloud computing should be compatible with cloud characteristics of on-demand self-service, rapid elasticity, and scalability. The following technical requirements are consistent with these characteristics:

1. **Be compatible with existing forensic formats.** Instead of creating new data formats, the new capabilities output data in existing formats to be easily ingested by other forensic tools. Our logs and disk images are provided in standard formats, and all are accompanied by a Digital Forensic XML (DFXML) file (Garfinkel, 2012). DFXML is used to express the cryptographic hashes and provenance information.
2. **Be easy to generate.** It must be easy to modify existing cloud deployments to add forensic capabilities. It must also be intuitive and simple for a user to request forensic data. Our changes to a stock installation of OpenStack can be made by running an installation script. Users can request forensic data with a single command or web click.
3. **Be open and extensible.** The implementation must be available for any OpenStack administrator. Developers should be able to extend and contribute new forensic capabilities. The platform we developed allows other developers to integrate other forensic tools quickly and easily. The software will be submitted to the OpenStack project.

4. **Be scalable.** The forensic tools must be usable for single cloud instances, while also supporting millions of cloud customers and virtual machines. FROST can support any number of instances and is limited only by the processing time it takes the host operating system to retrieve the forensic data.
5. **Follow existing practices and standards.** Where possible, cloud forensic tools should follow standard forensic practices. The forensic data we provide adheres to accepted practices and can be ingested by standard forensic tools such as Guidance EnCase.

For acceptance in court, the Department of Justice's "Search and Seizure Manual" (2009) applies Federal Rules of Evidence 901(b)(9), explaining that "to demonstrate authenticity for computer-generated records, or any records generated by a process, the proponent should introduce '[e]vidence describing a process or a system used to produce a result and showing that the process or system produces an accurate result.'" In most cases, the reliability of a computer program can be established by showing that users of the program actually do rely on it on a regular basis, such as in the ordinary course of business. Our solutions use ordinary data, such as firewall logs, even when we have enhanced the storage of data to add increased data security.

3.2. Specifications and capabilities

FROST has three primary components. First, a cloud user can retrieve an image of the virtual disks associated with any of the user's virtual machines, and validate the integrity of those images with cryptographic checksums. Second, a cloud user can retrieve logs of all API requests made to the cloud provider made using his or her credentials, and validate the integrity of those logs. The API is used for administering virtual machines, such as creating and starting VM instances. Third, the cloud user can retrieve the OpenStack firewall logs for any of the user's virtual machines, and validate the integrity of those logs. The OpenStack firewall operates at the host operating system, and the API is used to administer it, such as allowing or blocking network ports. These three components are useful and offer forensic data that are not available directly to cloud users today. In our informal discussions with cloud users and administrators of two large private clouds and forensic experts, they all requested capabilities that were consistent with these features.

Cloud users interact with their provider and manage cloud resources through the management plane using a web interface and API. FROST is accessible from each of these management plane interfaces. The implementation is modular to allow additional forensic capabilities to be added later.

4. Architecture

We describe the architecture of our solution. We show how we integrate with OpenStack, the type and format of the data we collect, and the methods for returning data to the requestor.

4.1. Integration with OpenStack

OpenStack has many components, but we focus on the two where we have integrated FROST: Nova and Horizon. Nova provides the compute service through virtual servers similar to those in Amazon EC2 and implements the compute API. Horizon provides the web-based user interface for OpenStack, and communicates with Nova through the

