



Norwegian University of
Science and Technology

Securing the IaaS Service Model of Cloud Computing Against Compromised Components

Aryan TaheriMonfared

Master in Security and Mobile Computing

Submission date: June 2011

Supervisor: Danilo Gligoroski, ITEM

Co-supervisor: Martin Gilje Jaatun, Sintef

Master Thesis Problem Description

Aryan TaheriMonfared

June 20, 2011

Securing the IaaS Service Model of Cloud Computing Against Compromised Components

According to a definition which is proposed by National Institute of Standards and Technology (NIST), Cloud computing is a model for on-demand network access to a shared pool of resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released. This process is done with minimal management effort or interaction with cloud providers. Cloud customers will have higher availability by means of this new model.

Moreover, Cloud computing is a new computing model and security is by many ranked first among challenges of the cloud model. In order to secure a cloud environment, it is crucial to harden the cloud infrastructure in a way that can handle and tolerate compromised components.

This research topic is not only recommended by European Network and Information Security Agency (ENISA) but also is identified as a major security challenge in an IaaS deployment of the Cloud model in OpenStack. OpenStack is one of the leading projects in developing opensource cloud platform. It will also cover parts of TClouds project objectives which is the European project on Trustworthy Clouds. This study is composed of three parts:

Possible cases of compromising a component

In this part, we will initially identify major attack scenarios in an IaaS deployment of the cloud computing model. This research is limited to those attacks that may lead to compromising of a component. These attacks can take place in different layers (i.e. Hardware, Operating System, Hypervisor and OS services). We will study cloud computing characteristics and their influences on each case. Moreover, risk analysis and building attack graphs for these scenarios will be further steps in our study.

Detection and Analysis of the compromised component

In the second part, we will extract possible symptoms of different attacks. Symptoms are useful in detecting an attack. Proactive and reactive measures can be applied so as to detect the compromised component. By analyzing the nature of the compromised component, we gather enough information for handling the component and mitigating corresponding risks.

Containment and Recovery of the compromised component

When compromised components are detected, they must be handled. First we

should secure the cloud environment so it won't fail catastrophically due to the compromised component. Securing the cloud can be done in several ways. As an example, by partitioning the network and isolating the compromised component or even shutting down the host of that component. In the second phase, the cloud infrastructure and components should be secured to prevent that attack from succeeding in the future.

Supervisor: Professor Danilo Gligoroski

Securing the IaaS Service Model of Cloud Computing Against Compromised Components

by

Aryan TaheriMonfared

MASTER OF SCIENCE

in

Faculty of Information Technology, Mathematics and Electrical Engineering

(Security and Mobile Computing)

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

(Trondheim)

June 2011

© Aryan TaheriMonfared 2011

Abstract

Cloud Computing is a new computing model, and its security aspects require special considerations. New characteristics of the cloud model have introduced new security challenges, and made some of the existing security techniques incompatible. Moreover, existing cloud environments are closed, operated by commercial providers, and their security mechanisms are proprietary as well as confidential. In other words, there is not much chance of observing how a real cloud environment is working, and how their providers adapt security measures to the new model.

Therefore, we have chosen an open source cloud platform to build our own cloud environment. The OpenStack cloud software met our requirements, but it was not mature enough. We have done a deep analysis of this platform, identified potential attack targets in it, and discuss impacts of a successful attack.

In order to secure our environment, the National Institute of Standards and Technology (NIST) incident handling guideline has been applied to the cloud model, and corresponding actions for each phase has been performed. To complete our study, we have proposed a set of cloud specific approaches that fulfill the incident handling requirements. These approaches address challenges identified in the guideline adaptation process. Additionally, we have studied the feasibility and compatibility of each approach against our deployed environment.

Additionally, we also have submitted a paper to IEEE CloudCom 2011 conference, based on my thesis. A draft version of the paper is included in Appendix A.

Table of Contents

Abstract	ii
Table of Contents	iii
List of Tables	vii
List of Figures	viii
Listings	x
Acknowledgements	xii
1 Introduction	1
1.1 Brief overview of Cloud Computing	1
1.1.1 Definition	1
1.1.2 Specifications	2
1.1.3 Challenges	6
1.2 Our Study Approach	7
1.3 Thesis Contributions and Report Structure	8
1.3.1 Openstack	9
1.3.2 Application of the Incident Handling Guideline	10
1.3.3 Proposed Approaches	10
1.3.4 Lab Setup and Configuration	11
1.4 Text Conventions	11
2 Background	12
2.1 Introduction to Cloud Computing Security	12
2.1.1 New Security Challenges	13
2.2 Risk Management	15
2.2.1 Methodology	15
2.3 Study Motivation	18
2.3.1 IaaS Security	18

Table of Contents

2.3.2	Security of a Compromised Component	19
3	OpenStack	21
3.1	Architecture	22
3.1.1	Hierarchical	22
3.1.2	Peer to Peer	22
3.1.3	Multiple Cluster Zones	22
3.2	OpenStack Compute Project (Nova)	25
3.2.1	Cloud Controller	26
3.2.2	Object Store (nova-objectstore)	30
3.2.3	Auth Manager	30
3.2.4	Volume Controller (nova-volume)	30
3.2.5	Network Controller (nova-network)	31
3.2.6	Scheduler (nova-scheduler)	32
3.2.7	Compute Controller (nova-compute)	33
3.2.8	API Server (nova-api)	35
3.2.9	Compute Interfaces	35
3.2.10	RBAC model in OpenStack	35
3.2.11	Operation	36
3.3	OpenStack Object Storage (Swift)	37
3.3.1	Components	38
3.4	OpenStack Imaging Service (Glance)	38
3.5	Other Components	39
3.6	Networking	39
4	Components at Risk	42
4.1	Virtualization Vulnerabilities	44
4.1.1	Vulnerabilities in the code	44
4.1.2	Types of vulnerabilities	45
4.1.3	Hypervisor	46
4.2	Cloud Platform (OpenStack)	47
4.2.1	Cloud Controller	49
4.2.2	Scheduler	51
4.2.3	Volume Controller	51
4.2.4	Network Controller	53
4.2.5	Compute Controller	54
5	Detection and Analysis of an Incident (Compromised Com- ponent)	56
5.1	Incidents	56

Table of Contents

5.2	Detection	57
5.2.1	Challenges	58
5.2.2	Detection Approaches	58
5.3	Analysis	59
5.3.1	Challenges	59
5.4	Actors' Requirements	60
5.4.1	Cloud Providers' Requirements	60
5.4.2	Cloud Consumers' Requirements	62
5.4.3	Challenges of Proposed Approaches	63
5.5	Detection and Analysis in an OpenStack Deployment	64
5.5.1	Identifying signs of an incident	65
5.5.2	Specifying precursors and indications sources	65
5.5.3	Analysis of the incident	67
6	Containment and Recovery of the Compromised Component	69
6.1	Existing Approaches	70
6.1.1	Intrusion Tolerance	70
6.1.2	Deployment Models	73
6.2	Containment, Eradication, and Recovery in an OpenStack Deployment	74
6.2.1	Case One: A Compromised Compute Worker	76
6.2.2	Case Two: A bogus component	88
7	New Approaches	95
7.1	Restriction of Infected Components	95
7.1.1	Filtering in the messaging server (cloud controller)	95
7.1.2	Filtering in each component	103
7.1.3	Disabling services	107
7.1.4	Removing instances from the project VLAN	109
7.1.5	Locking down instances' live migration	109
7.1.6	Quarantining instances	110
7.2	Replication of Services	114
7.2.1	Replication Layers	114
7.3	Disinfection of Infected Components	115
7.4	Consumer Approaches	116
7.4.1	Reactive	116
7.4.2	Proactive	117

Table of Contents

8 Lab Setup	118
8.1 OpenStack Compute Deployment	118
8.1.1 System Requirements	119
8.1.2 Architecture/Structure	120
8.1.3 Component Distribution	122
8.1.4 Installation	122
8.1.5 Configuration	125
8.1.6 Management	129
8.1.7 Operation	131
8.2 Performance Monitoring of the Infrastructure	132
8.2.1 Installation and Configuration	133
8.2.2 Data Sources and Graphs	133
8.3 Messaging Server Management and Monitoring	139
8.3.1 Installation	139
8.3.2 Operating	140
9 Conclusion	148
References	150
Glossary	158
Acronyms	160
 Appendices	
Appendix A - Paper	162

List of Tables

1.1	Actors activities in different service models [78]	4
2.1	Fortify Software survey at DEFCON 2010	13
2.2	Cloud Security Survey by CA Technologies [61]	20
4.1	Virtualization Vulnerabilities [64]	47
4.2	Codes	48
4.3	Cloud Controller Specifications	50
4.4	Scheduler Specifications	52
4.5	Volume Controller Specifications	52
4.6	Network Controller Specifications	53
4.7	Compute Controller Specifications	55
5.1	Incident detection approaches [40]	59
5.2	Incident analysis approaches [40]	60
6.1	Case One - A compromised compute worker scenario specifications	77
6.2	Containment Strategies	84
6.3	Case Two - A bogus component scenario specifications	89
8.1	Hardware Requirement for Cloud Controller	120
8.2	OpenStack-1 specification and services	123
8.3	OpenStack-2 specification and services	123
8.4	OpenStack-3 specification and services	123
8.5	OpenStack-4 specification and services	124
8.6	Cloud controller parameters and our deployment details [56]	127

List of Figures

1.1	Cloud Computing Service Models	3
1.2	Cloud Computing Definition [59]	5
1.3	Actor hierarchy	6
3.1	OpenStack projects and their relation [56]	21
3.2	OpenStack Hierarchical Architecture[38]	23
3.3	OpenStack Peer-to-Peer Architecture[38]	23
3.4	OpenStack Compute MultiCluster Zones[82]	24
3.5	MultiCluster Zones Implementations[82]	25
3.6	Nova components and their interaction[84]	26
3.7	RabbitMQ Internals and two modes[12]	28
3.8	OpenStack Compute basic architecture [82]	36
3.9	Swift components and their interaction[71]	37
4.1	Cloud Components	43
4.2	OpenStack Compute basic architecture [82]	49
6.1	An example of passive replicas and failures in the CC-VIT approach.	72
6.2	Case One - The nova-compute service in the OpenStack-4 host is compromised.	78
6.3	Blocking compromised compute communication. Red lighting represent disconnected communications.	80
6.4	OpenStack Nova services dependencies.	81
6.5	Stopping the compute service at the compromised host.	82
6.6	Discarding messages to/from the compromised node.	83
6.7	Case Two - A physical bogus compute worker node is added to the infrastructure.	88
6.8	Case Two - A virtual bogus compute worker is added as a consumer's instance.	90
6.9	Case Two - Bogus worker's connections are established.	91

List of Figures

6.10	Case Two - Platform wide exchanges are binded to the bogus worker.	91
6.11	A sample markov model for trust states of a component. . . .	93
6.12	A sample markov model for transitions between different trust levels of a component.	94
7.1	Closing a connection using RabbitMQ management	98
7.2	Unbinding a queue from an exchange using the Queues Management page of the RabbitMQ	100
7.3	Publishing a message to a queue using RabbitMQ management	101
7.4	Deleting an exchange using RabbitMQ management	102
7.5	Unbinding a queue from an exchange using RabbitMQ management	102
7.6	Deleting or purging a queue using RabbitMQ management .	103
7.7	Overview of RabbitMQ messaging server and applicable containment approaches.	104
7.8	Overview of possible filtering points in each component . . .	106
8.1	Logical relation of entities in an OpenStack environment . . .	119
8.2	Hosts structure in our laboratory configuration	121
8.3	A compute node in our laboratory configuration	121
8.4	Running services on each physical host	122
8.5	Message flow for running an instance [46]	132
8.6	CPU Utilization	134
8.7	NIC Traffic	135
8.8	Memory Usage	136
8.9	Combined Traffic statistics	136
8.10	Disk Input/Output	137
8.11	TCP Protocol statistics	138
8.12	Load Average	138
8.13	RabbitMQ Management Overview	141
8.14	RabbitMQ Node rabbit@openstack-1 Overview	142
8.15	RabbitMQ Connections	142
8.16	RabbitMQ a Connection Details	143
8.17	RabbitMQ Channels	143
8.18	RabbitMQ a Channel Details	144
8.19	RabbitMQ Exchange	144
8.20	RabbitMQ an Exchange Details	145
8.21	RabbitMQ Queues	146
8.22	RabbitMQ a Queue Details	147

Listings

6.1	Disabling a system user	85
6.2	Revoking an OpenStack user's credentials	86
6.3	Deleting an OpenStack user	86
6.4	List of services in the environment	91
7.1	Closing a connection using RabbitMQ command line interface	98
7.2	Disabling the nova-compute service	108
7.3	libvirt options for monitoring instances	110
7.4	Monitoring network interface statistics of an instance	111
7.5	virt-top sample output	111
8.1	Adding Nova package repository	122
8.2	Installing Nova packages	124
8.3	Installing the Glance package	124
8.4	Installing the SNMP agent	124
8.5	Installing Nova and SNMP Agent	125
8.6	nova.conf	125
8.7	Setting nova.conf permissions	127
8.8	Configuring MySQL for nova	127
8.9	Modified lines of snmpd.conf	128
8.10	Authorizing ICMP and SSH	128
8.11	Adding iptables rule to redirect the traffic	128
8.12	Generating credentials	129
8.13	Sourcing <i>novarc</i>	129
8.14	Uploading a VM image	129
8.15	Creating a key pair and running an instance	130
8.16	Checking an instance status	130
8.17	Terminating an instance	130
8.18	Deleting a bundle	131
8.19	Installing and configuring SNMP daemon	133
8.20	Installing and configuring SNMP daemon	133
8.21	Removing the existing RabbitMQ server and Erlang package	139

Listings

8.22	Installing Erlang dependencies	139
8.23	Installing Erlang from its source	139
8.24	Installing RabbitMQ	140
8.25	Installing RabbitMQ Management and Monitoring plug-in . .	140

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Martin Gilje Jaatun, whose encouragement, guidance, and support from the initial to the final step, enabled me to develop the idea and write the thesis.

I would also like to thank Professor Danilo Gligoroski and Professor Tuomas Aura for their encouragement and insightful comments.

Finally, I am immensely grateful to my family and friends, and dedicate the thesis to them; for their understanding, support, and endless love.

Aryan TaheriMonfared
June 2011, Trondheim

Chapter 1

Introduction

1.1 Brief overview of Cloud Computing

Cloud Computing is an old idea of providing computing resources as a utility. This computing model will reduce the upfront cost for developing and deploying new services in the Internet. In such an environment, resources can scale down and scale up quickly, thus under-provisioning and over-provisioning will not be a major threat to services in the cloud model.

Cloud Computing is a new computing model, and its definition and specifications are not standardized yet. There have been several attempts in providing a complete description of the cloud model, but they are not widely accepted either. The National Institute of Standards and Technology (NIST) definition of Cloud Computing [48] and the Berkeley view of Cloud Computing [20] are two main contributions in this field.

1.1.1 Definition

We will use the NIST definition of Cloud Computing [48] in the rest of our research.

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.” [48]

Essential characteristics of Cloud Computing are the main items in its definition. They emphasize new aspects of this model which discriminate it from other computing models.

1. **On-demand self-service:** A cloud consumer can provision computing resources without human intervention. [48]

2. **Broad network access:** Cloud Computing utilizes existing networking technologies to deliver services to customers and provide connectivity among stakeholders [48].
3. **Resource pooling:** Each cloud provider has several customers. Customers provision computing resources dynamically from a resource pool and release them to the pool when there is no demand [48].
4. **Rapid elasticity:** Provisioning of resources can happen rapidly, also the demand for resources may vary dynamically. When the demand increases, more resources are provisioned to scale out¹ and when it decreases, provisioned resources are released to scale in. This procedure happens quite fast [48].
5. **Measured services:** A pay-per-use business model is employed to measure the resource usage. Resource usage for different type of services are metered based on the service type criteria. Also the provisioning is managed and reported to required stakeholders [48].

1.1.2 Specifications

Service Models

Additionally, cloud services can be arranged into several service models. Each model addresses a set of customers' requirements. Three service models capture multiple groups of customers.

A group of customers may only require provided applications by the cloud provider; the mail service provided by Google is an example of such an application. This type of services are grouped in *Cloud Software as a Service (SaaS)* [48].

Another group may require more capabilities and control over provisioned resources. The customer can deploy its own applications using tools and libraries provided by the cloud provider. This type of cloud services are *Cloud Platform as a Service (PaaS)* [48].

The last model is *Cloud Infrastructure as a Service (IaaS)*. The customer has the most access to underlying layers and resources in this service model. The customer can choose a particular operating system and deploy any application on it [48].

¹Scaling out or horizontal scaling is referred to the application deployment on multiple servers [49].

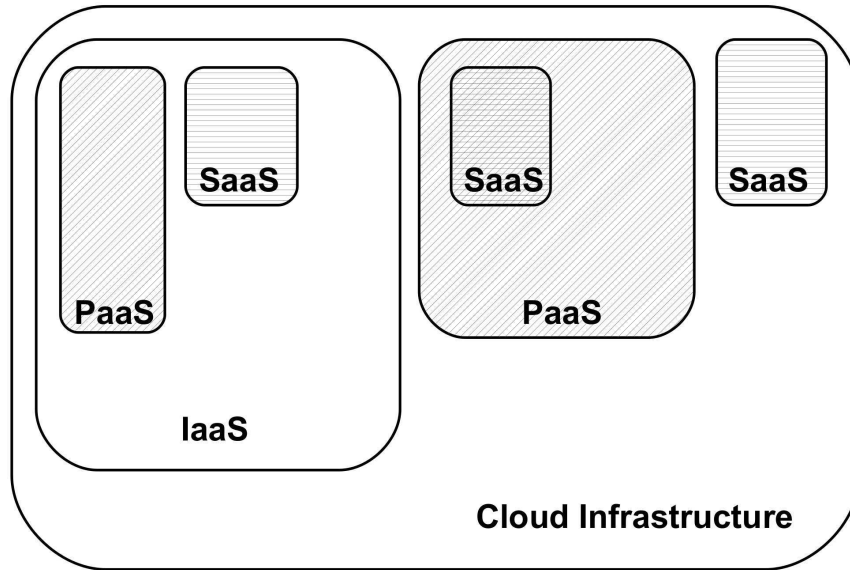


Figure 1.1: Cloud Computing Service Models

Cloud Computing service models and their combinations are depicted in Figure 1.1. Table 1.1, inspired from [78], explains the consumer's and provider's activities in the cloud model .

Deployment Models

The defined cloud by NIST has three main deployment models and a fourth one which is the composition of others.

When a single organization operates the cloud infrastructure, the *private cloud* deployment model is used. The infrastructure in this deployment model can be administrated locally or by third parties, also resources may exist on premise or off premise [48].

When several organizations, with similar goals, operate the cloud infrastructure, the *community cloud* model is used. Administration and resource location can be handled locally or by third parties [48].

The third deployment model is the *public cloud*. The cloud infrastructure in this deployment model is available to the public; The responsible organization may provide variety of cloud services using the public cloud model [48].

²The management process may contain development, deployment, maintenance and support.

1.1. Brief overview of Cloud Computing

Service Model	Consumer Activity	Provider Activity
SaaS	Use provided service for business activities	Manage ² the software application over underlying infrastructure
PaaS	Develops, integrates and administrates applications	Provide libraries and tools for consumers; control and maintain cloud resources and provided platforms
IaaS	Creates, configures and administrates virtual machines	Control physical resources and provide computing infrastructure for consumers

Table 1.1: Actors activities in different service models [78]

Hybrid cloud is a composition of several deployment models that supports the application portability.

NIST definition of Cloud Computing is depicted in Figure 1.2 [59].

Additionally, there are some similarities between Cloud Computing, Service Oriented Computing (SOC), Grid Computing, and High Performance Computing [32].

Stakeholders and Actors

NIST has identified five major actors in the NIST Cloud Computing Reference Architecture [78]. Each actor has a role and is responsible for a set of activities and functions.

- **Cloud Consumer**

The individual or organization that uses cloud services, provided by the cloud provider [78].

- **Cloud Provider**

The actor that provides and delivers cloud services to its consumers [78].

- **Cloud Auditor**

The cloud auditor is responsible for auditing cloud services, system operations, performance and security [78].

- **Cloud Broker**

The broker is the negotiator between the cloud consumer and provider [78].

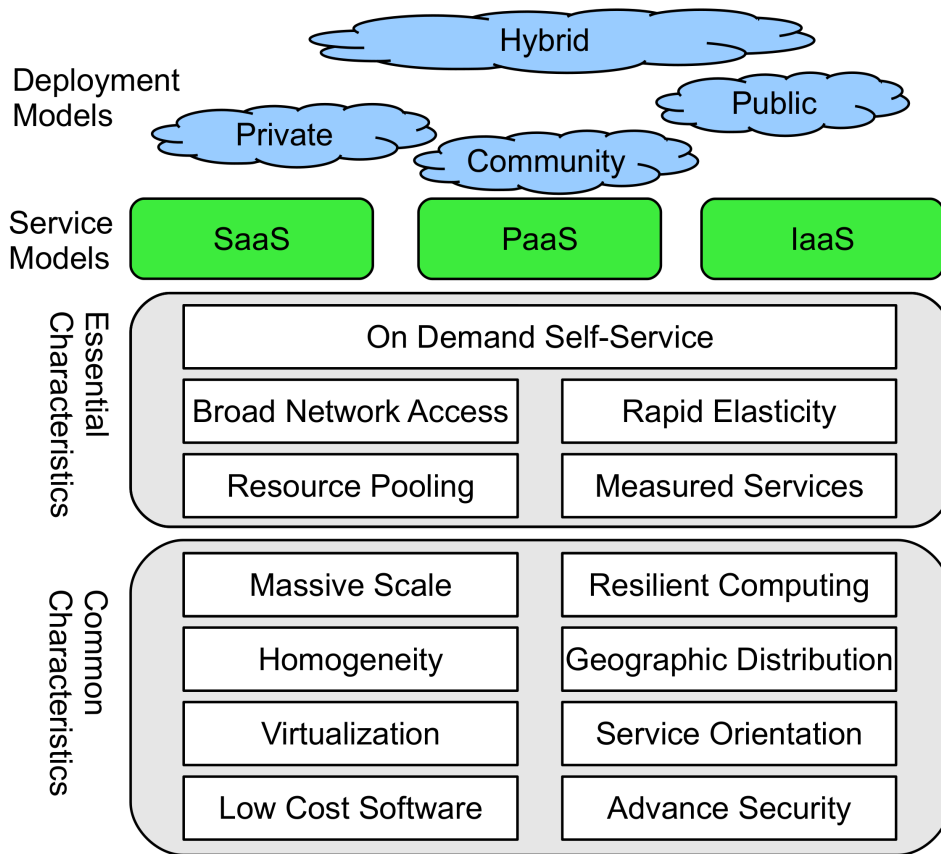


Figure 1.2: Cloud Computing Definition [59]

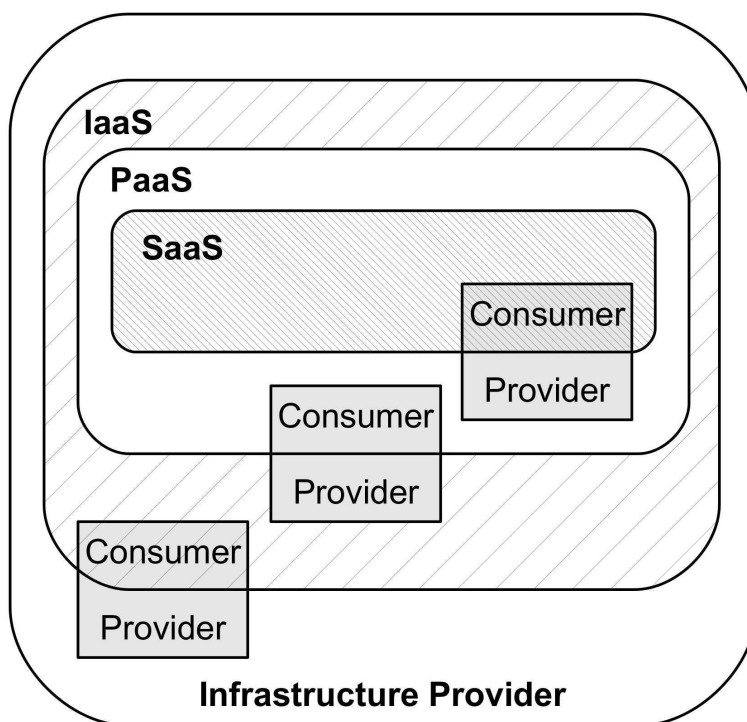


Figure 1.3: Actor hierarchy

- **Cloud Carrier**

The carrier is responsible for provider-consumer connectivity [78].

As depicted in Figure 1.3, the consumer-provider relation can be used hierarchically to describe the long chain of interactions among infrastructure providers to end users.

Different literatures use different terms for same actors, some of them identified more actors with different roles that may expand a cloud environment. In our study we will mainly use NIST terms and taxonomy.

1.1.3 Challenges

Armbrust et al. [20]. referred to 10 challenges in the new computing model. Challenges can be categorized into four groups [20]:

1. Adoption
 - Availability

- Lack of common interfaces and data lock-in
- Data confidentiality and auditability

2. Growth

- Networking bottleneck
- Unpredictable VM performance
- Storage scalability
- Distributed system bugs
- Rapid elasticity

3. Marketing

- Reputation isolation
- Licensing

These are several abstract challenges; we focus more on security related issues or security aspects of these challenges. We will introduce new security challenges in a cloud environment in Section 2.1.1.

1.2 Our Study Approach

Generally in a distributed environment with several stakeholders, numerous ways of attacking and compromising a component exist. Moreover, it is not possible to stop all attacks or to ensure that the system is secure enough against all threats, specifically in a highly distributed system.

Thus, the best approach is to understand impacts and assess the risk of a compromised component. So, we don't study attack methods, instead impacts of a compromised component on the provided service and other components will be analyzed. In order to study impacts of a successful attack, functionalities of each component are extracted. Functionality extraction was one of the main contributions in this part, because the documentation provided for the cloud platform (i.e. OpenStack) was not detailed enough. Moreover, studying impacts of an attack is not possible by simply going through documents instead of reviewing the code and analyzing the platform in a real deployment. In addition to component functionalities, interacting entities with a specific component and corresponding access methods will also be discussed. They are useful measures in recognizing impacts of a compromised component. This part is mainly discussed in Section 4.

After identifying impacts of a successful attack, we should find efficient approaches to tolerate such an attack and its damages. In this process, the incident should be detected and analyzed first. Detecting and analyzing an incident have a standard procedure that requires knowledge about the normal behavior and operation of the system. For this purpose, we used a variety of monitoring and profiling techniques to recognize the normal behavior of a distributed system. It should be noted that in an experimental environment the normal behavior of a system may vary from a real one with consumers' interaction. Another contribution in this part is applying the NIST incident detection and analysis procedure to the cloud model. Each step in the NIST guideline is mentioned and corresponding tasks has been done in our cloud environment. In addition to the NIST guideline, we studied other approaches which are proposed for a distributed system and discussed their advantages and disadvantages in an Infrastructure as a Service (IaaS) service model of a cloud environment. Finally, an abstract list of requirements for major actors in the cloud model is proposed that can facilitate the detection and analysis phase.

The last step for handling an incident is to contain and eradicate the incident and recover the system. We use the NIST guideline for incident handling [45] to perform the last step. In the first part we review multiple solutions and recommendations for the incident containment, eradication, and recovery. In the second part, we use the NIST recommendations to contain the incident in our experimental cloud environment.

We use the experience and results of this phase to propose a set of approaches for the incident containment, eradication, and recovery.

1.3 Thesis Contributions and Report Structure³

Contributions of my thesis can be divided into 4 categories, comprising:

1. Contributions to the OpenStack project and community, such as functionalities extraction, code analysis, bug report (Chapters 3, 4).
2. Application of the NIST incident handling guideline to the Cloud Computing model (Chapters 4 , 5, 6).
3. A set of proposed approaches to overcome challenges of the incident handling adaptation (Chapter 7).

³We strongly recommend the reader to access the mind map file to study the report structure and have an overview of the text flow. Mind map files can be accessed in the following address: <http://org.ntnu.no/cloudsecurity/thesis/Docs/>

4. Deployment and configuration of a real cloud environment with the required set of tools for monitoring and analysis (Chapter 8).

Additionally, a few characteristics of our research should be mentioned here:

- We identified, and analyzed security differences introduced by the new computing model and major obstacles in applying existing techniques.
- Instead of diving deep into a specific solution, we studied a variety of solutions, their advantages and disadvantages. This helps to decide which type of solutions are more feasible in a cloud environment; and what are their usecases.
- Approaches are tested and examined in our deployed environment to study their effectiveness and implications.
- A few approaches were not applicable in our environment, however we introduced them and discussed their hindering obstacles as well.

All documents and resources related to the thesis can be found in the following address: <http://org.ntnu.no/cloudsecurity/>. A set of links to the deployed environment is also provided in that page. Required credentials are available upon your request.

More details about each category is given in the following parts.

1.3.1 Openstack

Deep analysis of the OpenStack Compute project is one of our contributions. We identify each component in this cloud platform and study a specific set of parameters about each of them, including: components' functionalities, connected components, access methods, and impacts.

These specifications are not explicitly and clearly noted in the OpenStack documentations, although they play important roles in the incident handling and threat management of a component. Thus, we study the source code, interact with the OpenStack community, and experiment on our lab deployment to extract specifications, responsibilities and behaviors of each component. Additionally, services dependency and communication models are also discussed.

In this process several bugs have been identified and reported to the community.

The OpenStack cloud platform is discussed in Chapters 3 and 4.

1.3.2 Application of the Incident Handling Guideline

NIST has published a guideline [45] for incident handling. The guideline has a general approach and is neither tuned nor prepared for a cloud environment. One of the main contributions of my thesis is the application of the NIST guideline to the Cloud Computing model.

Each phase of the incident handling procedure is studied, and the corresponding recommended actions have been applied to the cloud model. The adaptation process faced major obstacles that have been introduced due to the new characteristics of the cloud model.

It should be noted that we focus on technical aspects of the incident handling procedure rather than the organizational and management aspects. Moreover, in a cloud environment incident handling mechanisms can be implemented in different cloud components; another contribution was identifying appropriate components that can contain the mechanism. In this part we recognized the proper component, propose the applicable approach, and discuss their implementation details.

Finally, two attack scenarios are discussed. Each scenario consists of a set of incidents with different targets and impacts. The incident handling procedure is applied for them and the raising challenges are determined.

This application process is explained in Chapters 4, 5, and 6.

1.3.3 Proposed Approaches

We identify challenges of applying the NIST incident handling guideline to the Cloud Computing model. We study these challenges and propose a set of solutions, some of them are the combination or the modified version of existing solutions, and some others are completely new.

For each approach following steps are performed:

1. Explaining the approach **specifications**.
2. Determining its **advantages and disadvantages**.
3. Specifying its **implementation details** in a cloud environment that is powered by the OpenStack cloud platform.
4. Identifying applicable **tools and design principles** for the provided solution.

Our proposed solutions are discussed in Chapter 7.

1.3.4 Lab Setup and Configuration

Each and every step of the thesis is based on our experimental deployment of a cloud environment using the OpenStack cloud platform. In our lab setup, we have four physical machines that are working as our cloud infrastructure. On top of them we have several software and tools, including: performance monitoring software, Advanced Message Queuing Protocol (AMQP) server management and monitoring software, cloud administration panel, etc. Detailed specifications of our lab setup is given in Chapter 8.

A paper based on the thesis

In order to publish our research result more effectively, and discuss their weaknesses and challenges, we have written a paper based on it. The paper is submitted to the IEEE CloudCom 2011 conference. A draft version of our paper can be found in Appendix A - Paper.

1.4 Text Conventions

- When the term *component* is used, we are addressing a software component or service in a cloud environment, unless explicitly stated otherwise.
- The OpenStack software has a project called *OpenStack Compute project*, and its code name is *Nova*. Nova has a component which is again named *compute*, or *nova-compute*. The node which is running a *compute* service is called *compute worker node/compute node/compute host*.
- When *layers* in a stack are addressed throughout the text, *cloud model layers* are meant, unless explicitly stated otherwise.
- In most cases instead of the long term "*virtual machine instance*", the shorter form is used "*instance*".

Chapter 2

Background

2.1 Introduction to Cloud Computing Security

"By 2012, 20 percent of businesses will own no IT assets. Several interrelated trends are driving the movement toward decreased IT hardware assets, such as virtualization, cloud-enabled services, and employees running personal desktops and notebook systems on corporate networks." Gartner 2010 prediction report [29].

One of the main obstacles in this movement toward Cloud Computing is its security challenges. A new computing model brings its own security doubts and issues to the market. Although Cloud Computing does not have any new technologies, its characteristics, service models and deployment models raise new security issues. As an illustration, a missing clear definition and specification of cloud perimeters, its dependability parameters and data confidentiality and integrity in this model are parts of those security challenges. Y. Chen et. al. [27] argued that most of security issues in Cloud Computing are not fundamentally novel. Many of them have been discussed in the time-sharing era. They introduced two novel security issues: *"the complexities of multi-party trust considerations"* and *"the crucial demand for mutual auditability"* [27].

We explained Cloud Computing specifications previously in Section 1.1, thus we just refer to them and study their corresponding security challenges here. As an example virtualization is one of the specifications of a cloud environment, and according to a Gartner report [28]

"60% of virtual servers will be less secure than the physical servers they replace through 2012."

Moreover, in addition to new security challenges in this computing model, attackers are becoming more interested in Cloud Computing. A Fortify Software survey [36] at the DEFCON 2010 conference emphasized on this interest and attention. The result revealed that among 100 participants of

2.1. Introduction to Cloud Computing Security

Percentage	Claim
96%	Cloud will provide more hacking opportunities
89%	Cloud providers were not proactive enough
45%	Already engaged in Cloud hacking
12%	Hacking for financial gain

Table 2.1: Fortify Software survey at DEFCON 2010

the survey, who were all IT professionals, almost half of them claimed that they already started intruding into cloud environments. Detailed results can be found in Table 2.1.

2.1.1 New Security Challenges

According to the Cloud Computing definition which is proposed by NIST [48], a cloud environment has five essential characteristics:

1. On-demand self-service
2. Broad network access
3. Resource pooling
4. Rapid elasticity
5. Measured Service

They introduce new security challenges to a cloud environment.

- **An alternative for botnets.** Cloud Computing offers more reliable and trustworthy functionalities as botnets [27].
- **Side channel and Covert channel vulnerabilities.** A cloud environment obviously employs resource sharing concept. Resource sharing increase the vulnerability of the system to Side Channel and Covert Channel attacks. This issue, related impacts and countermeasures are discussed throughly by T. Ristenpart et al. [65]
- **Long, sophisticated chain of trust.** Complex relations between variety of stakeholders in Cloud Computing model introduce another challenge into a cloud environment. cloud consumers should prove trustworthiness of provided services to their clients. It requires trust to corresponding Cloud providers and transitivity of this trust to clients [74].

- **Data jurisdiction and conflict of laws.** Data in the cloud can be stored in different geographical and organizational storage. This possible location diversity may have conflicts with data protection laws and constrains in some authorities.

These conflicts arise in different cases. First, a government may restrict storage location of a specific type of data. As an example we can refer to restriction by Canadian government on citizens' data location [60] and utilization of Cloud Computing for storing citizens' health records. Another case is when an agency is authorized to intercept or inspect specific data. If data is stored in the cloud and the cloud provider is not in the same authority domain of that agency, lawful interception will not be not feasible. US Patriot Act [35] is an example of a law that allows specific agencies to intercept communications for US safety, on the other hand Data Privacy Protection Directive (DPPD) [33] of the European Union prohibits such an interception. Thus if an American agency demands for lawful interception of a service which is running in the cloud that is mainly hosted in Europe, this conflict will arise [74].

- **Reputation Isolation [27] (Fate-sharing [22]).** Sharing same resources among a large group of customers with different use-case scenarios can have negative impacts on customers' reputation. As an example, blacklisting of innocent Amazon EC2 customers can be mentioned. In that incident a malicious Amazon EC2 customer had used the provided service for spamming, as a result a range of Amazon EC2 IP addresses had been blocked and blacklisted which affected even legitimate customers [74].

- **Applying old-fashioned incident handling procedure to the new computing model.** Long chain of trust with more stakeholders and complex interaction has made incident handling more sophisticated in this model. It is crucial to handle the incident and share enough information regarding the incident while at the same time the isolation among each stakeholder is maintained and their specific privacy policies remain intact [74].

Cloud specific incident handling has been studied by J. Reed discussed [42]. Research has been done on challenges and approaches toward incident handling in the cloud by Grobauer et al. [40]

- **Lack of standards and common interfaces increases data lock-**

in probability. These shortcomings in the current cloud environment disturb clouds interoperability and customers migration facilities. Under these circumstances after a catastrophic failure in a cloud environment, customers cannot migrate to another cloud smoothly. Thus a failure in a specific cloud environment will affect its customers adversely [48], [27].

- **Deletion, destruction and disposal of customers' data in the cloud.** Most cloud providers prefer to hide data or make them inaccessible instead of deleting them physically. Physical data deletion requires complex procedures and algorithms which are not worth deploying in all cases. This is a major obstacle in distributed systems with distributed storages.
- **Demand for mutual auditability among stakeholders [27].** As discussed previously, actors' complex interactions require a framework to ensure trustworthiness of each party to others. The framework should provide mutual auditability in a cloud environment.

2.2 Risk Management

Risk management of a given Cloud environment helps cloud stakeholders to balance the protection cost and security mechanisms that are utilized. Most companies have limited budgets for security; thus it is crucial to use appropriate risk management methodology to meet mission essential security requirements. [72]

2.2.1 Methodology

Risk management can be done in each phase of the System Development Life Cycle (SDLC). NIST [72] introduces five major phases in the SDLC, comprising: Initiation, Development/Acquisition, Implementation, Operation/Maintenance, Disposal.

The first step in the risk management is called *Risk Assessment*. Risk assessment determines potential threats and risks in a system [72]. The output of risk assessment phase will be useful in the next phase (i.e. *Risk Mitigation*). It helps in identifying appropriate procedures and measures for elimination and mitigation of risks. Risk Management Guide by NIST defines Risk as follows:

”**Risk** is a function of the **likelihood** of a given **threat-source’s** exercising a particular potential **vulnerability**, and the resulting **impact** of that adverse event on the organization.”

Thus, we should initially study threats in the IaaS service model of Cloud Computing against vulnerabilities in this model. Then possible impacts of exploiting each vulnerability should be determined. The risk assessment methodology has nine steps [72]:

1. System Characterization:

By characterizing an IT system and its corresponding resources, we can determine boundaries for risk assessments. Cloud Computing is a new model so you may not find a consistent definition and characterization yet. There are several definitions and characterizations articles in academia and standard institutes, we chose two articles in our study as references for the cloud characterization: The *NIST Definition of Cloud Computing* [48] and *Above the Clouds: A Berkeley View of Cloud Computing* [20]

2. Threat Identification:

Next step is to identify threats to Cloud Computing. By definition a threat is [72]:

”The potential for a particular threat-source to successfully exercise (accidentally trigger or intentionally exploit) a particular vulnerability.”

Cloud Security Alliance (CSA) [5] did an extensive study on top threats to Cloud Computing [30], that is useful in this phase.

3. Vulnerability Identification:

In this step flaws and weaknesses in the given service model of the cloud are studied. These weaknesses can be in any components/layers of the cloud environment. European Network and Information Security Agency (ENISA) developed and maintain a list of Cloud Computing vulnerabilities. [22]

4. Control Analysis:

In Control Analysis step, the system is studied to find out about the accuracy and effectiveness of security controls. These controls are designed to reduce threats impact or their execution probability.

Our main contribution in this thesis is to study existing control mechanisms and find their weaknesses. Then in Control Recommendation step we will propose our own approach for fulfilling uncovered security requirements. More information on this contribution can be found in Chapter 6.

5. Likelihood Determination:

The likelihood of exercising a vulnerability is calculated with respect to the capability of threat-source, vulnerability characteristics and applied security controls mechanisms. Likelihood can be *High*, *Medium* or *Low*.

6. Impact Analysis:

In Impact Analysis, a successful exercise of a threat is studied to determine its impact. These impacts are useful in our Detection and Analysis phase in Chapter 5; In order to study and recognize a compromised component, its impacts and side effects.

7. Risk Determination:

Risk is a function of threat exploitation likelihood, impact magnitude and effectiveness of security controls. The goal of this step is to study risk level based on its different parameters. Cloud specific risks are determined in Chapter 3 of [22].

8. Control Recommendation:

The goal of this step is to introduce controls that helps to reduce the likelihood of successful exercise of a vulnerability. We will introduce our mitigation and containment approach in Chapter 6, that are controls which are recommended by us to reduce the likelihood of compromising a component.

9. Results Documentation:

The outcome of this step is a risk assessment report that explains results of all eight first step, including: threats, vulnerabilities, risk likelihood, etc.

Complete security risk assessment of Cloud Computing Model is done by European Network and Information Security Agency (ENISA)[1] in the *Cloud Computing Security Risk Assessment Report*[22]. We will refer to this report as one of our major references in risk management of the cloud model.

2.3 Study Motivation

In order to motivate and justify our study, we divide the title and discuss the importance of each part separately. Importance of the Cloud Computing model was discussed in Section 1.1. Also, security of a cloud environment has been identified as the most challenging obstacle that hinders shifting to the new computing model, in Section 2.1. In the following parts, we explain why we chose the IaaS service model and also why we are interested in compromised components.

2.3.1 IaaS Security

Cloud consumers will lose strong control over their data when they shift to a cloud environment; this fact threatens their data security. In this new computing model, a security breach in the cloud infrastructure may affect all cloud consumers and cause a catastrophic failure [75].

Although many threats in a cloud environment have been identified in other fields previously, new characteristics of the cloud model introduces their own new security challenges. As an example, multi-tenancy and scalability are influential properties on security mechanisms of a cloud.

When we talk about threats to a system, they can be either the most *important* or the most *likely* threats. We focus more on the most important threats, despite they may not be likely.

IaaS security is important from different perspectives. The IaaS service model is one of the most used service models of Cloud Computing [61]. As it is written in Table 2.2, IaaS is the service model for almost half of all cloud environments. Additionally, it is used for business-critical services in 12.5 percent of cases in average. These statistics represent the importance of security mechanisms of a cloud environment in an IaaS service model.

Moreover, two other statistics are also notable in Table 2.2. First, according to the survey, in 34% cases of using IaaS, the cloud provider was identified as the most responsible stakeholder for the security of the system. It is interesting that the customers who are looking for IaaS services are more concerned about their security comparing to customers of other service models; thus they do the most security evaluation of cloud environment resources before deployment.

In another part of this survey [61], the most effective security technologies for a cloud environment are discussed. *Network Intelligence Systems* and *Virtual Private Networks* are identified as the most effective technologies. It is evident that these technologies should be applied in the infrastructure

of a cloud environment to make it attack resistant.

2.3.2 Security of a Compromised Component

A cloud environment is a distributed computing model. In such an environment a variety of components are distributed physically and geographically; they also communicate with each other to complete a task. These components are not essentially homogeneous or as secure as each other. Moreover, each of these components may get compromised. Thus it is important to identify a compromised component and understand its threat to other components in the environment. Finally, containing and recovery of such a component is crucial for long term availability of the cloud environment.