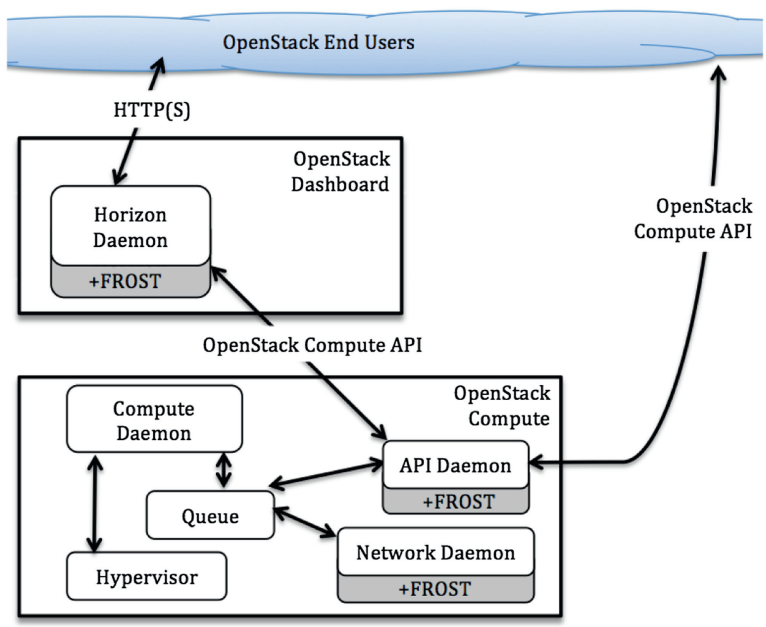

Fig. 1. Pictorial snippet of the OpenStack architecture showing where OpenStack Compute (Nova) and OpenStack Dashboard (Horizon) have been modified to add FROST. Horizon provides a web interface to the management plane and Nova provides an API interface to the management plane. The majority of changes for FROST were to the API Daemon.

compute API. Fig. 1 highlights where we modified Nova and Horizon to integrate FROST.

We add new Nova API calls that correspond to our forensic features. Cloud users who interact with OpenStack using the compute API are able to exercise our capabilities from command-line tools and in their own programs.

Horizon is built using Django and Python, and implements dashboards for OpenStack. We modify the specification for the dashboard that displays instance information and creates a new tab. This tab has links to our forensic capabilities. These links return data from their corresponding API calls.

OpenStack has a variety of credentials for different purposes. Our tools assume that OpenStack has authenticated the user making the request. The Horizon web interface requires only a username and password. The command-line API requires either an access key and secret access key (which can be retrieved using the API), or an X.509 certificate and private key. API requests are digitally signed using the private key, and this signature is transmitted to OpenStack along with the certificate. Nova also has a root certificate that can sign documents. We use this root certificate to add integrity to the storage of log data, which we call the Authenticated Logging Service (ALS).

4.2. Data retrieval

Each of the three FROST capabilities accesses unique data that are already stored by OpenStack or which we can easily enable for storage. Retrieval of data for the user depends on how and where the data are stored.

Retrieval of virtual disks is the most straightforward task. For each virtual machine, OpenStack creates a directory on the host operating system that contains the virtual disk, ramdisk, and other host-specific files. The file format of the virtual disk varies according to the hypervisor used. Since we use KVM as our hypervisor, the format of our virtual disks is QEMU QCOW2 images. The ability to retrieve the original virtual disks must support snapshots of disks from machines that are running, as well as downloads of disk images from stopped machines. QEMU provides utilities to convert QCOW2 images to raw format, and libewf can convert raw images to the EWF-E01 format.

Cloud users may run a firewall inside their VM, but OpenStack provides firewall services beneath the VM. By default OpenStack uses the Linux iptables firewall on the host machine to implement network security for the guest machines. A new chain, or group of rules, is created for each instance. Several default rules are automatically created, such as allowing the host to communicate with the guest. Cloud users are then able to create custom rules manually, such as allowing inbound SSH or HTTP traffic. OpenStack has no inherent configuration options to log network connections that match the firewall rules or connections that are denied by the firewall. However, iptables natively has this ability. We enable logging on all denied network connections and enable the user to retrieve logs for her OpenStack instances.

OpenStack has the ability to log request successes and failures when a user issues a request to Nova. For example, when a user uses the API to request a new VM, this request can be recorded. These logs are stored on the host operating system, and therefore are typically not available to cloud users. FROST stores these data, but in a method that allows the data to be segregated for each user and that includes integrity checking information.