2.3. Study Motivation

SaaS, IaaS and PaaS Usage		
	Europe	US
SaaS	62%	67%
PaaS	33%	35%
IaaS	46%	53%
Business-critical applications		
	Europe	US
SaaS	16%	22%
PaaS	9%	13%
IaaS	11%	14%
Cloud provider is most responsible for security		
	Combined	
SaaS	42%	
PaaS	21%	
IaaS	34%	
Prior security evaluation of service model before deployment		
	Europe	US
SaaS	61%	45%
PaaS	52%	46%
IaaS	66%	51%
Most important technologies for securing a cloud environment		
Technology	Importance	
Network intelligence systems	64%	
Virtual Private Networks	64%	
Log management	62%	
Identity federation	51%	
Encryption for stored data	45%	
User management and provisioning	45%	

Table 2.2: Cloud Security Survey by CA Technologies [61]

Chapter 3

OpenStack

In our study we primarily focus on OpenStack as our cloud platform software. OpenStack uses open source software, technologies and open source standards to build a cloud infrastructure. It has three projects, OpenStack Compute, OpenStack Object Store and OpenStack Imaging Service. These projects are interrelated as depicted in Figure 3.1.

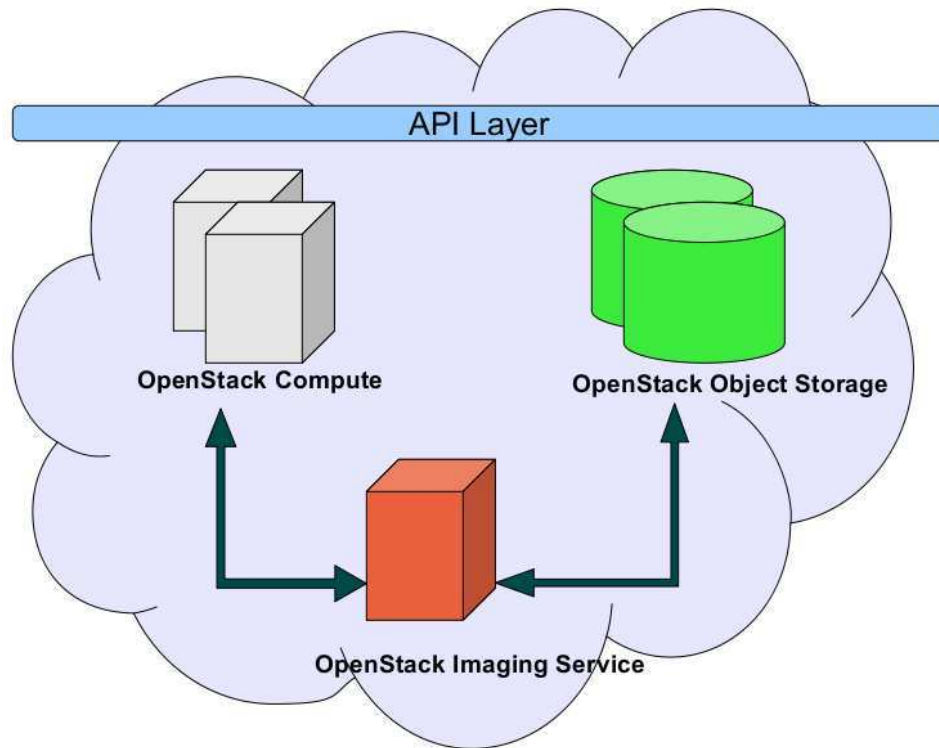


Figure 3.1: OpenStack projects and their relation [56]

More than 65 companies participate in the OpenStack [10] cloud platform. Among these companies, big names reside such as: NASA, RackSpace,

Citrix, Cisco, Dell, Intel, AMD. Many of them are using OpenStack as their cloud platform to provide other service models, including: Platform as a Service (PaaS), Software as a Service (SaaS), Network as a Service (NaaS), etc.

3.1 Architecture

The OpenStack compute project is based on two architectures: Shared Nothing (SN) and Message Oriented architecture. Most of the components can run on multiple machines. The internal communication among scheduler, network controller, volume controller, and compute controller is via AMQP [56]. In such a distributed environment messaging and asynchronous method calls are used to avoid deadlocks, livelocks and other common challenges.

Moreover, states are saved in a distributed data store to support the Shared Nothing (SN) architecture. Caching of these states for a limited period of time is also done to improve the environment performance and utilization [56].

In the initial design of the OpenStack two main architecture model has been introduced: hierarchical model and peer-to-peer model [38].

3.1.1 Hierarchical

The hierarchical architecture of an OpenStack deployment is similar to a set of DNS servers' structure. The cloud controller, that has the interface to cloud consumers, routes requests to an appropriate set of clusters [38]. An abstract drawing of this architecture is in Figure 3.2.

3.1.2 Peer to Peer

However, the peer-to-peer model is more like the model used for the interaction of a set of IRC servers. A zone received a request and may handle it or forward it to another zone [38], Figure 3.3.

3.1.3 Multiple Cluster Zones

Another architecture is also proposed in a blueprint [82]. This one focuses on a multi cluster architecture. Zones are defined to group OpenStack Compute services (i.e. VM instances host). Zones can be defined hierarchically in order to provide more flexibility for organizational structures, as shown in Figure 3.4. It is also feasible having multiple top level zones to support

3.1. Architecture

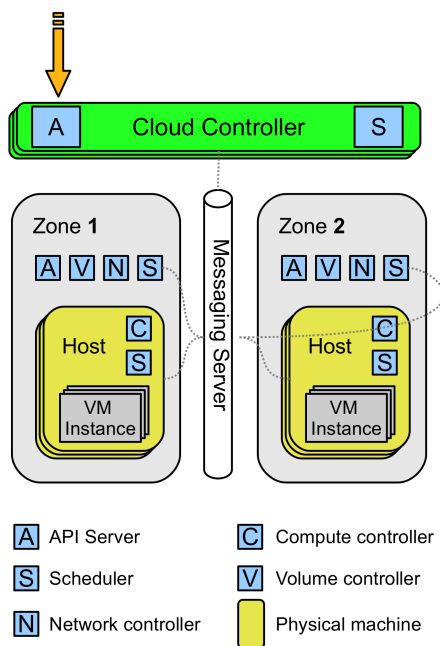


Figure 3.2: OpenStack Hierarchical Architecture[38]

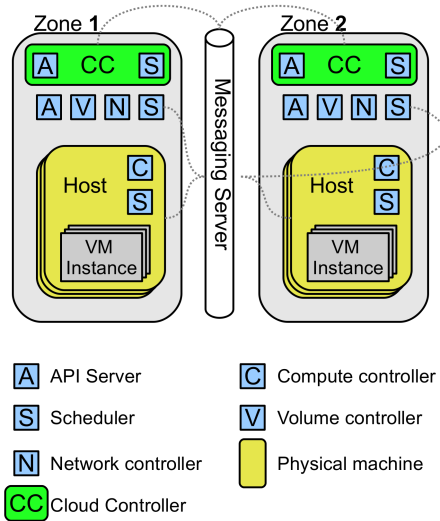


Figure 3.3: OpenStack Peer-to-Peer Architecture[38]

3.1. Architecture

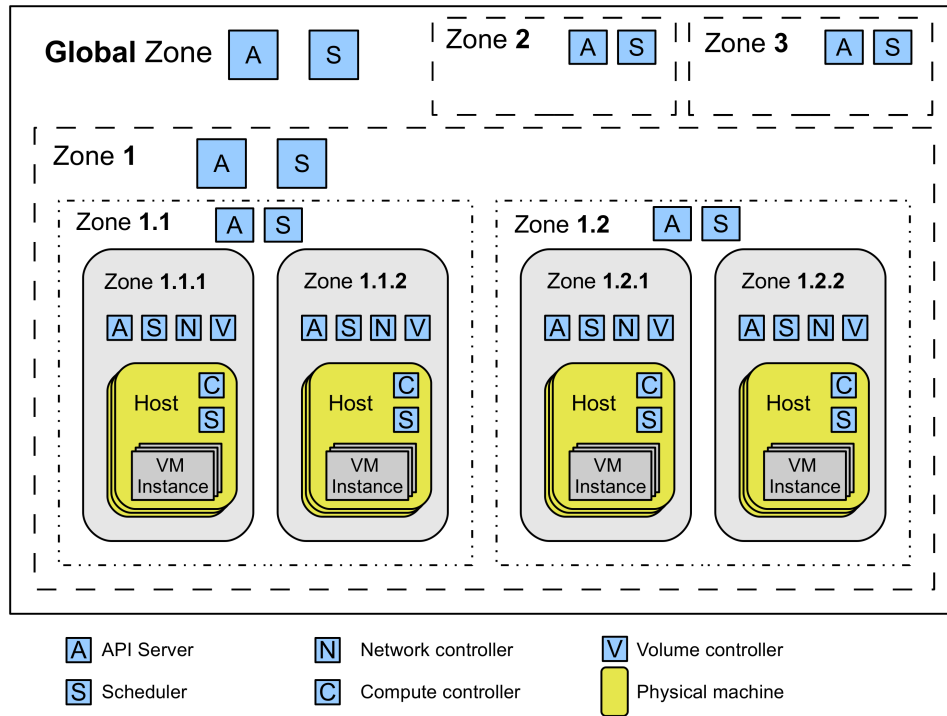


Figure 3.4: OpenStack Compute MultiCluster Zones[82]

different partitioning or organizational schemes, like: geographical or functional. The AMQP network is responsible for the communication among zones [82].

This approach is proposed to overcome scalability issues. It will provide partitioning functionality of hosts to facilitate large scale deployment, in the order of 1 million hosts and 60 million VM instances [82].

In MultiCluster Zones, the Scheduler service should route a request between Zones before the appropriate component picks up the request. It can be developed in several ways, as drawn in Figure 3.5 inspired from [82]. First, there can be a network of AMQP queues that are connected using schedulers in each zone. Zone schedulers are responsible for retrieving messages from their parent zone and put them in their local queues. A second method is to re-use the API server in each zone for forwarding the message to that specific zone [82].

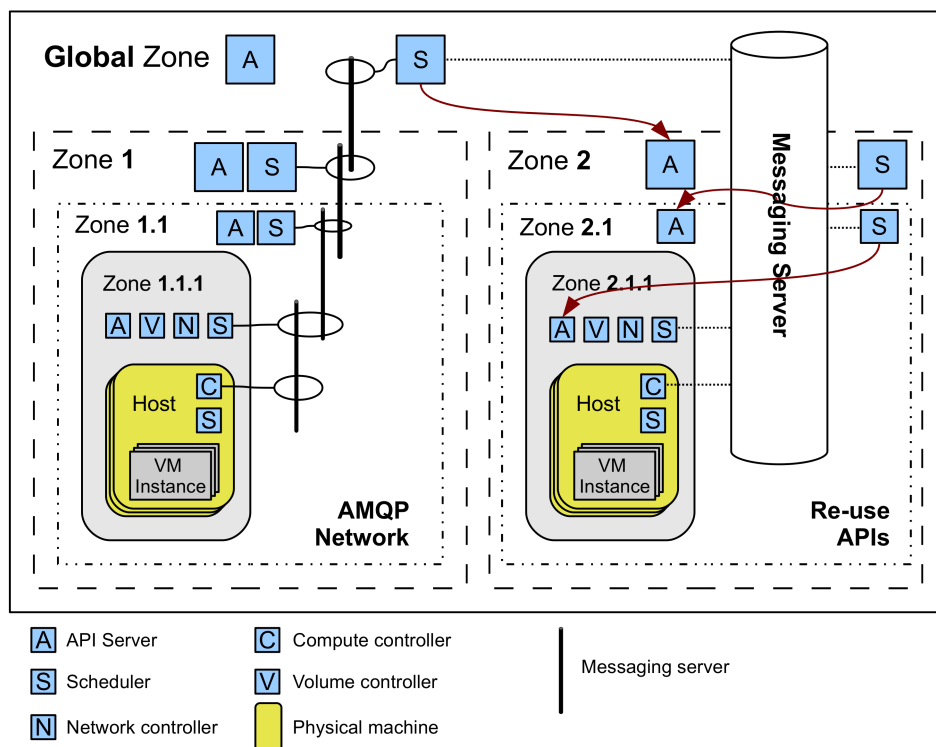


Figure 3.5: MultiCluster Zones Implementations[82]

3.2 OpenStack Compute Project (Nova)

Nova is the OpenStack compute project, providing an IaaS service model. OpenStack compute provides interfaces and utilities that interact with virtualization mechanisms, and is not a virtualization software. In this section we will explain basic concepts behind Nova. These details will help us to analyze different security perspectives of the OpenStack compute project. We will go through several concepts in Nova, including: administrative components (users and projects), supported virtualization mechanisms, system architecture, storage facilities, quotas, access control, supported third party interfaces, networking, Inter-process communication (IPC) and Remote Procedure Call and security groups.

Administrative components in Nova are simple. Images are part of each project and access to them is restricted based on corresponding project. Quotas are also defined per project. Each user has its own access and secret

3.2. OpenStack Compute Project (Nova)

credentials. Key-pairs which are used to manage images/instances are per user.

Nova supports several virtualization mechanisms, including: KVM, XEN, and User-mode Linux (UML).

In the following, main components of OpenStack will be introduced. It should be noted that, although these components are implemented by OpenStack specifically, same components with almost same use-cases can be identified in any other implementation of an IaaS service model platform.

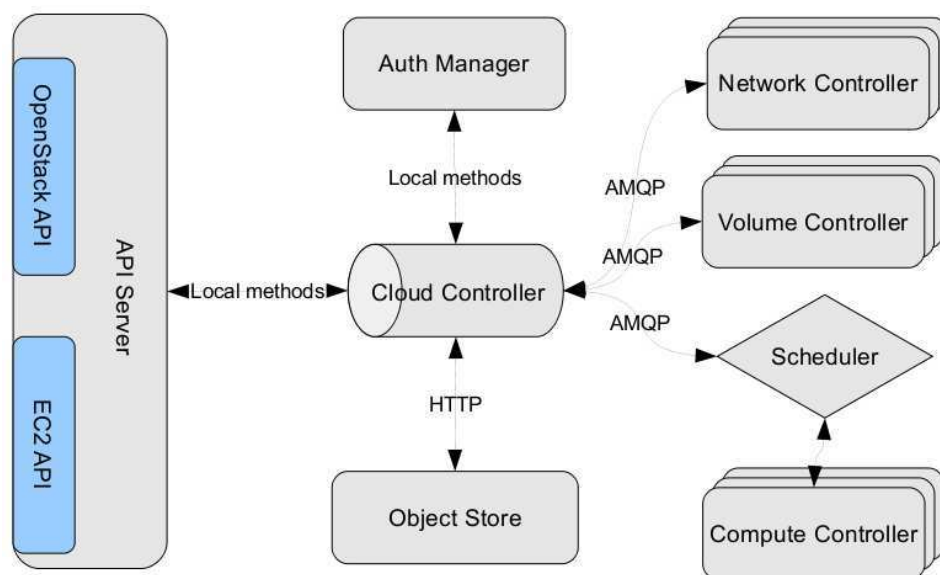


Figure 3.6: Nova components and their interaction[84]

3.2.1 Cloud Controller

The cloud controller is the global state of the system and communicates with all other components. In the Bexar release [31] of OpenStack, the cloud controller is a messaging server that handles the messaging between other components of the cloud platform. As all components communicate in an asynchronous fashion, having a messaging server is crucial. Currently RabbitMQ is the realization of the messaging server in OpenStack [85].

RabbitMQ is one of several implementations of AMQP. It is open-source and the server is written in Erlang [3]. It provides communication amongst all Nova components. The Publisher/Subscriber paradigm is used for com-

munication and on top of this paradigm, Remote Procedure Call (RPC) is built. This loosely coupled communication has several benefits, including:

- Communicating components are decoupled, which means they don't need to know each other's references.
- Full asynchronous communication
- Load balancing of RPC.

Both `rpc.call` and `rpc.cast` types of RPC are implemented over AMQP. Nova has adapters that are responsible for marshaling and unmarshaling of messages into proper RPCs. Each Nova component has two separate queues. The first queue only accepts messages with a specific form of key. The key pattern should be as `{NODE-TYPE.NODE-ID}`, `NODE-TYPE` can be `COMPUTE`, `NETWORK`, etc; `NODE-ID` is any identifier that can uniquely point to a specific node; This queue is used by the `rpc.call` operation. The second queue has less restriction and accepts messages with the `{NODE-TYPE}` pattern, this one is used when the `rpc.cast` command is executed. When a command must be redirected to a specific host (e.g. VM instance termination), the first type of queue is useful otherwise, using second queue is useful in more generic cases. [12]

Two roles are involved in using a queue, an Invoker and a Worker. An Invoker is the sender of a message to the queue; It can be done by either `rpc.call` or `rpc.cast`. Contrary, a Worker listens to the queue and receives messages, also replies to `rpc.call` messages. RabbitMQ internals is quite simple. Several players can be identified in this queuing system, including: Topic Publisher, Topic Consumer, Topic Exchange, Direct Consumer, Direct Publisher, Direct Exchange and Queue Element. The communication is done in two different modes: [12]

- **Topic**

When `rpc.call` or `rpc.cast` is executed, a Topic Publisher is created and pushes the corresponding message to the queue. This publisher is connected to a particular Topic Exchange until the message delivery, then it will be destroyed. When an instance of a Worker is created, a Topic Consumer is also instantiated for that specific Worker. It will listen to the queue and receive the message and behave based on what is defined in the Worker. This Topic Consumer is also connected to the same Topic Exchange. Each Worker has two Topic Consumer which are responsible for `{NODE-TYPE.NODE-ID}` and `{NODE-TYPE}` queues.

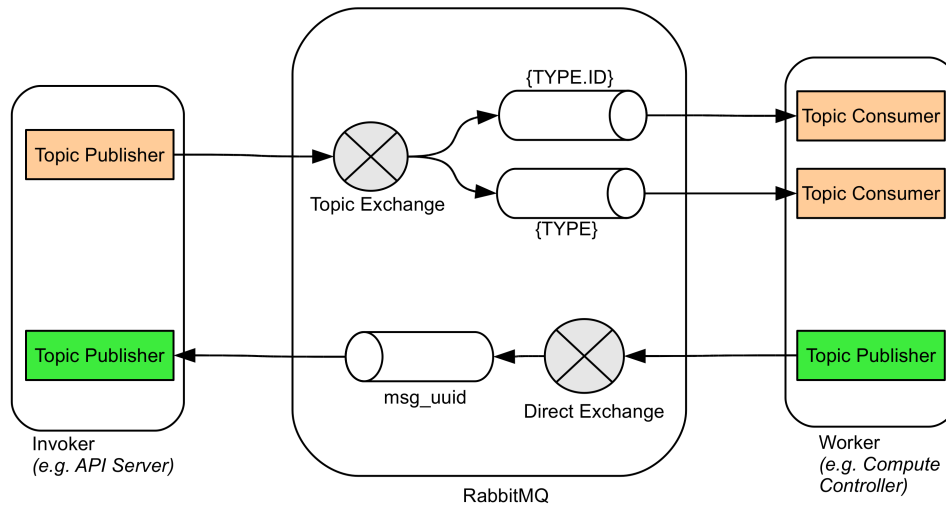


Figure 3.7: RabbitMQ Internals and two modes[12]

- **Direct**

Direct Publisher, Direct Consumer and Direct Exchange are main players of this mode. When a `rpc.call` operation is executed, an instance of Direct Consumer is created. The Direct Consumer is connected to a specific Direct Exchange using a queue. This is not a share queue. Direct Exchange and the unique queue are identified by their Universally Unique Identifier (UUID). Direct Publisher uses this identifier in each message. When an `rpc.call` is executed, a Direct Publisher is instantiated to provide the response for Direct Consumer. The incoming message has the UUID and Direct Publisher will use that identifier for further communication. Each `rpc.call` has its own Direct Exchange, and it works as a routing table.

The basic operation of the `rpc.call` and the `rpc.cast` is very simple. In case of a `rpc.call`, a Topic Publisher is instantiated and it will send the request to Topic Exchange. At the same time a Direct Consumer is created to receive the response when it is ready. At the other end, Worker side, one of those Topic Consumers will receive the request and forward it to the Worker. Upon finishing the task, the Worker will send back the response using a Direct Publisher. Again at the Invoker side, instantiated Direct Consumer will receive the corresponding response.

While executing `rpc.cast`, the same procedure is done except sending back the response. Thus in this case there is no need for Direct Publisher,

Direct Exchange and Direct Consumer.

In the current deployment of OpenStack in our lab setup only one Cloud Controller exists (See Chapter 8). As a result, only one RabbitMQ (Messaging Server) handles all communication among Invokers and Workers. Thus RabbitMQ becomes a single point of failure, and failing of this component will lead to a failure in the whole cloud platform. Two approaches have been proposed to solve these issues, RabbitMQ active/passive setup and RabbitMQ Clustering. These approaches provide high resiliency and high availability respectively [50]. We will study these mechanisms in Chapter 6.

The Communication between RabbitMQ and OpenStack Compute is provided using the Carrot library [41].

Functionality

The cloud controller handles the interaction between most of other component, comprising: compute controller, volume controllers, network controllers, API server, schedulers. Users' communication with the cloud controller is through the API server by means of Hypertext Transfer Protocol (HTTP).

The cloud controller routes an arrived request to a queuing engine that is for a relevant group of workers. Workers in a specific group listen to their own queues for new requests. After they get their requests and perform the corresponding tasks, they send the response to the cloud controller and it will send the response back to the originating user through the API Server.

The cloud controller uses the publisher/subscriber Paradigm. On top of this paradigm, Nova components use RPC to communicate with each other [12]. RabbitMQ[3] is used as the implementation of messaging server. Having a single instance of the cloud controller makes it a single point of failure and a risky bottleneck in the deployed cloud environment. Thus, in order to avoid these challenges different features of RabbitMQ can be employed (RabbitMQ Active/Passive, RabbitMQ Clustering) [50]. These features will be studied in Chapter 6 where handling, mitigation and containment procedures are explained.

The cloud controller functionalities can be enumerated as follows:

1. Queue management for inter-component messaging
2. Message routing and delivery for Cloud platform components

Connected Components

The cloud controller is at the heart of cloud platform and due to its functionalities, it is connected to most other components in the platform; comprising:

1. Network Controller
2. Volume Controller
3. Compute Controller
4. Object Store
5. Auth Manager
6. Scheduler
7. API Server

Access Method

In the cloud controller most communication with other components is done by means of the AMQP messages. In rare cases HTTP and Local methods are also used. The cloud controller retrieves VM images from Object Store using HTTP commands. Additionally, the API server and the Auth Manager communications are handled by local methods.

3.2.2 Object Store (nova-objectstore)

The object store is a HTTP server that provides storage and retrieval services for VM images. It is a simple file-based storage that has most of S3 interfaces. OpenStack Imaging Service with an image manager can be used instead of it. The other OpenStack project, OpenStack Object Storage, also provides image storage functionality [85].

3.2.3 Auth Manager

This component is responsible for managing users, projects and roles; It can also be connected to a database or LDAP server. It implements authentication and authorization functionalities.

3.2.4 Volume Controller (nova-volume)

The volume controller is responsible for handling dynamically attachable volumes. A volume is a detachable block storage device (e.g. USB Hard disk) that can be connected to only a single instance at each time.

Functionality

The volume controller is responsible for creating, deleting, attaching, detaching and persisting block device storage. These kinds of storage is independent of corresponding instances. They can be attached/detached to/from any instance even with different VM images (i.e. operating system types or configuration). Moreover, this controller sets up a remote volume on a compute host and removes the volume when necessary.

Connected Components

The volume controller is connected to the cloud controller and uses AMQP to communicate with it.

Access Method

Like many other components the only way to access the volume controller is to use AMQP messages.

3.2.5 Network Controller (nova-network)

By providing virtual networks, a network controller enables the communication among compute servers and also with the public network. (i.e. managing IP forwarding, bridges, VLANs, Fixed/floating IPs and DHCP)

Functionality

The network controller is responsible for IP address allocation and network configuration. A detailed overview of Network Controller functionalities is described in the following list:

1. Managing fixed IP addresses
 - Allocates a fixed IP
 - Deallocates a fixed IP
 - Leases a fixed IP (Called by DHCP-Bridge)
 - Releases a fixed IP (Called by DHCP-Bridge)
2. Managing floating IP addresses
 - Allocates a floating IP (Gets it from the pool)
 - Associates a floating IP to a fixed IP

3.2. OpenStack Compute Project (Nova)

- Disassociates a floating IP from a fixed IP
 - Deallocates a floating IP (Returns it to the pool)
3. Creates virtual networks
 4. Configures bridges, VLANs and forwarding rules

Connected Components

The network controller is connected to the Cloud Controller in the current architecture of OpenStack.

Access Method

As this component is only connected to the Cloud Controller, the only access method which exists is AMQP messages.

3.2.6 Scheduler (nova-scheduler)

The scheduler select the host that the new instance and volume should run on. This decision can be made using different algorithms that focuses on finding the most suitable compute controller [85]. The scheduler is also responsible for routing requests between components/services. In other words, it is responsible for forwarding requests to the proper services [82].

The scheduler is one of the most important components of OpenStack Compute. In a large scale ⁴ deployment of OpenStack, a reliable and scalable Scheduler is required. It will be achieved in several ways. First, the scheduler can be used for containment of a compromised component. Second, it should be configured to tolerate components failures that may lead to network partitioning, node failure, higher load etc. Therefore, in a large scale deployment it is not plausible having a single scheduler. To resolve this issue a distributed scheduler is proposed [34].

Functionality

The scheduler is responsible for routing messages which are delivered to the messaging server. Choosing the appropriate compute controller, volume controller and network controller, it will send the user's request to them.

- Choosing worker nodes

⁴By large scale we may mean one million hosts and 60 million virtual machines

- Routing users' requests
- Reacting to incidents take place in components (e.g. component failures or security breach)

Connected Components

As it was explained, the scheduler is connected to all of controllers. Also the scheduler should talk to the API server using a messaging server (i.e. the cloud controller).

Access Method

The scheduler is working in a distributed environment, as a crucial requirement for scalability and reliability, it communicates with other components using AMQP messages.

3.2.7 Compute Controller (nova-compute)

The compute controller manages communications between virtual machines and the hypervisor. It exposes resources from the compute server to virtual machines [85]. The compute controller is one of the most important components of OpenStack Nova. It is responsible for managing VM instances.

Basic responsibilities of a compute controller can be itemized as follow: [56]

- Run, Terminate and Reboot VM instances
- Attach and detach volumes
- Get console output

Moreover, it is possible to run multiple instances of a compute controller on several physically distributed machines.

A compute controller may communicate with several other components. Identifying these components and their communication specification is useful for further analysis of security threats to each component.

Functionality

The compute controller main functionality is related to VM instances and their volumes. The compute controller start, stop and reboot instances; also

it handles the attachment and detachment of volumes from instances. It can also provide the console output of a specific instance for debug purposes. More detailed version of some of high level functions are as follows:

- Run an instance
 1. Check if the instance is already running.
 2. Allocate fixed IP addresses.
 3. Setup networking facilities (i.e. VLAN, Bridge).
 4. Check compliance of the instance availability zone with the current running zone.
 5. Spawn the instance.
- Terminate an instance
 1. Disassociating its IP address
 2. Deallocating its IP address.
 3. Detaching volumes
 4. Destroying the instance
- Reboot an instance
 1. Reinitializing network configuration
 2. Rebooting the instance

Connected Components

Each instance of a compute controller is connected to a specific scheduler and a specific cloud controller. Also it communicates with VM instances that are running over its platform.

Access Method

Main communication method among a cloud controller, a scheduler and a compute controller is AMQP messages. Moreover, the compute controller uses hypervisor interfaces to communicate with its own VM instances.

3.2.8 API Server (**nova-api**)

The API server is responsible for receiving HTTP requests and in general converting commands and communicating with other modules using the AMQP and HTTP. [85] In other words, it is a web services front-end for the cloud controller. The API Server supports both Amazon and Rackspace interfaces.

One of the main design criteria behind OpenStack is a shared-nothing, messaging-based architecture. It will allow major components to run on multiple servers. By major components we mean **Compute Controller**, **Volume Controller**, **Network Controller** and **Object Store**.

3.2.9 Compute Interfaces

Django-Nova and OpenStack-Dashboard are web-based consoles for the OpenStack Compute project.

3.2.10 RBAC model in OpenStack

Roles define users' privileges and restrict their activities. In this model permissions are not assigned to users directly, instead users have some roles that provide them with the appropriate privileges. Users' effective privileges are derived from the intersection of user roles and a specific project roles. Basic design entities of Role-based Access Control (RBAC) deployment in OpenStack include the follows:

- Roles limit users' access.
- Projects limit users' access to a particular image.
- Resource consumption is limited based on each project

OpenStack has five predefined roles:

- **Cloud Administrator (*admin*)**: This role has complete access to different components in the system.
- **IT Security (*itsec*)**: Users with this role can quarantine instances.
- **Project Manager (*projectmanager*)**: Users with this role can manage users, images and instances of a project.
- **Network Administrator (*netadmin*)**: Users with netadmin role can manipulate public IP addresses and firewall rules.

- **Developer (*developer*):** This is the default role that is assigned to users.

3.2.11 Operation

As depicted in Figure 3.8, most components in the OpenStack Compute (Nova) project communicate together using AMQP. RabbitMQ[3] is currently used as the messaging server and each component has its own queue. These queues will be used for sending messages in. Each component provides a service that has a specific Service API Stub. Public APIs use these Service APIs to forward the users' request to the appropriate queue, thus responsible service will eventually handle the request. Authentication is done before an end user can communicate with public APIs; In this step, the Auth Service decides that the user's client can use which API service [82].

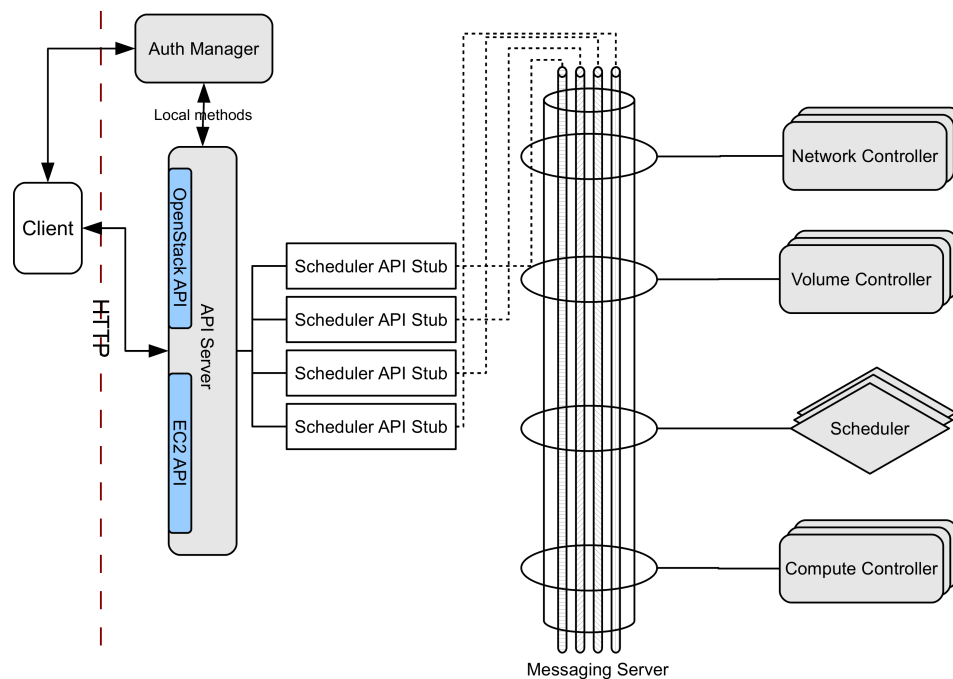


Figure 3.8: OpenStack Compute basic architecture [82]

3.3 OpenStack Object Storage (Swift)

Swift is the Object Storage project of the OpenStack. It is a highly available, consistent and distributed object store that is used for efficient and safe storage of large amount of data. OpenStack Object Storage project has variety of use-cases, like archiving data, storing multimedia content, providing the Cloud Computing elasticity and on-demand storage access.

Although we do not focus on the Swift project, we mention their components briefly to also become familiar with the rest of an OpenStack cloud environment.

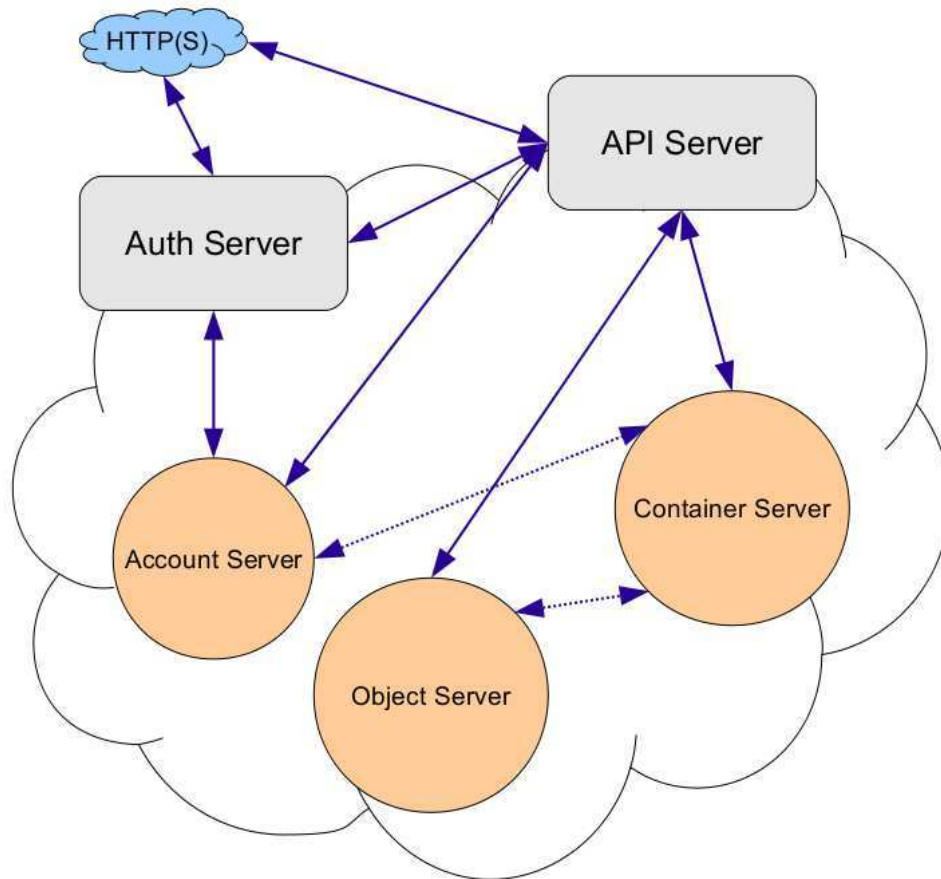


Figure 3.9: Swift components and their interaction[71]

3.3.1 Components

Proxy Server

The proxy server routes each request to the proper handler. In order to do this routing it has a looking up mechanism that searches for the location of the corresponding account, container, or object. It also exposes public interfaces. All objects are streamed through the proxy server and it can handle failures to a certain degree.

The Ring

A ring provides the exact physical location of an entity. Accounts, containers and objects has their own rings and other components interact with the specific ring to find out about the exact location of that object, container or account in the cluster.

Object Server

The Object Server can store, retrieve and delete objects from local devices. In other words, it is a local BLOB storage server. Binary form of objects will be stored on the filesystem. A hash value of the object name and the timestamp for the last operation are used to generate the object path.

Container Server

The container server is responsible for managing listing of objects. It stores list of objects in a specific container. These lists are also replicated across the cluster. Statistical information like total number of objects and total storage usage for the specific container are also stored by this server.

Account Server

It has same functionality as Container Server but handles lists of containers instead of objects in a specific container.

3.4 OpenStack Imaging Service (Glance)

The imaging service provides functionalities that facilitate dealing with virtual machine images. It has lookup and retrieval operations and can be integrated with different storage provides, including: OpenStack Object Store, Amazon S3 storage, S3 storage with Swift as the intermediate. [56]

3.5 Other Components

Project VPN (CloudPipe)

Cloudpipe provides a connection for end users to access their project instances in the VLAN networking mode. Cloudpipe uses Nova administrative commands to automatically create an instance for a given project. This instance provides a VPN service for end users so they can be connected to the private network of their projects. Users will have a secure access to their instances without exposing them to the public Internet.

The cloudpipe image is a simple GNU/Linux instance with OpenVPN on it. When the cloudpipe is launched for a user the following procedure will be done:

1. VPN keypair, for that specific project, is created and saved in the keys directory.
2. A new security group is created, port 1194 and ICMP packets are allowed in it.
3. Other credentials (certificate and private key) for the VPN instance is created and saved in CA/projects/{project_id}
4. All the information is zipped and encoded as base-64
5. An m1.tiny instance is launched and the given information is inserted in it. (specific flag is also used to specify VPN image)

3.6 Networking

In OpenStack Compute, cloud resources are organized in projects. VM instances are grouped in a Compute project. Each instance has its own private IP address. Linux bridging, as defined below, is used to provide connectivity between virtual interfaces among VM instances and physical interface with outside. Currently, only Linux bridge networking is available in Nova [56].

As defined in Linux-HowTo [81],

”A bridge is a device that separates two or more network segments within one logical network (e.g. a single IP-subnet).”

A bridge is placed between groups of machines that may talk to each other but not that frequently. The bridge checks the destination of a data packet and either passes it to the other side of the Ethernet segment or handles it locally. The bridging decisions are made independent of higher layer protocol types (IP, IPX, NetBEUI) and are dependent on Media Access Control (MAC) addresses of each Network Interface Controller (NIC). It should be noted that a bridge is somehow like a Layer 2 switch, which is transparent to other nodes in the network [81].

Nova has three types of network configuration, comprising: Flat, Flat-DHCP and VLAN mode. They can exist simultaneously in a cloud environment. Moreover, each instance in a Nova can have either Fixed IPs or Floating IPs. As their names explain, Fixed IPs cannot be released from or assigned to a VM instance before its termination, but Floating IPs are more flexible and can be released from an instance and assigned to another one at any time [56].

It is important to think about separating Host Identity (HI) from Host IP address; Floating IPs are useful in this approach. Host Identity Protocol (HIP) provides a method for separating host identify from its location (IP address). It uses public keys of a host to generate its HI [52]. Using HIP will also be useful for handling and containment of a compromised host when it is identified. Security mechanisms can simply treat that specific compromised host by means of its identity, without disturbing other nodes functionality.

Next, we will go through different network configuration modes briefly.

- **Flat Mode**

Instances are assigned fixed IP addresses from a specific subnet. This subnet is previously specified by network administrators. Machines with *network controller* and *cloud controller* services should have networking bridged. Bridge configuration should be done manually by network administrators.

The FlatManager in the Flat mode does not have any responsibility about bridge or VLAN handling. The administrator is responsible for creating bridges on all Compute Controller nodes. It is possible to force Compute Controller to inject network configuration to a VM instance.

- **Flat DHCP Mode**

In this configuration mode, IP addresses are assigned by means of a Dynamic Host Configuration Protocol (DHCP) server. Similar to Flat Mode, the bridge on the compute controller node connects all the

instances in that specific node together. This bridge also provides the connectivity among instances on this compute node and other nodes.

FlatDHCPManager starts up a DHCP server to assign IP addresses. In this mode network configurations are not injected into the VM instance.

- **VLAN Mode**

This mode is the default mode for OpenStack Compute. In this configuration, each project has its own VLAN, bridge and IP addresses subnet. Instances in a specific project are accessible only from the corresponding VLAN. Cloudpipe is a specific purpose VM instance that provide connectivity for users which are outside of the VLAN project. A DHCP server handles IP assignment to each instances of a given project. OpenStack Compute is responsible for managing all these VLANs, bridges and running/termination of Cloudpipe instances. More details about Cloudpipe can be found in Section 3.5.

It should be noted that in VLAN mode, if we have more than one compute controller, corresponding machines must be connected to each other using a switch that supports host-managed VLAN tagging.

Chapter 4

Components at Risk

According to the definition by Committee on National Security Systems (CNSS)[15], **compromise** means [2]:

”Disclosure of information to unauthorized persons, or a violation of the security policy of a system in which unauthorized intentional or unintentional disclosure, modification, destruction, or loss of an object may have occurred.”

When we talk about a compromised component in this document, we mean those components in a cloud environment that are disclosed, modified, destroyed or even lost. Finding compromised components and identifying their impacts on a cloud environment is crucial. It is useful in risk management, specifically in System Characterization and Impact Analysis steps of assessment methodology, you may find more details on risk management in Section 2.2. Additionally, it will help stakeholders plan appropriate incident handling strategies for their cloud environment in case of facing a compromised component. These components can be placed in several layers of cloud stack, as depicted in Figure 4.1.

We may identify three major elements amongst cloud components, namely Cloud node, Network equipment and Specific purposes host. More details about a cloud node is introduced, there exists four main layers before we reach to VM instances on the top⁵. A component can be compromised in any of those elements. There has been extensive research on compromising hardwares, operating systems, variety of services running above operating systems, hypervisors and virtualization technologies. As we discussed previously, Cloud Computing is a new computing model, thus we focus on those components in a cloud platform to identify new challenges and weaknesses.

In the first part of this chapter major components of a cloud environment will be studied. In this part after identifying components, their importance

⁵It should be noted that we avoided introducing more complexity by omitting cross layer coupling, e.g. most of virtualization technologies are tightly coupled with the host operating system.

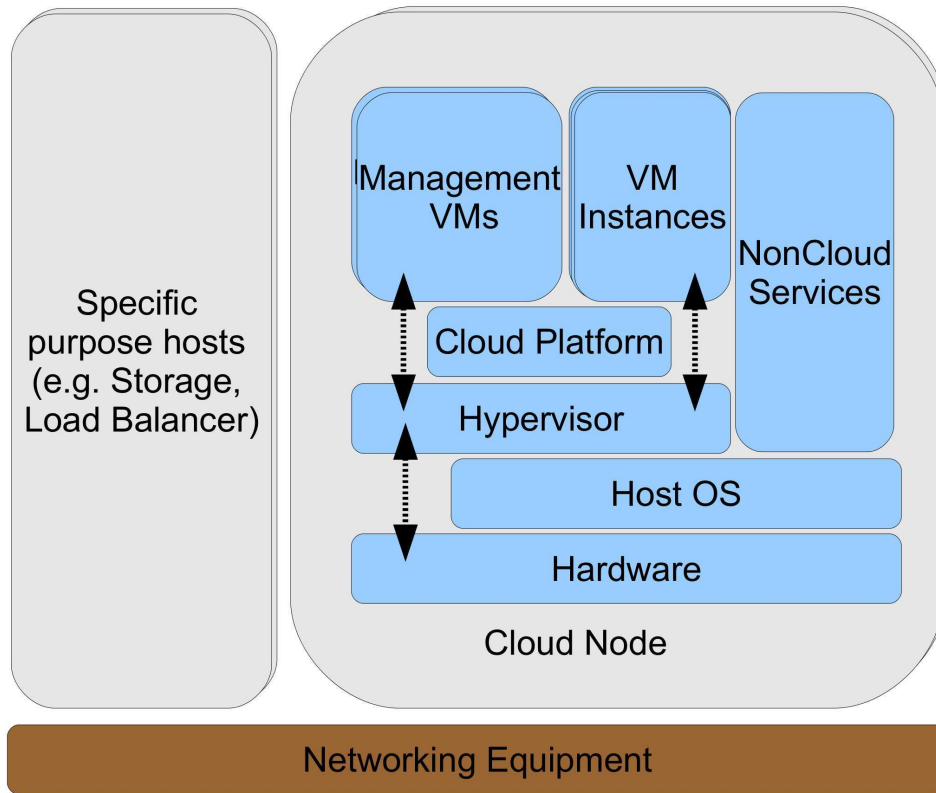


Figure 4.1: Cloud Components

and roles in the cloud model will be discussed. Additionally, components relations, interactions and their subcomponents will also be reviewed.