5. Design

The goals of enhanced API and firewall logging are to enable a cloud user to retrieve and validate the integrity of forensically-relevant log data. The Authenticated Logging Service supplements Nova's default logging capability. This service stores the same data as the traditional log, but a new hash tree segregates users' data and integrity checking information with minimal overhead for record storage or retrieval. Each OpenStack user account has his or her own subtree under the root.

When a user provisions a new virtual machine in OpenStack, a universally unique identifier (UUID) is assigned to the machine. These UUIDs become children of the owner's root, and logs for that machine are appended as follows. The subtree of any virtual machine has a depth of four for the year, month, and day of the log entry, with the log messages as leaves of the tree. Because the tree is constantly changing as new log entries are added, hash values for the intermediate hash tree nodes are recalculated daily. This structure enables a user to request any date range for any or all virtual machines, while reducing the additional overhead required.

The Authenticated Logging Services guarantees integrity of the log data using cryptographic hashes. Integrity checking allows the user to validate if data have been inserted, removed, or modified. For example, if Alice requests her logs for December, she can calculate the hash values that she expects in the tree and compare them to what the provider claimed they should be. If an attacker modified the log data in transit, the integrity check would fail and alert Alice to errors or manipulation.

6. Implementation

We provide details about the implementation of FROST and show how users interact with the tools.

We implemented the forensic extensions using DevStack, an OpenStack development environment, on Ubuntu 12.04. We used OpenStack Folsom, which was released September 27, 2012. We used the Xen hypervisor and Ubuntu guests, but our implementation can support any hypervisor and guest operating systems.

6.1. Authenticated logging service

The Authenticated Logging Service uses Merkle trees (Merkle, 1988) as the data structure for storing API and firewall log data. Unlike previous work, we are not concerned with hiding the structural information associated with the tree, nor about prohibiting redaction in exported subtrees.

Hash trees offer three advantages. First, storing summary information about a larger dataset enables efficient validation and minimal data transmission. For any subset of data in the tree, the algorithm hashes chunks of the data, and uses those hashes to compute the hash of the whole

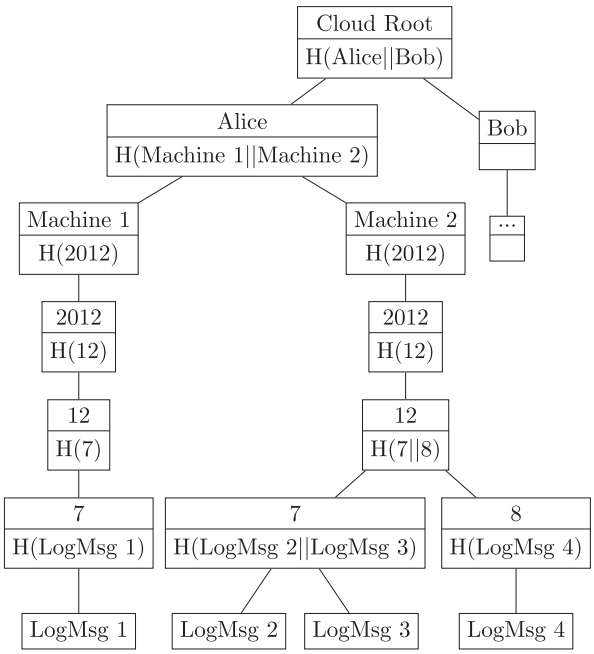

Fig. 2. Tree structure used to store API logs by user, machine, year, month, and day, showing log entries for Alice's two virtual machines on December 7–8, 2012. The value at each branch node is a hash of the concatenation of the values of its children. These hash values enable integrity validation for any subtree of the whole.

tree. It is unnecessary to reveal or transmit the entire tree. Second, given the way we organize the tree, a user can easily query for data over any date range. Third, the hash tree natively enables a user to validate the integrity of a subset of log data.

Our algorithm for storing API and firewall logs is as follows. These two sets of data are stored separately. Since the design is the same for each, we describe only the storage of API logs. As shown in Fig. 2, the cloud provider maintains a single, append-only hash tree for all users. When a new user joins the cloud service, a subtree is created for the user under the root. The user's tree root is signed using the user's public key. All API logs associated with that user are stored in his or her subtree. Data under the user's root are organized in five layers, corresponding to the machine instance, year, month, and day of the respective log entry. Raw records are found at the leaves, stored as children of the day. The value at each branch node is calculated by concatenating the values of its children and computing the hash of that aggregate. Every minute, the provider computes a hash of the children at each node and updates the value of each node with a new hash. The provider also signs the root of the tree, and the root of each cloud customer, using the Nova root certificate.

When a user wishes to retrieve the logs associated with a particular instance, the cloud provider returns the raw log messages and any hash values necessary to validate the integrity of the result up to the user's root. For example, in the most trivial case shown in Fig. 2, the provider would return only a single log message and the hash value at node "Alice." Using the Nova root certificate, the provider also hashes and signs all data being returned and records these values in a DFXML log file which is returned to the user. Alice could then compute the hashes and validate that the value she calculated for "Alice" matches what the provider claimed.

6.2. API implementation

Many users interact with cloud platforms with command line tools that call API functions. The Nova API daemon is the endpoint for API queries. Our API extension file contains code to implement our features. We register these extensions with Nova, and add the ability to call them from the dashboard and the command-line `novaclient`. New API calls are added to OpenStack by placing their functionality in a contribution directory, and modifying `novaclient` to allow the user to call the API. Each of our forensic capabilities is implemented in this manner. We then hook the Nova logging handler to send log messages to our replacement logging service, described below. We also hook the iptables manager to label firewall messages with the instance ID associated with them. The Nova Network daemon then carries out the work of correctly modifying the iptables rules as the system and the user create them.

To use FROST a user must have already authenticated to OpenStack with his or her private key or credentials. The authenticated user can access only the logs for machines that he or she owns, as enforced by Keystone, the OpenStack identity service. The API validates that the requestor has permission to access the instance for which he or she is requesting forensic data.

Nova logs are stored in/var/log/nova/on the host operating system. When a user requests his or her Nova logs, FROST searches this file for lines that contain that user's personal identifier.

Listing 1 shows the output of using FROST from the command line to retrieve the Nova logs for a single virtual machine. FROST returns the Nova entries that match that UUID, and also creates a DFXML file named report.xml. The DFXML file contains provenance information about the execution of FROST and a hash of the log data for integrity validation.

```
$ nova get-nova-logs 0afcfbcd-b836-4593-a02c-25d8d3a94b00 verify.xml
[truncated]
2012-12-01 13:30:49 INFO nova.api.openstack.wsgi [req-0afcfbcd-b836-4593-a02c-25d8d3a94b00 admin demo]
    POST http://10.34.50.142:8774/v2/5ee3040fa890428387f56111576cf819/servers
2012-12-01 13:30:49 DEBUG nova.quota [req-0afcfbcd-b836-4593-a02c-25d8d3a94b00 admin demo] Created
    reservations ['915e9c89-b3bc-4091-8b75-3b555961ec3e', '72c39d24-0a96-42ca-96f1-593da3aa9f81',
    '57843316-872b-4b40-a853-2aa7c730262e'] from (pid=16036) reserve /opt/stack/nova/nova/quota.py:697
2012-12-01 13:30:50 DEBUG nova.compute.api [req-0afcfbcd-b836-4593-a02c-25d8d3a94b00 admin demo] Going to
     run 1 instances... from (pid=16036) _create_instance /opt/stack/nova/nova/compute/api.py:492
[truncated]
```

Listing 1. Execution of the FROST API to retrieve the Nova logs for virtual machine 0afcfbcd-b836-4593-a02c-25d8d3a94b00 showing user "admin" provisioning a new virtual machine. These data are available only to users with FROST or with provider assistance.

Firewall logging must be enabled, since it is not enabled by default in OpenStack. Because OpenStack creates default rules for each running virtual machine, we append another rule that logs all dropped packets to/var/log/syslog. For each instance, we prepend a special prefix to the log messages that labels the UUID of the machine. Doing so enables us to parse the log file and identify those lines that correspond to the particular virtual machine that the user requests.