As Cloud Computing is a new model, little research has been done on its components' security, previously. Therefore, we will focus more on new components of a cloud platform in this chapter.

In the second part, components characteristics will be studied from a security perspective. We will go through functionalities, connected components and access methods of each component. Then impacts of compromising those components will be introduced.

4.1 Virtualization Vulnerabilities⁶

Virtualization is one of the crucial specifications of the Cloud Computing. According to a 2010 IDC Report [62], more servers were virtualized during last four months of 2009 comparing to the same period in 2008. Obviously, Cloud Computing plays an important role in this demand. The IBM X-Force® team study virtualization vulnerabilities from different perspectives in a 2010 Report [64]. Some of these viewpoints are important for our research, and we will discuss them in the following:

As it was expected, the number of disclosed vulnerabilities has increased during the past few years, till 2008. In the period of 2008-2009, this number has decreased. This trend can be due to vendors' attention to security aspects of virtualization.

During a 10-year period (1999-2009), most of disclosed vulnerabilities had high severity. Impacts of exploiting a severe vulnerability may not be tolerable in a cloud environment.

4.1.1 Vulnerabilities in the code

It is important to localize the vulnerable code. Localization helps to identify responsible stakeholder(s) faster and the containment procedure will happen smoothly. The IBM report [64] divided vendors into two groups, Virtualization software vendors and Third-party software vendors. From 2006 third party components have more vulnerabilities comparing to the other type.

As a result, virtualization software developers and providers should also be concerned about third parties possible vulnerabilities. Moreover, virtualization and in general, cloud platforms should have mechanisms for mitigating threats that are caused by third-party components.

According to the report [64], virtualization products can be either Server or Workstation type. Server products are software where the operating system and the hypervisor are merged, this type is also called bare metal virtualization. On the other hand, workstation products are installed on the host operating system, so they are separated.

This definition is important because the trend of vulnerability location differs in each type. In workstation products, third-party components have less vulnerabilities than virtualization vendors' components. In server products however, third-party components are much more vulnerable (i.e. only 30 percent of vulnerabilities happen in virtualization components). It should

⁶This section is based on an IBM report [64] that focused on vulnerabilities in the virtualization software.

be noted that in a cloud environment, server products are used as virtualization product in most cases. The reason is obviously because of the better performance.

In the period 2005-2009, the number of disclosed server vulnerabilities exceeded workstation vulnerabilities. Server products complexity and importance can justify these statistics.

The exact numbers and statistics related to products vulnerabilities are not available publicly. We tried to communicate with the report authors to obtain those statistics but it was not successful.

4.1.2 Types of vulnerabilities

Eight types of virtualization vulnerabilities have been identified by IBM [64]. We will go through those vulnerabilities that are important in our study. As shown in Table 4.1, each vulnerability type and its corresponding occurrence percentage are given for both workstation and server products.

- **Host:** Vulnerabilities that threaten the *operating system* of the host machine.
- **Guest:** Vulnerabilities that affect virtual machines that are running on a hypervisor. The host OS and hypervisor remain safe. These vulnerabilities have the same nature as host vulnerabilities in a non-virtualized system.
- **Escape to host:** Vulnerabilities that can violate virtualization isolation. These vulnerabilities let an attacker gain access to the host machine from virtual machines running on it. This type of vulnerability also affects the risk profile of the hosting machine. As a result, virtual machine vulnerabilities will be included in host vulnerabilities if an escape-to-host vulnerability exists.
- **Escape to hypervisor:** Vulnerabilities that violate the isolation by letting an attacker to gain access to other virtual machines or the hypervisor itself.
- **Virtualization system:** Vulnerabilities that threaten the virtualization system itself and may not reach to host machine. They pose the same type of risk as host vulnerabilities, so they may threaten guest machines as well.

- **Web application:** Vulnerabilities that exist in host or virtualization system applications which are used for management and control of the virtualization infrastructure.

Although this type of vulnerabilities has been identified as a separate group by IBM, we prefer to count them as either Host or Virtualization System vulnerabilities.

There are some noteworthy points in Table 4.1. First, the corresponding percentage for Host and Escape-to-host vulnerabilities in server products are zeros. On the other hand, each of Escape-to-hypervisor and Virtualization System's percentage are more than 30.0%. This can be justified using the fact that the hypervisor and the host operating system is the same component in a server product.

Second, almost 75% of vulnerabilities in Server products are related to hypervisor and virtualization system. These statistics clearly emphasize the importance of host security hardening and its catastrophic impact in case of a failure.

Third, in this report *cloud platforms* are not noted individually; We may safely assume the cloud platform as a crucial component of the Virtualization System in the Cloud model. Due to the high vulnerability percentage in the virtualization system and consequently in the cloud platform, risk profile is higher in those components⁷. Higher risk profile and less previous research on cloud platform vulnerabilities made us to focus on this area.

4.1.3 Hypervisor

Hypervisor is the virtualization component that manages the guest OSs on a host and controls the flow of instructions between the guest OSs and the physical hardware. [44]

Several hypervisors with different specifications exist. They may differ in functionalities and features. As an example UML does not use virtualization technology.

In a secure deployment of a cloud platform, understanding security characteristics of the utilized hypervisor is crucial. Several hypervisors are widely used in a cloud environment: Xen, KVM, QEMU, and Microsoft Hyper-v.

⁷We should also consider the existence of threat-sources and their desire for exploiting cloud platforms vulnerabilities; See Section 2.2.1 for more details.

4.2. Cloud Platform (OpenStack)

Type	Stakeholder(s)	Workstation Percentage	Server Percentage
Host	Cloud Provider	30.8%	0%
Guest	Cloud Provider, Cloud Consumer	26.3%	15.0%
Escape to host	Cloud Provider	24.1%	0%
Escape to hypervisor	Cloud Provider	3.8%	35.0%
Virtualization System	Cloud Provider	4.5%	37.5%
Web Application	Cloud Provider, Cloud Consumer, Service Customer	9.8%	10%

Table 4.1: Virtualization Vulnerabilities [64]

Analyzing security of hypervisors is out of our thesis scope, moreover there has been large number of attempts in assessing and securing hypervisors. As an example Murray et al. [53] has introduced an approach for securing the Xen hypervisor using disaggregation. Security has been achieved by moving the domain builder into a minimal trusted compartment. The domain builder has high privileges in a hypervisor, thus making it smaller will reduce trusted computing base.

4.2 Cloud Platform (OpenStack)

This section will introduce some specifications of components in an IaaS service model of the OpenStack cloud platform. Knowing these specifications let us identify impacts of exploiting a vulnerability in a component.

Most of the OpenStack components can run on the same machine or be distributed over several machines. Initially, we will spread out processes in a realistic manner. By realistic manner we mean that components are distributed as it should be done in a large scale deployment of a cloud environment. Adding more complexity is avoided in this stage.

Specific characteristics of each component will be introduced, including: functionalities, access methods, connected components⁸, etc. Identifying connected components is crucial in order to explain the impact and possible threats of a compromised component on the environment. Then, impacts of different incidents will be analyzed.

⁸We assumed all components as nodes in a graph and if they communicate with each other they are connected in the graph

4.2. Cloud Platform (OpenStack)

Code	Description
CC	Cloud Controller
NC	Network Controller
VC	Volume Controller
CoC	Compute Controller
OS	Object Store
AM	Auth Manager
Sc	Scheduler
VMI	Virtual Machine Instance
AS	API Server
MSG	Messaging
HTTP	Hypertext Transfer Protocol
LM	Local methods

Table 4.2: Codes

It should be noted that although we focus on the OpenStack as a specific cloud software in our study, more or less same components and processes may be identified in other cloud platform implementations.

Based on OpenStack references [56], [85], [84], [57] and its source code, following components can be spread out on different boxes in the deployment of the OpenStack Compute project (Nova):

- Cloud Controller (Messaging Server)
- Object Store
- Scheduler
- Volume Controller
- Network Controller
- Compute Controller

We will not study the Object Store component here, because it is a simpler form of Imaging Service and is not utilized in a large scale deployment.

Abbreviations/codes used in this section are listed in the Table 4.2. They will be reused in the following chapters.

We redraw the Figure 3.8 in Figure 4.2 to ease understanding of this section.

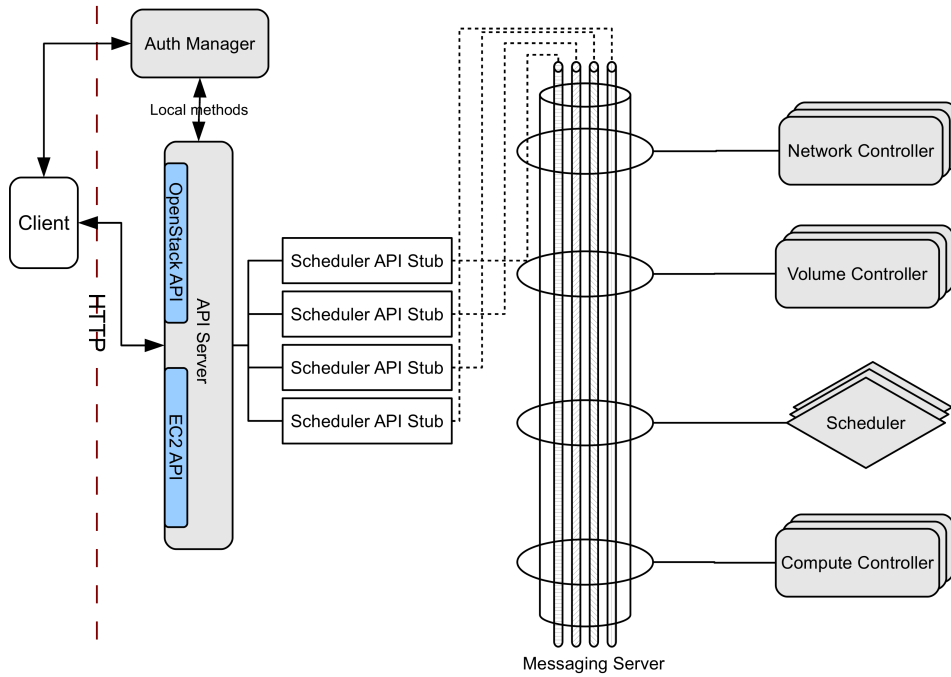


Figure 4.2: OpenStack Compute basic architecture [82]

4.2.1 Cloud Controller

The cloud controller is connected to most of the cloud components and is responsible for providing a messaging service to the platform. As we discussed before, the cloud controller is a single point of failure in the hierarchical architecture of the OpenStack. Including other specifications, connected components and access methods help us to identify several vulnerabilities in this part:

1. Impacts of DoS attacks can lead to a catastrophic failure in the cloud platform.
2. A compromised cloud controller⁹ threaten a variety of information security components, including:

⁹Compromised cloud controller is an ambiguous phrase. In this section, by compromised cloud controller we mean that the cloud controller component has some vulnerabilities which have been exploited. Exploited vulnerabilities may provide attackers with higher privileges in the Operating System layer. We consider worst case scenario in our study to overcome this issue.

4.2. Cloud Platform (OpenStack)

Functionality	
CC.F.1	Queue Management
CC.F.2	Message Routing
Connected Components	
NC	Network Controller
VC	Volume Controller
CoC	Compute Controller
OS	Object Store
AM	Auth Manager
Sc	Scheduler
AS	API Server
Access Methods	
MSG	Messaging
HTTP	HTTP
LM	Local methods
Impacts	
CC.I.1	Decrease in availability
CC.I.2	Catastrophic failure in case of a successful Denial of Service (DoS)
CC.I.3	Threaten message confidentiality
CC.I.4	Threaten message integrity
CC.I.5	Disturb cloud platform functionalities

Table 4.3: Cloud Controller Specifications

- **Message Confidentiality:** An attacker that resides in a compromised Cloud Controller can access the queuing system, and check the content of messages in the messaging server. End-to-end encryption mechanisms can provide message confidentiality for messages in transit.
- **Message Integrity:** An attacker can alter a message in delivery phase. Signing messages is a good technique to identify altered messages.
- **Service Availability:** An attacker simply decreases system availability by corrupting messages. Redundant Cloud Controllers and peer-to-peer architecture may improve availability. Adding redundant Cloud Controllers should be studied precisely, because wrong use of this technique can deteriorate system availability.

Corresponding solutions in each item are not implemented in the OpenStack yet. We will study their possibilities for development later in this thesis.

4.2.2 Scheduler

The scheduler gets messages from API server through the messaging server (RabbitMQ) and communicates with compute controllers, network controllers and volume controllers to select the appropriate worker node to execute the request.

Reviewing a scheduler functionalities, we can identify several disastrous impacts of a compromised scheduler on the rest of components and the cloud environment itself.

- Disturbing the IaaS platform availability and Quality of Service (QoS)
- Nefarious manipulation and modification of users' requests
- Reacting to incidents take place in components (e.g. component failures or security breach)

4.2.3 Volume Controller

An attacker can use her/his escalated privileges on a Volume Controller to not only impair system availability but also disclose customers' data on their volumes or even threaten customers' instances by attaching malicious volumes.

4.2. Cloud Platform (OpenStack)

Functionality	
Sc.F.1	Selecting the worker node for an specific request
Sc.F.2	Isolating compromised components
Sc.F.3	Handling components failures
Connected Components	
CC	Cloud Controller
CoC	Compute Controller
NC	Network Controller
VC	Volume Controller
AS	API Server
Access Method	
MSG	Messaging
Impact	
Sc.I.1	Decreasing availability and QoS
Sc.I.2	Intercepting and tampering consumers' requests
Sc.I.3	Threatening containment mechanisms

Table 4.4: Scheduler Specifications

Functionality	
VC.F.1	Creating volumes
VC.F.2	Deleting volumes
VC.F.3	Attaching volumes
VC.F.4	Detaching volumes
Connected Components	
CC	Cloud Controller
CoC	Compute Controller
Sc	Scheduler
Access Methods	
MSG	Messaging
Impacts	
VC.I.1	Decreasing availability
VC.I.2	Attaching consumer's volume to attacker's instance
VC.I.3	Attaching attacker's volume to consumer's instance

Table 4.5: Volume Controller Specifications

4.2. Cloud Platform (OpenStack)

Functionality	
NC.F.1	Creating virtual networks
NC.F.2	Configuring IP packets forwarding
NC.F.3	Handling bridges
NC.F.4	Managing VLANs
NC.F.5	Managing fixed IPs
NC.F.6	Managing floating IPs
Connected Components	
CC	Cloud Controller
Sc	Scheduler
Access Methods	
MSG	Messaging
Impacts	
NC.I.1	Decreasing availability
NC.I.2	Associating attacker's floating IP to consumer's fixed IP
NC.I.3	Associating consumer's floating IP to attacker's fixed IP
NC.I.4	Injecting a malicious instance to a project VLAN
NC.I.5	Manipulating firewall rules

Table 4.6: Network Controller Specifications

4.2.4 Network Controller

Like most of other components, a compromised network controller can affect system availability and QoS in a large scale. A compromised network controller can allocate all IP addresses in a project, preventing running of new instances.

A compromised controller can also associate a customer's floating IP address to attacker's fixed IP address. It will forward customer intended traffic to the attacker's instance.

The network controller has access to firewall rule sets. An attacker can misuse its access and change firewall rules for nefarious purposes.

As described before, a network controller is responsible for creating virtual networks, VLANs and bridges. An attacker can make the controller to add attacker's instance to a specific project, so that instance will be in the same VLAN as other instances of that project.

4.2.5 Compute Controller

Impacts of a compromised compute controller in a cloud environment are:

1. Decrease in availability and QoS may happen in several cases:
 - Decrease in the entire system availability, when the compromised compute do not work as expected and the system do not have enough compute workers to meet the QoS and Service Level Agreement (SLA) requirements.
 - Decrease in the availability of a specific VM instance which is the target of an attacker. It can happen by unscheduled termination or restarting of that instance. Detaching required volumes from instances or attaching wrong volumes to them also lead to service unavailability.
 - A compute worker may also give fake information and statistics to the scheduler. Thus, the scheduler selection algorithm mistakenly chooses the compromised compute controller and this will help the attacker to decrease service availability even more. This approach also signifies impacts of attacks which are described in item 2 and 3.
 - A compromised compute controller starts to allocate IP addresses. Soon the project will run out of free IP addresses, although number of running instances is not as big as number of leased IP addresses.
2. A compromised Compute Controller lets the attacker to start instances without cloud consumer's request. It will not only increase the service cost for the customer, but also can be an initial step in intercepting the cloud consumer's traffic. Assuming that the exploited vulnerability provides the attacker with high privileges in the OS layer, the attacker can listen to the traffic on the compromised machine. Listening to an instance traffic from host OS is a big topic which is out of scope of our study, but it should be considered that eavesdropping can be done by using the hypervisor administration interfaces or by stopping all other instances and listening to the traffic which is going through host NIC card.
3. Detaching customers' volumes from their instances and attaching them to attackers' instances is a threat to data confidentiality of cloud consumers.

4.2. Cloud Platform (OpenStack)

Functionality	
CoC.F.1	Running instances
CoC.F.2	Terminating instances
CoC.F.3	Rebooting instances
CoC.F.4	Handling volumes
CoC.F.5	Providing console output
Connected Components	
CC	Cloud Controller
Sc	Scheduler
VMI	VM Instances
Access Methods	
MSG	Messaging
LM	Local methods
Impacts	
CoC.I.1	Decreased availability
CoC.I.2	Increased service cost
CoC.I.3	Volume data disclosure
CoC.I.4	Traffic interception
CoC.I.5	Fake console output

Table 4.7: Compute Controller Specifications

4. A compromised compute controller can give a fake console output of an instance.

Chapter 5

Detection and Analysis of an Incident (Compromised Component)

Detecting an incident in general is a challenging task. The most tricky part is reducing false positive ratio in incident detection. We try to detect and analysis those incidents which are caused by a compromised component.

In the previous chapter we studied different characteristics of cloud components. We have focused on cloud platform components, functionalities, connected components, access methods and their impacts in case of being compromised. This chapter will use the outcome of the previous one to study detection methods and analyze compromised components. Digging impacts of a compromised component will reveal its symptoms. These symptoms are useful in detecting security breaches and further analysis of their details.

Moreover, traditional incident detection techniques are not enough anymore. Incident detection by means of comparing signatures and stateful approaches are not suitable in this model [19].

5.1 Incidents

According to the NIST guide on computer security incident handling [45], four types of incident may happen in a system: DoS, Malicious code, Unauthorized access, Multiple component. The same grouping can be applied to a cloud environment incidents.

- **Denial of Service:** Each component in a cloud environment can be the target of a DoS attack. In an IaaS service model of Cloud Computing, this component can be under supervision of either the cloud consumer or the cloud provider. The cloud consumer's responsibility may be limited to VM instances that are running over IaaS. However, the cloud provider has more components to protect against a DoS.

- **Malicious Code:** Existence of a malicious code in a cloud component will make that component to behave nefariously. At the cloud consumer's side malicious code may reside in customer's VM images. At the provider's side malfunctioning of a component due to a malicious code has more symptoms, however containing such an incident is more challenging.
- **Unauthorized Access:** An unauthorized access for a cloud consumer may happen by compromising different access methods: Remote management interfaces for controlling their cloud resources and instances (e.g. A Web-based portal provided by cloud provider for their customers); Remote interfaces for connecting to their instances (e.g. SSH connections to GNU/Linux instances).
The same type of incidents may happen for cloud providers. An unauthorized access may let the attack to access a platform component.
- **Multiple Component:** Such an incident includes several other incidents that can be heterogeneous with respect to their types.

For each incident variety of containment and recovery approaches can be used. These approaches will be discussed in Chapter 6.

5.2 Detection

Detecting a security breach, with a reasonable delay, requires a systematic security monitoring. Several functionalities should be provided previously [40]:

- Monitoring of existing event sources (e.g. the operating system and other related services log files)
- Utilization of event sources for monitoring security related events where necessary (e.g. Intrusion Detection System (IDS)).
- Utilization of analysis methods for identifying security incidents.

Beside these systematic monitoring mechanisms, there should be a framework for gathering and analyzing users' reports. Although relying on users' reports for identifying security incidents is not suitable for real-time detection, having such a framework is crucial.

5.2.1 Challenges

Grobauer et al. [40] identified several issues for incident handling in a cloud environment. They also introduce their approach to overcome those issues and corresponding disadvantages of their approach.

Cloud consumers' issues are as follows [40]:

- Event sources are not accessible for cloud consumers as they are controlled by the cloud provider.
- Event sources are fixed and customers cannot add new security specific sensors.
- Communication interfaces for exchanging incident related information are not mature enough (if exists).
- It is hard to find the responsible stakeholder for an specific incident. This also refers to the same challenge named as fate-sharing in Section 2.1.1.

5.2.2 Detection Approaches

Grobauer et al. [40] also have introduced possible approaches and their corresponding challenges:

- The cloud provider should give its customers **enough access to analyze indication sources** that contains relevant information to their incidents. This approach has a major issue. Identifying relevant data sources, publishing enough information and restricting customers to these specific data sources is difficult. The output should be generated for a specific user and should not leak information about other customers or the provider itself.
- Instead of giving direct access to customers, cloud providers can **report occurred incidents to influenced customers**. Unification of these reports is as crucial as their accuracy. Because if a customer receives a detailed report which is not compatible with any standards or previously negotiated format, it won't be useful at all.
- The cloud provider should **develop APIs to deliver systematic event-monitoring** for customers. The APIs have several specifications and requirement which are briefly introduced in [74].

A1	Customers' enough privileges to access and analyze data sources
A2	Cloud providers report incidents to relevant customers
A3	Cloud providers develop event-monitoring APIs for customers
A4	Providers or third parties offer Security-as-a-Service offerings to cloud consumers

Table 5.1: Incident detection approaches [40]

- Another approach is providing **security services for cloud consumers**. "Security as a service" offerings can be provided by either cloud providers or third parties. This is a good approach because regularly customers have no interests in developing their own incident detection capabilities by means of provided facilities (i.e. Analyzing raw data from indication sources, reviewing unified reports from the provider, utilizing the provider's APIs).