Listing 2 shows the output of using FROST from the command line to retrieve the firewall logs for a single virtual machine. FROST returns the firewall logs that match that UUID, and also creates a DFXML file named report.xml.

```
$ nova get-firewall-logs 0a18799f-c198-4dbb-b369-b49184e3dfbc verify.xml
0a18799f-c198-4dbb-b369-b49184e3dfbc: Nov 28 11:13:38 domU-12-31-39-17-29-5D kernel: [  310.765760]
    IPTables-Dropped: IN=eth0 OUT= MAC=12:31:39:17:29:5d:fe:ff:ff:ff:ff:ff:08:00 SRC=130.85.36.72 DST
    =10.97.42.171 LEN=52 TOS=0x00 PREC=0x00 TTL=48 ID=29222 DF PROTO=TCP SPT=55739 DPT=443 WINDOW=1002
    RES=0x00 ACK URGP=0
0a18799f-c198-4dbb-b369-b49184e3dfbc: Nov 28 11:13:36 domU-12-31-39-17-29-5D kernel: [  309.623023]
    IPTables-Dropped: IN=eth0 OUT= MAC=12:31:39:17:29:5d:fe:ff:ff:ff:ff:ff:08:00 SRC=172.16.0.23 DST
    =10.97.42.171 LEN=103 TOS=0x00 PREC=0x00 TTL=64 ID=42188 PROTO=UDP SPT=33905 DPT=53 LEN=83
[truncated]
```

Listing 2. Execution of the FROST API to retrieve the firewall logs of virtual machine 0a 18799f-c198-4dbb-b369-b49184e3dfbc showing traffic to ports 443 and 53 being dropped. This level of logging is exposed only to users with FROST or with provider assistance.

Disk images are stored in the filesystem of the host operating system. The file path includes the name of the instance, which is used to identify the correct image to return to the user.

Our implementation supports the retrieval of disk images from virtual machines that are powered off. New versions of QEMU and Libvirt include functionality to create snapshots of running instances, but these features have not yet been added to OpenStack.

Listing 3 shows the output of using FROST from the command line to retrieve a disk image for a single virtual disk with volume name myvol-e9a5612d. FROST returns the disk image for myvol-e9a5612d, and also creates a DFXML file named report.xml in the same way as above. The requestor can validate the integrity of the image by comparing the hash value in the DFXML, as computed by the cloud provider, with the hash value computed by the requestor.

```
$ nova get-disk myvol-e9a5612d report.xml
MD5:  b17ee04095b2a3d81eed98628072eab6
SHA1: 399f5ffaccd09fe43d642d5f0d996875ca650c9f

$ sha1sum myvol-e9a5612d
399f5ffaccd09fe43d642d5f0d996875ca650c9f myvol-e9a5612d
```

Listing 3. Execution of the FROST API to retrieve a disk image of volume myvol-e9a5612d. The user can easily validate the integrity by comparing the checksums.

6.3. Management console web implementation

The Management Console for OpenStack Compute contains an Instance Detail page for each virtual machine guest created by the user. We added a new tab for "Incident Response" to the Instance Detail section. This tab contains our forensic tools, and provides a space for future forensics and incident response related features.

Fig. 3 shows the Incident Response page for a virtual machine. On this page a user can click to retrieve Nova logs, firewall logs, and a disk image. These links return a zip file that contains the data requested and a DFXML file.

7. Preliminary evaluation

We conducted two evaluations of FROST. The first is an objectives-based assessment to validate that FROST can scale and produce correct results. The second is a consumer-oriented demonstration and independent appraisal to gather feedback from potential users.

We tested FROST by creating 100 fictitious users and used the API to launch five virtual machines for each user simultaneously. For each virtual machine, we associated firewall rules that allowed only SSH. With 500 virtual machines running, we used a network scanner to scan ports 1–1024 on each machine. This was done to trigger the firewall to block network traffic on the prohibited ports. We then chose a random user's key from the list of 100 users, and a random instance from the list of 500, and used the API to try and stop the virtual machine. There was only a 1% chance that the chosen user owned the chosen virtual machine, and this procedure generated Nova logs for both successful and unsuccessful attempts.

We then chose 20 users at random and for each user requested the API logs, firewall logs, and disk image for each of the user's instances. We validated the integrity of each log and disk image returned by computing the hash of the data and comparing it to the hash value in the DFXML file. No anomalies were observed.

To scale to more users the logging mechanism needs only more storage space. Each API and firewall log entry can be no larger than 1 KB. Using SHA-1 as the cryptographic hash algorithm requires 160 bits for each tree node (user, VM, year, month, day). In the worst case this creates 1664 bytes per entry. Therefore, the logging mechanism can store more than 645,000 log entries in 1 GB of storage. We believe that modern servers can easily handle this load. Cloud providers could choose to share this cost with customers who wish to enable the logging service.

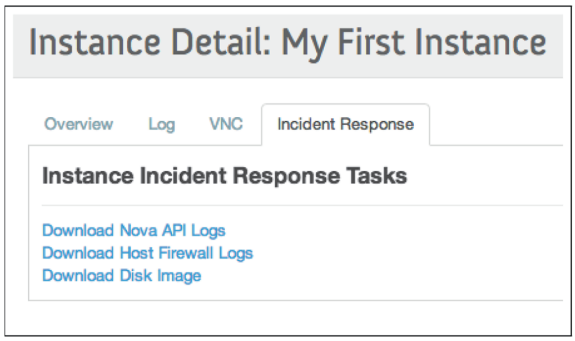

Fig. 3. Screenshot of the OpenStack web interface showing our new incident response tab and links to FROST functions to download Nova logs, firewall logs, and disk images for one virtual machine. These links provide easy access to forensic functions for cloud users.

Cloud providers can expect minimal performance impact after deploying FROST. The overhead of calculating checksums and providing them to users is negligible. The time and bandwidth required for a user to download his or her logs or disk images is dependent upon the size of the data. We also expect users to request large data volumes, such as disk images, infrequently.

We demonstrated FROST to 12 users and administrators of a large private government cloud; their reactions were positive. One administrator said "[FROST] is exactly what OpenStack has been missing" and "I appreciate shifting the load [of investigation] away from me and onto our users." The audience was confident that FROST would be useful in incident response and forensics due to its ease of use. Users exercised FROST's web and API interfaces and described them as "intuitive and consistent with Open-Stacks design." Most users anticipated automating their use of FROST, such as for collecting logs on a daily basis. They were also interested in using FROST for non-forensic purposes, such as troubleshooting and compliance. The administrators plan to deploy FROST to this cloud in mid-2013.

This evaluation shows that the integrity, completeness, and accuracy of the forensic data are intact, as identified by SWGDE and NIST in Section 3.1. The legal requirements are similarly met. Our solutions use computer data which are already collected and used in standard practice, or like firewall logs, are standard practice in computer networks and are easily enabled in OpenStack.

8. Discussion

We discuss advantages, limitations, trust assumptions, and open problems of FROST.

FROST offers advantages to forensic investigators over today's options for data acquisition. Obtaining a search warrant and serving it to a cloud provider, or requiring a system administrator's intervention to collect data puts the investigator at the mercy of others to complete the data acquisition. Today's forensic tools like EnCase Enterprise also perform acquisition inside the guest VM, which could be compromised. FROST performs acquisition at the host operating system.

One limitation of FROST is that it still requires trust in the cloud provider. In particular, users must trust the host operating system, hardware, network, and cloud employees. We (2012) previously explored options for addressing these concerns, such as rooting trusting in hardware with trusted platform modules. Today FROST concedes some trust in the system for the ability to perform forensics remotely. FROST comes as a software solution with almost no cost to the provider other than some disk space and the support required to maintain and troubleshoot FROST. More comprehensive trust solutions, including TPMs, require more substantial cost on a large scale, and hardware changes to the entire cloud infrastructure.