This approach has a specific challenge due to limited knowledge of the service provider about the customer resources and infrastructure.

5.3 Analysis

When an incident is detected or a legitimate external report has been received, analysis of the situation should be done immediately. In an analysis of a possible incident, initially it is required to check for false positive alarms. Then, if there was a real incident, different characteristics of that incident should be analyzed [40].

Immediate reaction after a confirmed incident plays a vital role in business continuity, service QoS and its availability. A delayed reaction will let the attacker to clear her/his traces.

5.3.1 Challenges

A cloud consumer may face several challenges in an analysis of an incident in the cloud. As it was in detection phase, the cloud consumer has no knowledge about the cloud provider infrastructure. This lack of knowledge includes missing information about the underlying architecture, location of security indication sources and sensors, exact interaction of the customer's resources with providers infrastructure and so forth [40].

Moreover, in an IaaS service model of a cloud environment, gathering detailed information from firewalls, host services log files and other shared

5.4. Actors' Requirements

A1	Customers' enough privileges to access and analyze data sources
A2	Cloud providers report incidents to relevant customers
A3	Cloud providers develop event-monitoring APIs for customers
A4	Providers or third parties offer Security-as-a-Service offerings to cloud consumers

Table 5.2: Incident analysis approaches [40]

resources is challenging, because they have information about other customers as well. Another obstacle, in the way of analysis for cloud consumers, is about complex relationships of stakeholders in a cloud environment and ambiguous roles of each stakeholder in case of an incident [40].

5.4 Actors' Requirements

Studying the detection and analysis phase of the incident handling procedure, and applying new characteristics of Cloud Computing model, we identified several requirements for a cloud provider and a cloud consumer. Additionally, some influential challenges have been explained which will hinder implementation of these requirements or adaptation of existing mechanisms.

5.4.1 Cloud Providers' Requirements

In order to facilitate incident detection and analysis in a cloud environment, the cloud provider plays a vital role and has a big responsibility. The cloud provider should develop following items to play its role in the incident handling. Most of these items are orthogonal. In other words, a cloud consumer may request several items (i.e. security functionalities, services) together. Also, different consumers may not have similar demands. Thus, it is better developing all of them to cover a larger set of consumers.

- **Security APIs**

The cloud provider should develop set of APIs that deliver event monitoring functionalities and also provide forensic services for authorities. Event monitoring APIs ease systematic incident detection for cloud consumers and even third parties. Forensic services at virtualization level can be implemented by means of a virtual machine introspection libraries. An example of a introspection library is the XenAccess that allows a privileged domain to access live states of other virtual machines.

A cross-layer security approach seems to be the best approach in a distributed environment [74]. This approach should be implemented and analyzed in a real case environment to study its advantages and disadvantages.

- **Precursor or Indication Sources**

The cloud provider deploys, maintains and administrates the cloud infrastructure. The provider also develops required security sensors, logging and monitoring mechanisms to gather enough data for incident detection and analysis at the infrastructure level. As an example, security agents, intrusion monitoring sensors, application log files, report repository, firewall statistics and logs are all part of security relevant indication sources.

In case of a security incident, the cloud provider should provide raw data from these sources to affected customers and stakeholders. Thus they will be capable of analyzing raw data and characterizing incident properties.

This approach has its own challenges which will be discussed in the next section.

- **External reports**

The cloud provider should provide a framework to capture external incident reports. These incidents can be reported by cloud consumers, end users or even third parties. This is not a new approach in handling an incident, however finding the responsible stakeholders for that specific incident and ensuring correctness of the incident¹⁰ require extensive research. An illustration, Amazon has developed "Vulnerability Reporting Process"[21] which delivers same functionalities as described before.

- **Cloud provider's responsibilities**

Although it might not seem very important, a timely response to an incident requires heavy interaction of stakeholders. In order to ease this interaction at the time of crisis, responsibilities of each stakeholder should be described in detail.

- **Security services**

Cloud consumers may not be interested in developing security mechanisms. The cloud provider can deliver a security service to overcome

¹⁰Avoiding false positive alarms

this issue. Security services which are delivered by the provider can be more reliable in case of an incident and less challenging in the deployment and the incident detection/analysis.

When a provider delivers a security service for its customers, the provider already knows about its own infrastructure; thus it won't face any problems in evidence gathering or incident analysis because of missing information about underlying architecture or limited access to indication sources.

- **Infrastructure information**

When the cloud consumer or another third party wants to develop an incident detection and analysis mechanisms, they may need to understand the underlying infrastructure and its architecture. However, without cloud provider cooperation that won't be feasible. So, the cloud provider should disclose enough information to responsible players to detect the incident in a timely fashion and study it to propose the containment strategy.

5.4.2 Cloud Consumers' Requirements

A cloud consumer, as well as its provider, has several responsibilities and must fulfill requirements to ensure effectiveness of an incident detection and analysis.

The following contains identified requirements or possible approaches for a cloud consumer:

- **Consumer's security mechanisms**

The cloud consumers might prefer to develop its own security mechanisms (e.g. incident detection and analysis mechanisms). Customer's security mechanisms can be based on either the cloud provider's APIs or reports from variety of sources, including: provider's incident reports, end-users' vulnerability reports, third parties' reports.

- **Provider's agents in customer's resources**

By implementing provider's agents, the cloud consumer will facilitate approaching a cross-layer security solution. In this method, the cloud consumer will know the exact amount and type of information that has been disclosed. Moreover, neither the cloud consumer nor the provider needs to know about each others' architecture or infrastructure design.

- **Standard communication protocol**

In order to have a systematic incident detection and analysis mechanisms, it is required to agree on a standard communication protocol that will be used by all stakeholders. This protocol should be independent of a specific provider/customer.

- **Report to other stakeholders**

If the customer cannot implement the provider's agent in its own instances, another approach to informing stakeholders about an incident is by means of traditional reporting mechanisms.

These reports should not be limited to an incident only, customers may also use this mechanism to announce a suspicious behavior for more analysis.

- **Cloud consumer's responsibilities**

Roles and responsibilities of a cloud consumer in case of an incident should be defined previously; thus it will be feasible to react immediately in a crisis. It should be clear that after detecting the first symptoms of an incident, the cloud consumer must start communicating with which components of a cloud and expect what kind of responses.

5.4.3 Challenges of Proposed Approaches

As it was discussed in previous sections, proposed approaches by different authors and also our own approaches have some disadvantages. Brief overview of challenges is given in this part.

- **Complexity of providing customer's specific data from shared precursor or indication sources**

In a cloud environment indication sources, security sensors, and log files contain information about all customers and even the provider itself. In such a system, it is important to filter out not-relevant data, when giving them out to a specific customer for incident analysis. This issue is not only limited to customers' data, but also exists when the provider try to disclose a specific data to other stakeholders for further analysis.

- **Adaption of existing security mechanisms**

Although Cloud Computing has few new technologies, applying existing security mechanisms may not happen smoothly. New character-

istics of a cloud environment impose several constraints on a security mechanism.

From an incident detection and analysis perspective, *real time analysis techniques* must be improved to provide timely responses. Also, these techniques should analyze the system while it is running (*live analysis of an instance*); This will ensure the service availability and promised QoS level. Analysis of a running instance might be possible by taking a snapshot of the instance and investigating that snapshot.

Another required improvement for a detection and analysis mechanisms of a cloud environment can be achieved by changing log/report generation techniques. The awareness about resource sharing and multi-customer fashion of the environment should be added to these techniques. As one of the outcomes, the log generation can generate separated log files for customers or utilize labeling techniques to tag a customer specific log record. It will also relax the first challenge in this section.

- **Lack of information about other parties**

The cloud consumer requires provider's infrastructure architecture to understand about the nature of an incident and its threat. Security service provider (i.e. external third parties, or cloud provider) should know about the business logic and architecture of the cloud consumer to deliver the suitable security service.

- **Lack of standard protocols and interfaces for communication**

As Cloud Computing is a new computing model, incident reports and detection-related communications are not standardized yet. Additionally, unified interfaces do not exist or used widely.

5.5 Detection and Analysis in an OpenStack Deployment

We discussed challenges of an incident detection and analysis in a cloud environment. Moreover, several requirements have been identified for a cloud provider and its customers.

Despite the fact that new requirements are imposed to the incident detection and analysis mechanisms, the general framework for incident detection and analysis remains the same. In this section we will go through all steps for incident detection and analysis, and apply them for a cloud

environment which is powered by the OpenStack. We use the incident handling procedure, proposed by NIST, *COMPUTER SECURITY INCIDENT HANDLING GUIDE* [45].

This procedure has six main steps: Identifying signs of an incident, Specifying sources for precursors and indications, Analysis of the incident, Documentation, Prioritization, and Notification of the incident [45].

5.5.1 Identifying signs of an incident

Signs of an incident have been identified previously in Chapter 4. Impacts of exploiting a vulnerability are symptoms of a specific incident. These impacts for the OpenStack cloud platform are listed in Tables 4.3, 4.7, 4.6, 4.4, and 4.5.

5.5.2 Specifying precursors and indications sources

NIST introduced several sources for Indications and Precursors. Equivalent sources in an OpenStack deployment will be introduced in this part.

NIST proposed four types of data sources: Computer security software alerts, Logs, Publicly available information, and people [45]. Detailed mapping is as follows:

Computer security software alerts

- **Intrusion Detection Prevention System (IDPS):**

The OpenStack has no embedded IDPS functionalities. But it is possible to setup an IDPS on host machines which are using the OpenStack cloud platform. It is also possible to offer an IDPS service to cloud consumers which will be discussed later.

- **Antivirus, antispymware, antispam:**

The OpenStack has no embedded facility for securing the cloud against malicious software. Adding this functionality to the host machine is feasible.

- **File integrity checker:**

It does not exist as a feature of the OpenStack but can be added by using third party integrity checkers.

- **Third party monitoring service:**

Although no such services have been introduced yet, the modular and

distributed design of the OpenStack make it possible to employ a third party monitoring service and plug it to the rest of the system.

Logs

- **OS, services, and applications logs:**

The current release of the OpenStack only works over GNU/Linux OSes; In this type of OS, important system log files and relevant services/applications logs are:

- `/var/log/messages` : General and system related logs
- `/var/log/mysql*` : MySQL database logs
- `/var/log/nova/*` : OpenStack Compute project logs
- `/var/log/libvirt/*` : libvirt [7] virtualization API logs
- `/var/log/apparmor/*` : AppArmor [73] logs

In addition to simple textual log files, other high level tools are also useful. As an example of these tools, *top*, *meminfo*, *ipstats* should be mentioned. They audit runtime system information

- **Network device logs:**

Both physical and virtual network devices provide this kind of log files.

The OpenStack networking log file is located at `/var/log/nova/nova-network.log.*`. The OpenStack will also support other virtual network devices in the future (e.g. Open vSwitch [9]), in that case, their corresponding log files should be audited as well.

Publicly available information

- **Public vulnerabilities and exploits:**

Publicly available bugs are reported to <https://bugs.launchpad.net/openstack>.

A security vulnerability can also be found among them. Publicly available exploits are listed in different web sites (e.g. www.milw0rm.com), and are developed in variety of projects (e.g. the Metasploit project)

- **Incidents at other organizations:**

The kind of data sources are more dependent on organizational interactions and awareness; thus it is out of the scope of our study.

People

This indication type is completely out of our research scope.

5.5.3 Analysis of the incident

With respect to the NIST guide on incident handling [45], analysis of an incident may not follow any specific steps. But there are recommendations in order to have an easier and more effective analysis. We will mention useful recommendations, in our research, from the NIST guideline and discuss their realization in an OpenStack deployment.

- **”Profile Networks and Systems”**

Although this is not an accurate measure to determine the expected activity of a component, profiling networks and systems is a vital technique in detection and analysis.

In an OpenStack deployment, we should profile important components and determine their expected activities. Components are those items which were identified in Chapter 4. An example of component activity is its network bandwidth usage.

In our lab setup, we used a variety of monitoring and profiling techniques to determine expected activities of components under different tasks. Mainly we used the RRDTool [55] for the statistical data gathering and Cacti [4] for the demonstration. Detailed information on our measures and graphs are available in Chapter 8.

- **”Understand Normal Behavior”**

In order to distinguish the malicious behavior from the normal one, the normal behavior should be described previously. As an example, the required steps for starting an instance should be clear to the incident handler. We explain steps of starting an instance in Section 8.1.6 and the corresponding internal procedure in Section 8.1.7.

Understanding the normal functionality of a highly distributed cloud platform is not easy. Thus, systematic approaches on each component should be utilized. When we focus on a single component, the complexity of a large system is avoided. However, components interaction is an important measure in determining the malicious behavior of a component. So, expected interaction from a component should also be determined and compared to the actual one.

- **”Use Centralized Logging and Create a Log Retention Policy”**

In a cloud environment having centralized logging mechanism is not feasible and effective. In a large scale deployment of the OpenStack

with one million hosts, providing centralized logging/analysis is a waste of resources. Instead modern logging and notification mechanisms should be employed. As an example, we mention two logging approaches, first one is a cloud specific logging which is proposed by Golovinsky et al., *Syslog Extension for Cloud Using Syslog Structured Data* [39]. Second approach is a more general one, named Common Event Expression (CEE) [77], that introduces a standard log language for event description, logging and exchange. Using the latter, correlation, aggregation, auditing and incident handling become easier. On the other hand, the former one focuses on a cloud environment specifications¹¹.

- **”Keep All Host Clocks Synchronized”**

To provide a meaningful analysis, events should be stamped based on the same clocks. Using Network Time Protocol (NTP) is technique for synchronizing clocks among several machine.

- **”Create a Diagnosis Matrix”**

Creating such a matrix is useful to understand the relation between a symptom and the associated incident category. We can also break down each category and identify several incidents in them, then depict the symptom-incident relations.

- **”Consider Filtering the Data”**

In a highly distributed environment such as a cloud system, an enormous number of events are reported per time unit. Analyzing all of them without any prior filtering is not feasible. Thus, input data should be abstracted and filtered before analysis.

- **”Run Packet Sniffers to Collect Additional Data”**

Analysis of network traffic is also an import task in incident handling. Virtual switches and network equipments are new components in network layer of Cloud Computing. Traffic analysis and interception of a virtualized environment, when the traffic does not pass real hardware, should be handled at the virtualization layer¹².

¹¹The IETF approach has been criticized a lot and it is in an experimental status, currently

¹²We may assume the bridge functionality of the Linux kernel as a utility for the virtualization layer. This is noteworthy because we may not use virtualization specific networking mechanisms (e.g. Open vSwitch, Open Flow) and use bridge mechanisms for the communication of instances in the same host

Chapter 6

Containment and Recovery of the Compromised Component

Current security mechanisms cannot secure cloud environments against all attacks; they cannot even provide a timely action in response to a successful attack [23].

Containment strategies differ in different cloud environments. Also, cloud consumers' allocated resources are not under their direct/physical control. Consumers control their resources using several access methods which may get compromised as well. Specifically in the IaaS service model, the issue is more challenging for responsible organizations (i.e. providers). One of the main reasons is the increased control of a cloud consumer over its allocated resources and virtual instances [63].

The cloud consumer may develop some procedures for containing its service in case of an incident, but applying these procedures is challenging as well. The cloud provider has to ensure that recent changes in the normal operation of a specific service is due to an incident and not a false positive.

Another challenge in Cloud Computing containment is related to conflicts between cloud providers' and cloud consumers' containment procedures and policies.

We identified several aspects that should be considered in this phase:

1. We should address the greatest risks and strive for sufficient risk mitigation at the lowest cost, with minimal impact on other mission capabilities [72].
2. The containment, eradication, and recovery should be done in a cost effective fashion. Thus, a cost-benefit analysis of each approach should be performed before application.
3. In a highly distributed system such as a cloud environment, we cannot apply stateful measures, they won't scale.

4. It is not feasible to stop all attacks or secure all components to avoid exploiting any existing vulnerabilities.
5. In addition to the previous item, existing security mechanisms are not completely applicable to the new computing model and they cannot protect the system from all attacks and cannot provide a fast reactive response to an incident.
6. As we cannot harden a cloud environment against all possible attacks, containment strategies and tolerating a successful attack are required approaches.

Our study approach is a case-based one, because:

- Several components, with different functionalities, may require a variety of containment realization mechanisms.
- Providing a single mechanism to cover all incidents, is not possible.
- A combination of mechanisms is possible, and also recommended for covering an attack which exploits several vulnerabilities.
- In each case, we will study different ways of an incident occurrence (e.g. malicious code can be injected in to either a cloud platform service (nova-compute) or OS modules/services.)

6.1 Existing Approaches

This section will discuss a couple of existing approaches which are applicable in the containment, eradication, and recovery phase of the incident handling process. We will go through them briefly and explain why they are useful in our deployment. Moreover, we will mention their drawbacks and weaknesses. We will use modified version of these approaches along our proposed solutions (Chapter 7) to fulfill this phase requirements.

6.1.1 Intrusion Tolerance

The intrusion tolerance approach utilizes fault tolerance technology for underlying hardware and software layers to provide service continuity and acceptable QoS while having compromised components.

Several approaches exist for tolerating intrusions. In this section we will go through them, and study their advantages, disadvantages. Chapter 7 contains our own approaches that utilize these mechanisms as well.

A set of intrusion tolerant techniques are mentioned in the following, that are applicable to our research:

- CC-VIT: Virtualization Intrusion Tolerance Based on Cloud Computing [76]
- SITAR: A Scalable Intrusion Tolerant Architecture for Distributed Services [80]
- CloudFIT: A platform for deployment of Intrusion and Fault Tolerant applications in a cloud. This approach utilizes intrusion tolerant systems using replicas. [6]

CC-VIT

Tan et al. proposed "*CC-VIT: Virtualization Intrusion Tolerance Based on Cloud Computing*" approach that has five main technologies, comprising: Hybrid Fault Model, Active Replicas and Passive Replicas, State Updates and State Transfer, Proactive Recovery and Diversity [76].

Hybrid Fault Model The Hybrid model provides features that supports heterogeneous service replicas, by service replicas we mean operating systems, middleware platforms and top layer¹³ services. It can be applied in two ways: Redundant Execution on Single Host (RESH) and Redundant Execution on Multiple Host (REMH) [76].

In RESH, redundant replications of a same service will be executed on the same physical machine. Each replication will run on an isolated virtual machine. This application tolerates random faults in replicas which disrupt the service delivery. On the other hand, in REMH, redundant replications are utilized again but this time they can be distributed over several physical machines. The communication among these replicas is handled by means of Group Communication.

Active Replicas and Passive Replicas Passive replicas are introduced in this model to reduce service cost, resource consumption and increase resource utilization. At least $F+1$ active replicas is required in a working system to detect an error. When a failure is detected in an active replica, a passive replica will replace that instance [76]. Instead of using $3F+1$ replicas for tolerating F failure in the system as in the Byzantine Fault Tolerant

¹³We focus on layers in the cloud stack.

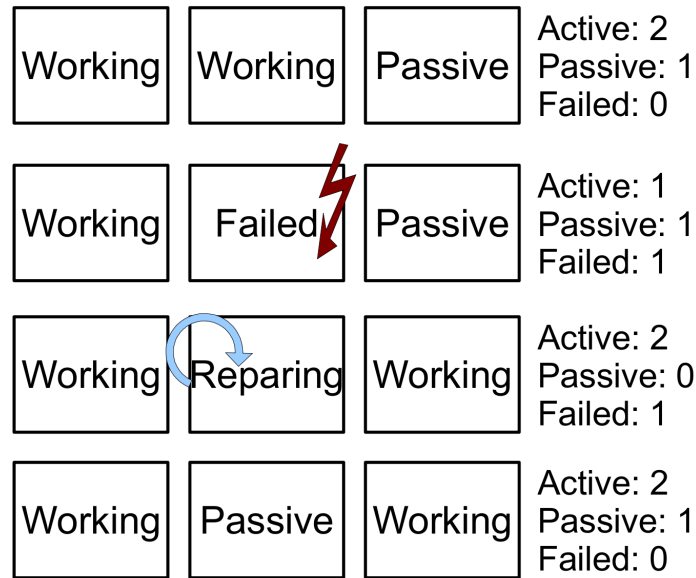


Figure 6.1: An example of passive replicas and failures in the CC-VIT approach.

(BFT) [51] algorithm, they propose using $2F+1$ replicas to tolerate same number of failures, Figure 6.1. The experiment results also support their claim.

It is not clear exactly what they mean by passive replica¹⁴. A passive replica can be either warm or cold standby that has a significant influence on activation time and failure probability. For more information on dependability analysis check [58].

State Updates and State Transfer Handling states is another important aspect of redundant replications. Stateless replication is faster and more efficient because the state synchronization time (i.e. state store, state transfer and state update time) and network/storage overhead is omitted. The stateless approach may lead to data loss and lower efficiency with regards to failure specification and timing details [76].

CC-VIT uses regular state updates to synchronize passive replicas. The replica manager is responsible for activating passive replicas and updates their states after a specific number of state changes.

¹⁴We contacted authors to verify the meaning but got no response from them.

Proactive Recovery Traditional intrusion tolerant systems can only tolerate finite number of replica failures. Thus, in a long run of the system it will face catastrophic failure when the number of failed replicas exceeds the system threshold.

CC-VIT establishes a proactive recovery plan to reinitialize failed replicas on a regular basis. This recovery strategy has its own drawbacks as well, including: higher resource consumption due to proactive recovery overhead, longer downtime because of state synchronization constraints [76].

Diversity Having diversity among replicas helps the system to survive for a longer period of time under an adversary attack. An adversary is not able to exploit same weaknesses if there exist enough diversity in the environment.

CloudFIT

The goal of CloudFIT project is to develop a fault and intrusion tolerant application for the cloud model, based on Byzantine Fault Tolerance (BFT). The main difference between CloudFIT and CC-VIT is about the number of replicas and their management. In the CloudFIT approach they use two characteristics of a cloud environment: resource elasticity and on demand access to a shared pool of computing resources. Thus, they change number of replicas in a dynamic fashion.

Additionally, trustworthiness of the virtualization infrastructure is one of their assumptions.

6.1.2 Deployment Models

G. Zhao et al. studied five deployment models of a cloud environment to eliminate specific security concerns from Cloud Computing [86]. They focused on the following security concerns:

- Service availability and reliability in case of a failure.
- Data lock-in concerns.
- Security of data stored in the cloud (i.e. confidentiality, integrity and authentication).

They proposed five deployment models to address previous concerns. They separated compute service of their cloud model from its storage service. Application of these reference models to OpenStack is made easier and

6.2. Containment, Eradication, and Recovery in an OpenStack Deployment

more feasible, because OpenStack also has two projects: Nova (OpenStack Compute) and Swift (OpenStack Storage).

These models are as follows:

1. Separation Model: Avoid excessive control of a single provider
2. Availability Model: Decrease probability of a catastrophic failure by establishing extra Data Processing Service and Cloud Storage Service. This model should be used with caution. It should be noticed that adding more replication does not necessarily increase availability and decrease probability of a catastrophic failures. For more information and detailed discussion refer to Book [58].
3. Migration Model: This model specifically replicates the storage service by means of Cloud Migration Service. As a result, cloud consumers' data will not tied to a specific cloud provider.
4. Tunnel Model: This model provides level of isolation between Data Processing and Cloud Storage. It makes the collusion of these two services difficult.
5. Cryptography Model: It is derived from Tunnel Model and adds another layer of security to the system by providing cryptography service for data stored in Cloud Storage.

This approach has several advantages. First, they are at deployment level, despite that most of other solutions are at implementation level. When a solution is at the deployment level, it is easier to apply it to existing platforms, while solutions at the implementation level require heavy code update and implementation in the platform. Secondly, the solution can be utilized among several clouds and it relies on inter-cloud interaction. Thirdly, employed models are not opaque for users, thus they can trust in their systems as they are aware of its components' interactions ("Models are user oriented"). Existing solutions are more development oriented so they are opaque for users. [86]

6.2 Containment, Eradication, and Recovery in an OpenStack Deployment

As in the previous section, we use the NIST guideline [45] for containment, eradication, and recovery. Containment procedures are not independent of

the incident type. Thus, each incident type has its own set of containment strategies [45].

As we mentioned in the beginning of this chapter, we will study a few incident cases. In each case, a set of items will be either mentioned or discussed, some other parameters should also be identified during incident handling:

- Infrastructure description
This includes the OpenStack deployment architecture, components interactions, running services on each node, etc.
- Estimated date/time when the incident started
Knowing how long the attack was ongoing is important. It will help in determining impacts of the incident on the cloud environment.
- Incident type
Each incident requires a different set of handling techniques, thus identifying the incident type is crucial. We categorize incidents in the same set of types as proposed by the NIST: Denial of Service, Malicious Code, Unauthorized Access, Inappropriate Usage, and Multiple Component incident. The incident type does not contain same information as the incident description. It will be used to identify and apply the set of procedures proposed by the NIST for each incident type.
- Current status of the incident
At the time of handling the incident, depending on the current status of the the incident, handling procedures may vary.
- Incident description
The description of the detected incident should be provided. This description contains information such as, incident distribution, characteristics, sources, targets, etc. As an example, the incident distribution determines the number of affected components, their physical/virtual locations, related compromised layers, etc.
- Affected resources description
During the incident handling procedure, affected cloud components and their roles in a working cloud environment should be identified.
- Response actions
In each case, we will explain techniques which have been used for mitigating and handling the incident after its detection.

- Causes of the incident

The last step in incident handling is to identify threat sources and vulnerabilities that caused the incident and avoid same causes in other components and future operation of the cloud environment.

In each of the following scenarios, we will enumerate recommended actions by NIST and explain their realization in a cloud environment which is powered by the OpenStack platform.

6.2.1 Case One: A Compromised Compute Worker

The first case which we will discuss, has only one compromised component. In this case the nova-compute service in the compute worker is compromised, Figure 6.2. Detailed description of these components and their responsibilities are explained in Chapter 3.

Two incidents have happened simultaneously in this scenario, malicious code and unauthorized access. The malicious code is injected to the nova-compute service and introduces some misbehavior in it, such as malfunctions in the hosting service of virtual instances, nefarious usage of granted privileges to request for more IP addresses and cause IP address exhaustion in a specific consumer's project.

The malicious code is injected by means of another incident, unauthorized access. The attacker gains access to resources on the OpenStack-4 host, that he/she was not intended to have. Using those escalated privileges, the attacker changed the python code of the nova-compute and restarted the service. Thus, nova-compute started to behave maliciously.

We also discuss each incident and its cloud environment realization in Section 5.1. Table 6.3 contains specifications of this scenario.

Recommended actions by NIST and their corresponding realization in the OpenStack deployment are explained in the next three parts. Each of these parts is related to a specific major task in incident handling, comprising: containment, eradication, and recovery.

Containment

As we explained the case one scenario, it is a combination of two types of incidents, malicious code and unauthorized access. The first part will discuss our realization of recommended containment strategies for malicious code, and the second part explains the same concept for an unauthorized access.

Four actions which are recommended responses to a malicious code incident:

6.2. Containment, Eradication, and Recovery in an OpenStack Deployment

Infrastructure description	
Architecture	Simple hierarchical
Operating System	Ubuntu 10.04.1 LTS
Number of compute node	4
Number of cloud controller	1
Total number of physical machines	4
Average number of running instances on each host	10
Total number of running instances	40
Incident description	
Incident type	Malicious code and Unauthorized access
Current status	Ongoing attack, the malicious code is not patched nor contained yet
Compromised component(s)	One compute worker host
Physical Location	OpenStack-4
Affected Layers	Cloud platform layer, the OpenStack nova-compute service
General Information	Malicious code is injected into the nova-compute service of the OpenStack-4 host
Resources at risk	Running instances on OpenStack-4, Stakeholders and resources interacting with running instance on OpenStack-4 or the infected nova-compute service

Table 6.1: Case One - A compromised compute worker scenario specifications

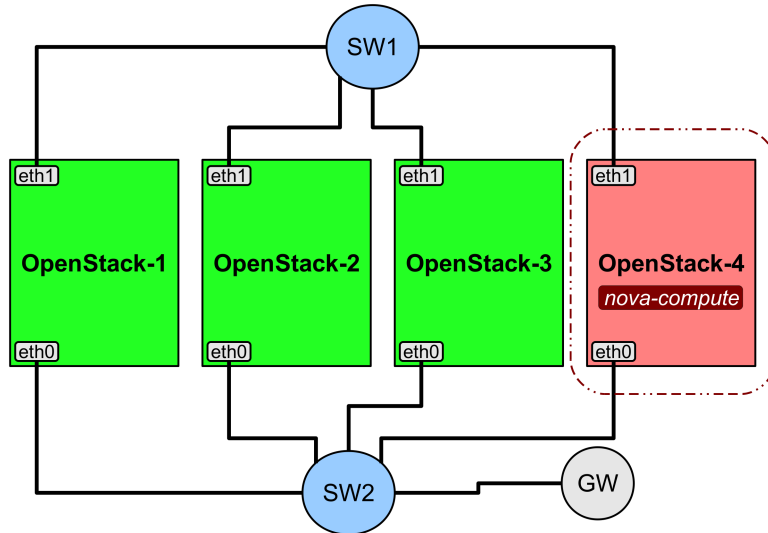


Figure 6.2: Case One - The nova-compute service in the OpenStack-4 host is compromised.

- **”Identifying and Isolating Other Infected Hosts”**

Study the profile of the infected host and compare it to other worker nodes profiles, in order to identify compromised hosts. Comparing profiles of components is simple using provided monitoring facilities. The incident handler should login to `http://openstack-4/cacti`, and choose the *List View* then select those parameters that he wants to compare. By pressing the *View*, the handler can compare the profile of the infected host with other suspicious hosts.

- **”Blocking Particular Hosts”**

The strategy should be analyzed in depth before its application. In a cloud environment when the consumer’s instance is running in an infected worker node, it is not plausible to disconnect the node without prior notice/negotiation to affected consumers (This constraint can be relaxed by providing the proper SLA).

In addition, blocking the compromised host can be done with different levels of restrictions. Initially the communication with the outside of the organization should be blocked¹⁵, assuming that the attacker is

¹⁵By the term *organization*, we mean all entities who are responsible for managing the cloud infrastructure, which can be referred to as the cloud provider.

located outside of the organization infrastructure. Also, any further attack to the outside of the organization using compromised hosts will be mitigated.

In the second step, communication of the compromised host with other components in the infrastructure is also restricted and the host is marked as compromised/infected/suspicious. Thus, other nodes will avoid non-critical communication with the compromised node. It will help the infrastructure to communicate with the compromised node for containment, eradication and recovery procedures and at the same time the risk of spreading the infection is reduced.

The last step can be blocking the host completely. In this approach staff should access the host directly for analyzing the attack as well as assessing possible mitigation, handling strategies.

Moreover, blocking infected hosts will not contain the incident. Each host has several consumers' instances (VM instances) and volumes running on and attached to it. Blocking hosts will only avoid spreading the incident to other hosts but instances are still in danger. An approach in a cloud environment is to disconnect instances and volumes from the underlying compromised layer. Signaling the cloud software running on the compromised host to release/terminate/shutdown/migrate instances and detach volumes are our proposed approaches. A drawing of this approach is in Figure 6.3. We should use a quarantine compute worker node as the container for migrated instances. After ensuring the integrity and healthiness of instances they can be moved to a regular worker node. This quarantine compute worker will be explained more in the following chapter.

These approaches can be implemented at the cloud infrastructure layer for simplicity (Blocking by means of nodes firewall, routers, etc.)

- **”Soliciting User Participation”**

The interaction can be implemented using different methods. Security bulletins maintained by cloud or service providers is an example of notifying other stakeholders about an incident. Incident or vulnerability reporting mechanisms are also useful when an outsider detects an incident or identifies a vulnerability. These two methods can be developed and deployed independent of the cloud platform. Security bulletins are provided by the security team who handles security related tasks. Also, reporting mechanisms are delivered by means of ticketing and reporting tools.

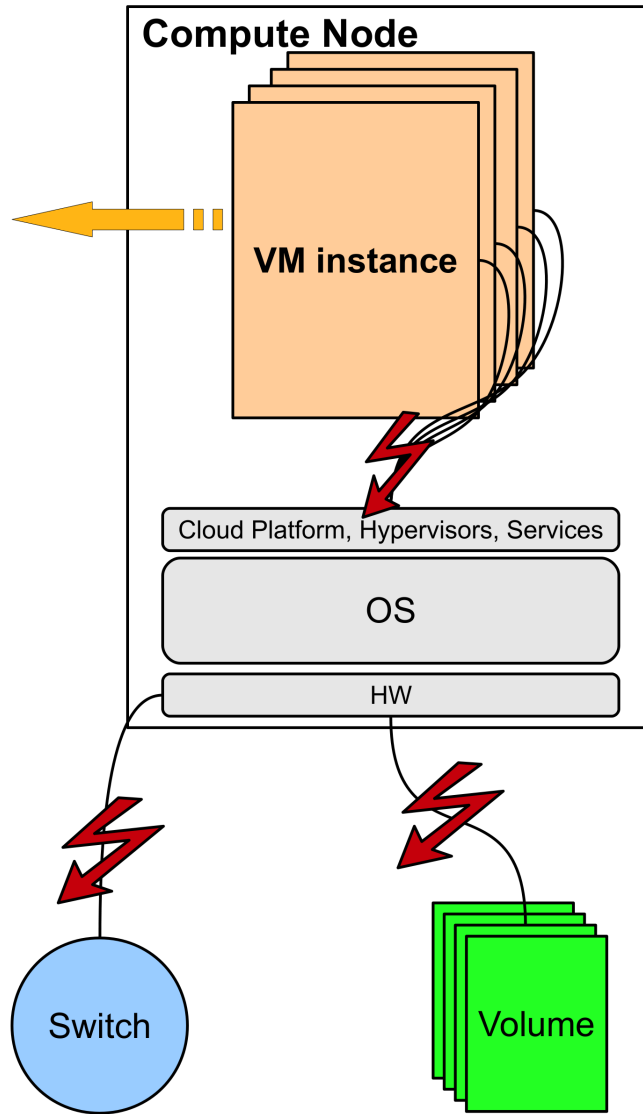


Figure 6.3: Blocking compromised compute communication. Red lightning represent disconnected communications.

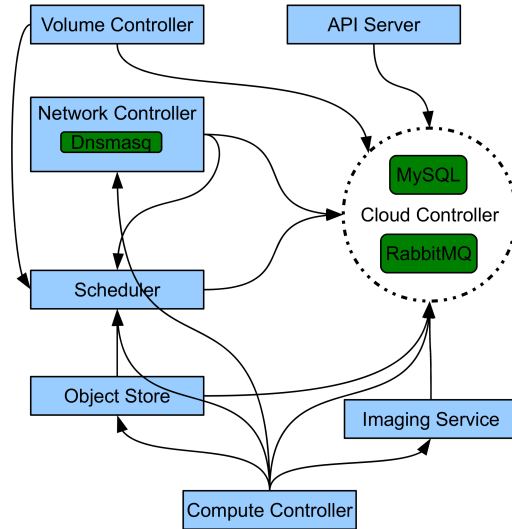


Figure 6.4: OpenStack Nova services dependencies.

Direct and real-time communication among stakeholders is a complement to above mentioned methods.

- **”Disabling Services”**

In order to disable a particular service, we should check the service dependencies diagram first. An example of such a diagram is depicted in Figure 6.4. Disabling a service can take place in two ways.

It is possible to stop the service at the compromised host Figure 6.5. In our scenario we can stop the nova-compute service to disable the compute service (service nova-compute stop). It will instantly disconnect the cloud platform from running VM instances. *In the OpenStack platform stopping the nova-compute service will not terminate running instances on that host.* Thus, although the compute service is not working anymore, already running instances will continue to work even after terminating nova-compute. Additionally, it is not possible to terminate an instance after stopping its corresponding compute service, because the administration gateway (i.e. nova-compute) is not listening to published messages. In order to maintain control over running instances we can migrate instances from the compromised node to a healthy one before we terminate the compute service¹⁶.

¹⁶We should consider migrating instances to a quarantine node first, and move them to

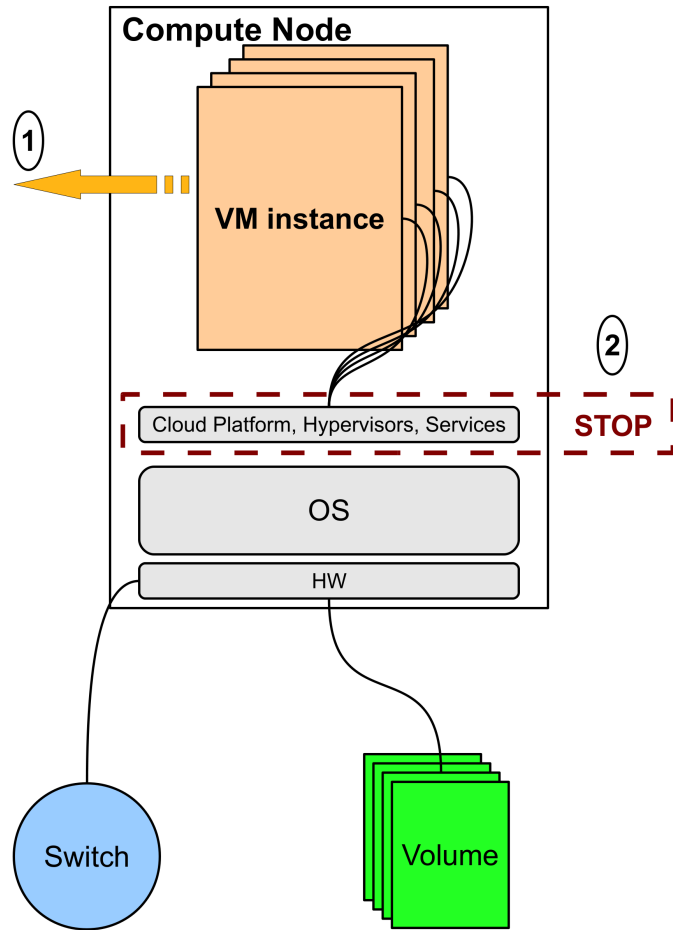


Figure 6.5: Stopping the compute service at the compromised host.

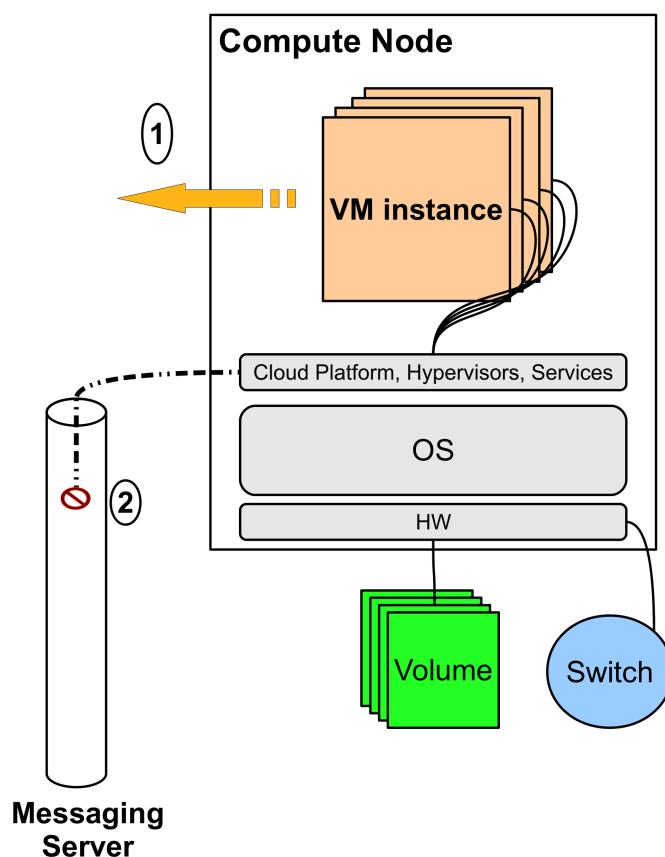


Figure 6.6: Discarding messages to/from the compromised node.

Another approach is discarding messages published by the compromised component or those destined to it, Figure 6.6. This is a centralized method and the cloud controller or the messaging server should filter out messages with the source/destination of the infected host¹⁷.

We explained four actions for containing a malicious code incident now we continue by explaining four other actions which are recommended responses to an unauthorized access incident:

a regular node after ensuring their integrity and healthiness, as explained in Section 7.1.6

¹⁷In a publisher/subscriber paradigm the destination may be eliminated or masked by other parameters. So, we may filter messages that contain any evidence of being related to the infected host.

6.2. Containment, Eradication, and Recovery in an OpenStack Deployment

NIST recommended action	Brief Description
"Identifying and Isolating Other Infected Hosts"	Extract incident symptoms to detect other infected hosts.
"Blocking Particular Hosts"	After identifying the compromised component and its corresponding host (i.e. the compromised worker/-compute host), that host should be blocked.
"Soliciting User Participation"	Interaction among cloud stakeholders (e.g. cloud providers, cloud consumers, third parties, end users, etc.) is a mandatory step toward fulfilling incident containment requirements.
"Disabling Services"	Disabling the infected service (nova-compute in our scenario) may reduce impacts of the compromised host. Disabling a service can disrupt other services and cause deviation from promised SLA by the provider.

Table 6.2: Containment Strategies

- **"Isolate the affected systems"**

The same procedures as those which have been explained for "Identifying and Isolating Other Infected Hosts" (Section 6.2.1) and "Blocking Particular Hosts" (Section 6.2.1) can be applied here.

- **"Disable the affected service"**

The same procedure as the one which has been explained for "Disabling Services" (Section 6.2.1) can be applied here.

- **"Eliminate the attacker's route into the environment"**

Access methods which have been used by the attacker to access cloud components should be blocked. Complete list of access methods to each component, in an OpenStack deployment, has been introduced in Chapter 3.

Implementing filtering mechanisms in the messaging server is a crucial requirement which is highlighted in different strategies. The cloud provider should be capable of blocking messages which are related to the attack and blocks the attacker's route into the cloud environment.

It should be noted that the mechanisms which we have used to fulfill requirements imposed by "Blocking Particular Hosts", "Disabling Services"

identifying and Isolating Other Infected Hosts” and “Disabling Services” (Section 6.2.1) are applicable in this strategy as well.

- **”Disable user accounts that may have been used in the attack”**

The user account that has been used by the attacker can be in different layers, such as system, cloud platform, or VM instances layer¹⁸. Based on the membership layer, the disabling and containment procedure will differ.

The user at OS layer (i.e. system user) of the worker host should be disabled by direct access to the host or by means of directory/federation service in case they are in place. In our deployment we did not exploit a federation service so we should disable the system user by accessing the host.

System users’ accounts have more administrative responsibilities and are not assigned to cloud consumers. Thus in case of a security breach with a system user involvement, the threat to the cloud environment will be limited to the system and hosts which have accepted the attacker’s user as a legitimate one. In the operating system we have used in the lab setup, disabling a user is as follows:

Listing 6.1: Disabling a system user

```
root@openstack-1:~# passwd -l <USER>
```

The attacker may also have escalated privileges at the OS layer, and can inject a rootkit to modify the authentication process. Disabling a system user through standard procedures will not cease nefarious impacts of an authentication rootkit. Such a scenario reveals the importance of monitoring and intrusion detection systems in a cloud environment. As described previously, we assume that the attacker will not have enough access to inject a rootkit backdoor.

Cloud platform users can be defined in two main categories: cloud provider’s users, cloud consumers’ users. Cloud provider’s users are responsible for administration, management, and maintenance of the cloud environment. However, user accounts in the cloud consumer category are assigned to consumers and are responsible for provisioning

¹⁸It should be noted, although we may use directory and federation services to unify users among services and layers, this may not be a feasible nor plausible approach in a cloud environment. However, federation is applicable at each layer (e.g. system, cloud platform, VM instances).

requested services and resources, as well as administrating and managing them. Moreover, one of the main issues with a hijacked platform user is that its impacts are not limited to a specific host, thus the incident handler cannot contain the incident by blocking a specific host.