The enhanced logging mechanism is not foolproof. First, it is impossible to detect if an untrusted logger intentionally fails to record an event without access to the logger. Second, cloud clients could collude with the logger to roll back or modify events that may be difficult for a third party, such as law enforcement, to detect. Because we assume that the cloud provider is non-coercible, this concern is mitigated.

One open problem is preservation of data in the cloud. Rapid elasticity is a feature of cloud computing, but it comes with the challenge of preserving data in an investigation until that data can be identified and retrieved. OpenStack needs the capability for manual or automatic data preservation to maintain the record of activity of a malicious cloud user. This could be achieved by archiving logs and virtual disks for some period of time after the cloud consumer requests their disposal. However, forensic accountability may present tension with user privacy and requires careful thought.

Forensic examination of cloud layers remains open. For example, forensic capabilities for hypervisors or virtual networks and software-defined networking should be addressed.

Another open problem is the evolution and maturity of OpenStack. OpenStack has an active development community and regular software releases. Future modifications to OpenStack may affect FROST's functionality. We are working to add FROST to the public OpenStack project.

Future work remains in several areas. FROST is unique to IaaS environments. While the reference implementation has been done with OpenStack, implementations for other IaaS platforms are feasible. Platform-as-a-Service and Software-as-a-Service give less control to cloud users. Forensic capabilities for these environments remain to be done, and require considerations for their unique challenges. The provider will have to collect more of the forensic data, such as logging in the guest operating system.

Expansion within OpenStack also remains. It would be useful to support other virtual disk formats, since OpenStack supports many hypervisors. Other forensic capabilities could be added to FROST, such as data preservation or server-side e-discovery. Users will need the capability to create snapshots of running instances, which will be possible with the latest versions of QEMU and Libvirt. It would be useful to have automated snapshots of virtual machines, and the ability to detect changes between snapshots. Forensic tools for object storage, like that provided by OpenStack Swift and Amazon S3, would be useful today.

Our preliminary evaluation indicates that FROST is able to support many users without introducing undue

overhead. More comprehensive testing in a large scale environment is needed to gage the performance impact. We suggest an analysis to compare a production environment with and without FROST.

While FROST implements the acquisition phase of the forensic process, future work should consider solutions for other phases of the process affected by cloud computing. FROST produces data that can be consumed by standard tools, but as data volumes increase other analysis tools will be necessary.

9. Conclusion

We have introduced the FROST suite for OpenStack, the first collection of forensic tools integrated into the management plane of a cloud architecture. These tools enable cloud consumers, law enforcement, and forensic investigators to acquire trustworthy forensic data independent of the cloud provider. In addition to incident response and forensics, FROST can also be used for real-time monitoring, metrics, or auditing.

FROST offers concrete user-accessible forensic capabilities to cloud consumers. While many organizations are still hesitant to adopt cloud solutions because of security concerns, FROST arms them with powerful and immediate response capabilities. Similar tools should be a part of all commercial cloud services, and we look forward to the creation and adoption of more such tools to enhance forensic readiness for cloud computing.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful suggestions. We thank Simson Garfinkel, Ken Zatyko, and Tim Leschke for comments on early drafts. We also thank Ron Rivest and Stuart Haber for insights and suggestions related to hash trees.

Sherman was supported in part by the Department of Defense under IASP grants H98230-11-1-0473 and H98230-12-1-0454, and by the National Science Foundation under SFS grant 1241576. Dykstra was supported in part by an AWS in Education grant award.

References

Amazon Web Services. Amazon web services: overview of security processes. Available at: http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf; 2011 [accessed 28.10.12].

Amazon Web Services. Amazon CloudWatch. Available at: http://aws.amazon.com/cloudwatch/; 2013 [accessed 05.02.13].

Clarke DE. Towards constant bandwidth overhead integrity checking of untrusted data [Ph.D. thesis]. MIT; 2005.

Crosby SA. Efficient tamper-evident data structures for untrusted servers [Ph.D. thesis]. Rice University; 2009.

Dykstra J, Riehl D. Forensic collection of electronic evidence from infrastructure-as-a-service cloud computing. Richmond Journal of Law and Technology 2012;19. Available at: http://jolt.richmond.edu/wordpress/?p=463.

Dykstra J, Sherman AT. Understanding issues in cloud forensics: two hypothetical case studies. In: Proceedings of the 2011 ADFSL Conference on digital forensics security and law. ASDFL; 2011a. p. 191–206.

Dykstra J, Sherman AT. Understanding issues in cloud forensics: two hypothetical case studies. Journal of Network Forensics 2011b;3(1):19–31.

Dykstra J, Sherman AT. Acquiring forensic evidence from infrastructure-as-a-service cloud computing: exploring and evaluating tools, trust, and techniques. Digital Investigation 2012;9(Suppl. S90–S98). The Proceedings of the Twelfth Annual DFRWS Conference.

Garfinkel S. Digital forensics xml and the dfxml toolset. Digital Investigation 2012;8(3–4):161–74.

Google. Access logs & storage data (experimental) – Google cloud storage. Available at: https://developers.google.com/storage/docs/accesslogs; 2012 [accessed 28.10.12].

Haber S, Hatano Y, Honda Y, Horne W, Miyazaki K, Sander T, et al. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In: Proceedings of the ACM Symposium on information, computer & communication security (ASIACCS'08) 2008. p. 353–62.

Kundu A. Data in the cloud: authentication without leaking [Ph.D. thesis]. Purdue University; 2010.

Li J, Krohn M, Mazières D, Shasha D. Secure untrusted data repository (SUNDR). In: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6 (OSDI'04). Berkeley, CA, USA: USENIX Association; 2004. p. 9–9.

Martini B, Choo KKR. An integrated conceptual digital forensic framework for cloud computing. Available at: http://dx.doi.org/10.1016/j.diin.2012.07.001; 2012 [accessed 10.09.12].

Marty R. Cloud application logging for forensics. In: Proceedings of the 2011 ACM Symposium on applied computing. New York, NY, USA: ACM; SAC '11 2011. p. 178–84.

Merkle RC. A digital signature based on a conventional encryption function. In: A Conference on the theory and applications of cryptographic techniques on advances in cryptology. London, UK, UK: Springer-Verlag; CRYPTO '87 1988. p. 369–78.

Microsoft. About storage analytics logging. Available at: http://msdn.microsoft.com/en-us/library/windowsazure/hh343262.aspx; 2012 [accessed 12.11.12].

National Institute of Standards and Technology. Digital data acquisition tool specification. Available at: http://www.cftt.nist.gov/Pub-Draft-1-DDA-Require.pdf; 2004 [accessed 16.09.12].

National Institute of Standards and Technology. The NIST definition of cloud computing. Available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf; 2011 [accessed 05.01.13].

OpenStack. Keynote recap, day 2: why we do what we do. Available at: http://www.openstack.org/blog/2012/10/keynote-recap-day-2-why-we-do-what-we-do/; 2012a [accessed 26.10.12].

OpenStack. OpenStack open source cloud computing software. Available at: http://www.openstack.org/; 2012b [accessed 13.12.12].

Ruan K, Carthy J, Kechadi T, Crosbie M. Cloud forensics: an overview. In: Advances in digital forensics VII 2011.

Scientific Working Group on Digital Evidence (SWGDE). Data integrity within computer forensics. Available at: https://www.swgde.org/documents/Current%20Documents/2006-04-12%20SWGDE%20Data%20Integrity%20Within%20Computer%20Forensics%20v1.0; 2006 [accessed 16.09.12].

Taylor M, Haggerty J, Gresty D, Lamb D. Forensic investigation of cloud computing systems. Network Security 2011;3:4–10.

U.S. Dept. of Justice. Searching and seizing computers and obtaining electronic evidence in criminal investigations. Available at: http://www.justice.gov/criminal/cybercrime/docs/ssmanual2009.pdf; 2009 [accessed 16.09.12].