Using a provider's user account to attack the cloud environment can cause serious damage and affect most components of the cloud model. However, nefarious usage of a cloud consumer account will affect consumers' service and may hardly reach to underlying provider resources.

Disabling a user in the OpenStack platform can be done using the following command:

Listing 6.2: Revoking an OpenStack user's credentials

```
root@openstack-1:~# nova-manage user revoke <USER>
```

This will revoke the compromised user's credentials, so after restarting the VPN service that specific user cannot connect to the project anymore. It is also possible to delete the attacker's user:

Listing 6.3: Deleting an OpenStack user

```
root@openstack-1:~# nova-manage user delete <USER>
```

Eradication

- **"Disinfect, quarantine, delete, and replace infected files"**

These strategies are applicable in two layers depending on the container of the injected malicious code. The malicious code can be injected in to either the cloud platform services (i.e. nova-compute) or the OS modules/services.

If the injected malicious code is in OS modules/services, utilizing existing techniques are effective. By existing techniques, we refer to anti virus software and traditional malware handling mechanisms. In this case nothing new has happened, although side effects of the incident may vary a lot.

However, if the malicious code is injected into a cloud platform service (in our case nova-compute), existing anti virus products are not useful, as they are not aware of the new context. Cleaning a cloud platform service can be very hard, so other approaches are more plausible. In general, we can propose several approaches for eradicating a malicious code incident in a cloud platform:

- Updating the code to the latest stable version and apply appropriate patches to fix the vulnerability.
- Purging the infected service on the compromised node
- Replacing the infected service with another one that uses a different set of application layer resources (e.g. configuration files, repositories, etc.)

It should be noted that in a highly distributed system such as a cloud environment, doing complicated tasks such as fixing a single infected node in real time fashion does not support the cost effectiveness policy. Thus, terminating the infected service or even the compromised node and postponing the eradication phase can be an appropriate strategy.

- **”Mitigate the exploited vulnerabilities for other hosts within the organization”**

In order to complete the task, we should also update the cloud platform software on other nodes and patch identified vulnerabilities.

Recovery

- **”Confirm that the affected systems are functioning normally”**

Profiling the system is useful in the recovery phase as well as detection and analysis phase. After containment and eradication of the compromised component, the component profile should be the same as a healthy component or be the same as its own profile before being infected. Using the provided tools in our deployment (i.e. Cacti) we can specify the exact period and components which we want to compare.

- **”If necessary, implement additional monitoring to look for future related activity”**

After identifying the attack patterns and the compromised node profile, we should add proper monitoring alarms to cover those patterns and profiles. As an example, if the compromised compute worker starts to request for a large number of IP addresses, after its infection, this pattern should be saved and monitored on other compute workers. So, if we experience a compute worker with the same profile and behavior, that worker node will become suspicious for being infected.

In our monitoring tools, the administrator can define threshold for different parameters; if the current profile of the system violates the

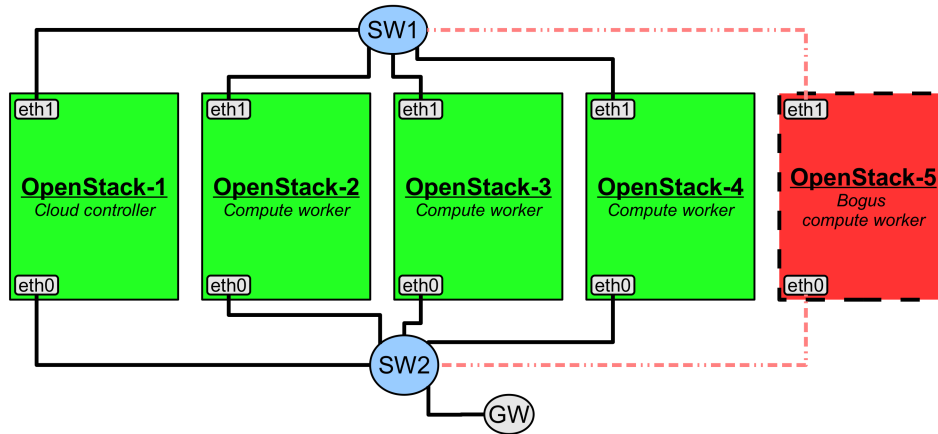


Figure 6.7: Case Two - A physical bogus compute worker node is added to the infrastructure.

threshold, graphs will be drawn with other color to notify the user. We can also add other monitoring tools to generate the ticket in case of a matching profile, that is not required yet.

6.2.2 Case Two: A bogus component

A bogus service is a threat for the cloud environment security. As the OpenStack is an open source software, an attacker can access the source code or its binaries and deploy a cloud platform service. When the attacker is managing a service, he/she can manipulate the service in a way that threaten the integrity and confidentiality of the environment. This section will discuss such an incident that a bogus cloud platform component is added to the environment. We will focus on a nova-compute service as the bogus cloud platform component.

A bogus nova-compute service or in general any cloud platform component can run on a physical machine or a virtual instance. Adding a physical node to the cloud infrastructure by an attacker, is unlikely; however, for the sake of completeness we study both the case that the bogus service is running on a new physical machine and the one when it is running on a virtual instance. Both cases are depicted in Figures 6.7, and 6.8.

When the bogus service is running on top of an instance, the network connectivity may be more limited comparing to the other case (i.e. the bogus service is running on a physical node.). Initially any given instance is only connected to the second interface, (*eth1*). This connectivity is provided

6.2. Containment, Eradication, and Recovery in an OpenStack Deployment

Infrastructure description	
Architecture	Simple hierarchical
Operating System	Ubuntu 10.04.1 LTS
Number of compute node	4
Number of cloud controller	1
Total number of physical machines	4
Average number of running instances on each host	10
Total number of running instances	40
Incident description	
Incident type	Inappropriate Usage
Current status	Ongoing attack, the bogus compute worker is still up and serving a part of requests
Physical Location	OpenStack-5
Affected Layers	Cloud platform layer, the OpenStack nova-compute service, consumers' instances
General Information	A bogus compute worker node is added to the platform, it is a threat to the provider's and consumers' data confidentiality and integrity. Also a threat for the system availability.
Resources at risk	Running instances on OpenStack-5, Stakeholders and resources interacting with running instance on OpenStack-5

Table 6.3: Case Two - A bogus component scenario specifications

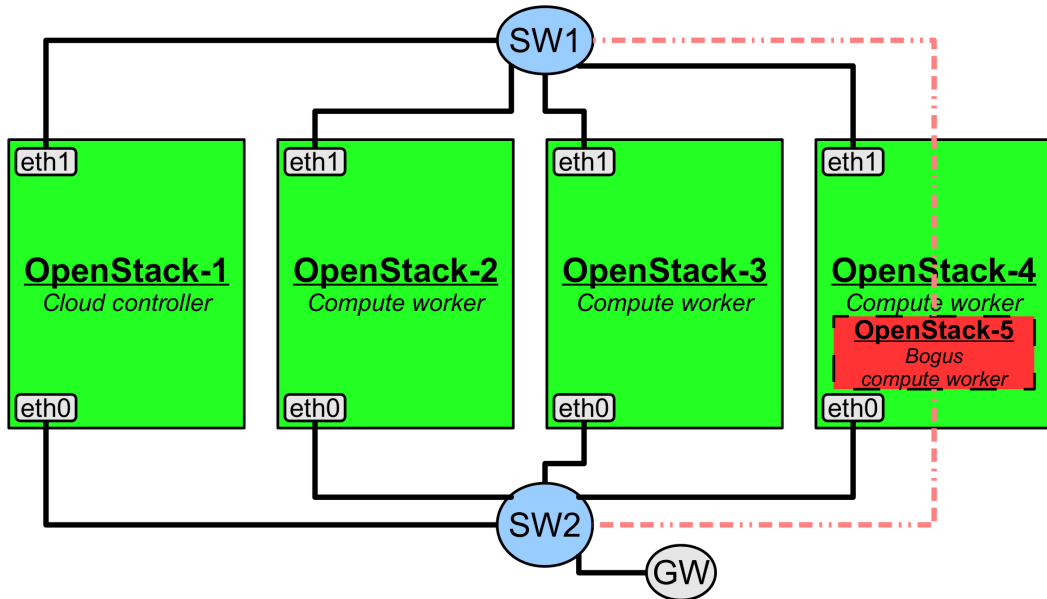


Figure 6.8: Case Two - A virtual bogus compute worker is added as a consumer's instance.

by means of the bridge connection (*br100*) that connects virtual interfaces (*vnetX*) to the rest of the environment. Thus, a running instance has no connectivity to the *SW2* by default.

However, connectivity to the outside world can be requested by any consumer (e.g. an attacker) through a legitimate procedure. Thus, in Figure 6.8, we also connect the instance to the *SW2*.

Moreover, such a scenario is more applicable to a cloud environment that is using an open source platform (e.g. OpenStack, OpenNebula, Eucalyptus, etc.), because an attacker can find services' source code and binaries easily.

We simulate the virtual bogus compute worker by deploying the *nova-compute* service on a running instance. Details of this deployment is explained in Chapter 8. There were multiple obstacles for simulating this scenario, including: the running instance, which turns to be also a bogus worker, must have the hosting capabilities; the bogus worker must respond to cloud controller requests to be recognized as a working node.

After the installation and configuration is completed, we can check to see if the cloud controller recognized the new worker (i.e. bogus compute worker) using the *nova-manage* utility as depicted in Listing 6.4. The instance ID is

6.2. Containment, Eradication, and Recovery in an OpenStack Deployment

192.168.0.2:60856	227B/s (218.1kB total)	128B/s (123.5kB total)	1	guest	running
192.168.0.2:60857	335B/s (322.3kB total)	128B/s (123.5kB total)	1	guest	running
192.168.0.2:60858	624B/s (597.1kB total)	129B/s (123.5kB total)	1	guest	running

Figure 6.9: Case Two - Bogus worker's connections are established.



Figure 6.10: Case Two - Platform wide exchanges are binded to the bogus worker.

*i-00000036*¹⁹ and its IP address is *192.168.0.2*.

Listing 6.4: List of services in the environment

```
root@openstack-1:~# nova-manage service list
openstack-1 nova-network enabled :-) 2011-05-30 14:56
openstack-1 nova-compute enabled :-) 2011-05-30 14:56
openstack-1 nova-scheduler enabled :-) 2011-05-30 14:5
openstack-2 nova-compute disabled XXX 2011-05-30 15:00
openstack-3 nova-compute enabled XXX 2011-05-30 14:55
openstack-4 nova-compute enabled XXX 2011-05-30 14:54
i-00000036 nova-compute enabled :-) 2011-05-30 14:56
```

When the bogus worker is running, its corresponding queues and bindings are also created, Figures, 6.9 and 6.10.

Detection

Detecting a bogus worker node or instance is a complex task, if the infrastructure has not previously employed a proper set of mechanisms. However, a few parameters can be monitored as an indication of a bogus worker.

Generally, a bogus worker is not working as well as a real one, because its main goal is not providing a regular service. A bogus worker aims to

¹⁹In Figure 6.8 instead of *i-00000036* we used *OpenStack-5* ID, to make it more readable.

steal consumers' data, intrude on the cloud infrastructure, disrupt the cloud environment QoS, and so forth. Without any prior preparation a suspicious worker can be identified by monitoring the service availability and QoS parameters on each worker. Moreover a suspicious virtual worker can also be recognized because of its high traffic towards the cloud infrastructure messaging servers.

Containment

Containing a bogus worker consists of both proactive and reactive techniques. When a bogus worker is detected the containment procedure is fairly simple (i.e. applying reactive techniques). However, deploying a set of proactive techniques is more challenging.

- **Cryptographic mechanisms**

In this method each worker must have a certificate signed by a trusted authority. This authority can be either an external one or the cloud controller/authentication manager itself. Having a signed certificate, the worker can communicate with other components securely. The secure communication can bring us any of the following: confidentiality, integrity, authentication, and non-reputation.

In this case, worker's communication and authenticity is important for us. For this purpose we can use two different schemes: message encryption or a signature scheme. Each of these schemes can be used for the whole communication or the handshake phase only.

When any of those schemes are applied only to the handshake phase, any disconnection or timeout in the communication is a threat to the trust relation. As an authenticated worker is disconnected and reconnected, we cannot only rely on the worker's ID or host-name to presume it as the trusted one. Thus, the handshake phase should be repeated to ensure the authenticity of the worker.

Although applying each scheme to all messages among cloud components is tolerant against disruption and disconnection, its overhead for the system and the demand for it should be studied case by case.

By applying each of those schemes to all messages, we can tolerate disconnection and disruption. However, using cryptographic techniques for all messages introduce an overhead for the system which may not be efficient or acceptable.

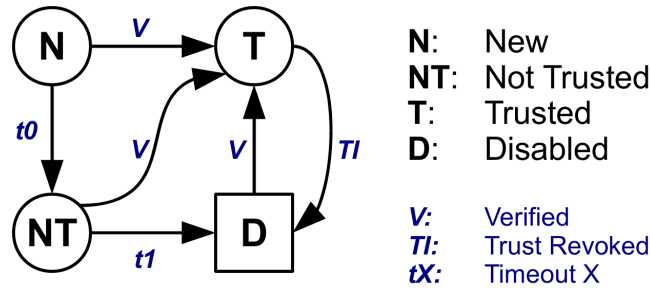


Figure 6.11: A sample markov model for trust states of a component.

Implementing this method in our environment is simple. The RabbitMQ has features that facilitate communication encryption and client authentication. The *RabbitMQ SSL support* offers encrypted communication [14].

Moreover, an authentication mechanism using the client SSL certificate is offered by the *rabbitmq-auth-mechanism-ssl* plugin [47].

- **Manual confirmation**

In this method, recently added workers are not used for serving consumers' requests until their authenticity is confirmed by the cloud provider. This method requires human intervention; thus, it can become a bottleneck in the cloud infrastructure. Techniques, explained in the next part, can relax the bottleneck issue.

- **Trust levels and timeouts**

Introducing a set of trust levels, a new worker can be labeled as a not trusted worker. Workers which are not trusted yet, can be used for hosting non-critical instances, or can offer a cheaper service to consumers.

In order to ensure the system trustworthiness in a long run, a not-trusted worker will be disabled after a timeout. A simple Markov model of those transitions are depicted in Figure 6.11.

Assuming we have only two trust levels, Figure 6.12 depicts transitions between them. As an example, *T0* can be achieved by the human intervention; and the second level of trust *T1* is gained by cryptographic techniques or trusted computing mechanisms.

- **No new worker policy**

In addition to all those technical approaches, a set of management

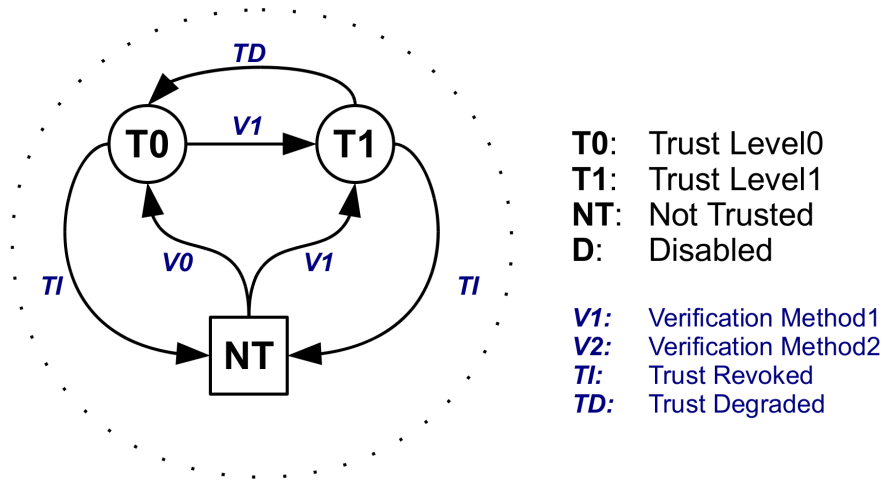


Figure 6.12: A sample markov model for transitions between different trust levels of a component.

policies can also relax the issue. As an example, no new worker should be added unless there is a demand for it. The demand for a new worker can be determined when the resource utilization for each zone is above a given threshold.

Chapter 7

New Approaches

This section introduces our proposed approaches for containment, eradication and recovery. Proposed strategies are grouped based on two criteria, the responsible stakeholder for developing and deploying the strategy, and the target layer for that strategy.

Based on the first criterion we may have either cloud provider or cloud consumer as the responsible stakeholder. And based on the second criterion, the target layer can be either infrastructure/hardware layer or service/application layer.

7.1 Restriction of Infected Components

A general technique for containing an incident is restricting the infected component. The restriction can be applied in different layers, with a variety of approaches, such as: filtering in the AMQP server, filtering in other components, disabling the infected service or the communicator service (i.e. the service that handles the communication among a set of components). Additional measures can also be employed to support the restriction, like: removing infected instances from the project VLAN, disabling live migration, or quarantining infected instances.

We explain each of these approaches in the following sections.

7.1.1 Filtering in the messaging server (cloud controller)

We will propose several filtering mechanisms in the messaging server in order to contain and eradicate an incident in a cloud environment. The OpenStack platform has been used to build our experimental cloud environment. This approach is a responsibility of the cloud provider and the target layer in the cloud platform application layer.

The filtering in the messaging server has its own advantages and disadvantages, which will be discussed next.

Advantages

- The filtering task at the messaging server level can be done without implementation of new functionalities. We can use existing management interfaces of the RabbitMQ (either Command Line Interface (CLI) or web interface) to filter the compromised component.

However, in a large scale deployment of the platform, the situation may vary. When automation and real-time responses are crucial, we have to avoid mechanisms which require human intervention. Even in this case we should only implement a set of functionalities that uses management interfaces for filtering. Thus, instead of an operator who terminates a connection manually, the cloud controller will do that when it is required.

- The filtering task can be done in a centralized fashion by means of the management plug-in, although we may have multiple instances of the messaging server.
- Implementing this approach is completely transparent for other stakeholders, such as cloud consumers.
- We can scale out²⁰ the messaging capability by running multiple instance of the RabbitMQ on different nodes. Scaling out the messaging server will also scale out the filtering mechanism²¹.
- This approach is at the application layer, and it is independent of network architecture and employed hardware.
- The implementation at the messaging server level helps in having a fine-grained filtering, based on the message content.

Disadvantages

- A centralized approach has its own disadvantages as well, such as being a single point of failure or becoming the system bottleneck.
- Implementing the filtering mechanism at the messaging server and/or the cloud controller adds an extra complexity to these components.

²⁰Scaling out or horizontal scaling is referred to the application deployment on multiple servers [49].

²¹But it may require a correlation entity to handle the filtering tasks among all messaging servers.

- When messages are filtered at the application layer in the RabbitMQ server, the network bandwidth is already wasted for the message that has an infected source, destination, or even context. Thus, this approach is less efficient compared to the one that may filter the message sooner (e.g. at its source host, or in the source cluster)
- Most of the time application layer approaches are not as fast as hardware layer one. In a large scale and distributed environment the operation speed plays a vital role in the system availability and QoS.

It is possible to use the zFilter technique as a more efficient implementation of the message delivery technique. It can be implemented on either software or hardware. The zFilter is based on the bloom-filter data structure. Each message contains its state; thus this technique is stateless [43]. It also utilizes source routing. zFilter implementations are available for the BSD family operating systems and the NetFPGA boards in the following address, <http://www.psirp.org>.

- Filtering a message without notifying upper layers, may lead to timeout trigger and resend requests from waiting entities. It can also cause more wasted bandwidth.

A variety of filtering mechanisms can be utilized in the messaging server; each of these mechanisms focuses on a specific component/concept in the RabbitMQ messaging server. We can enforce the filtering in messaging server *connection*, *exchange*, and *queue* that will be discussed next.

Connection

A connection is created to connect a client to an AMQP broker [13]. A connection is a long-lasting communication capability and may contain multiple channels [79]. By closing the connection all of its channels will be closed as well. The list of connections in our OpenStack deployment is available in Figure 8.15. Details of each connection and its corresponding channels are also available using the management interface, as depicted in Figure 8.16.

First approach to block the compromised component is closing its client connection. Closing the connection will stop all channels in that connection.

In order to close a connection we do the following steps:

1. Browse management page (i.e. <http://openstack-1:55672/mgmt/>).
2. Go to "Connections".

Connection 129.241.252.119:49256

▼ Overview

From client	623B/s	Username	guest
To client	128B/s	Protocol	0-8
		SSL	□
		Authentication	AMQPLAIN
		State	running
		Timeout	0

▼ Channels

Channel	Details				
	Mode (?)	Prefetch	Unacked	Unconfirmed	Status
129.241.252.119:49256:1		0	0	0	Active

▶ Client Library

▼ Close This Connection

Force Close

Figure 7.1: Closing a connection using RabbitMQ management

3. Select the target connection using its peer address.
4. Choose "Close This Connection" from the connection details page and press "Force Close". (Check Figure 7.1)

We can also use either the CLI interface or HTTP API for this purpose. The CLI command is as follow:

Listing 7.1: Closing a connection using RabbitMQ command line interface

```
# rabbitmqadmin close connection \
    name=<CONNECTION NAME>
```

Channel

A channel is created using a connection, and it can be used to send and receive messages [13]. However, we can not manipulate a single channel, instead we can close the container connection of that specific channel.

Exchange

An exchange is a message routing agent which can be durable, temporary, or auto-deleted. Messages are routed to qualified queues by the exchange. A Binding is a link between an exchange and a queue. An exchange type can be one of *direct*, *topic*, *headers*, or *fanout*. [66]

1. **direct:** If the message routing key was equal to the binded queue routing key, the message is passed to the message queue [79].
2. **topic:** This exchange has the same behavior as the direct one with a small difference. The message key should match the queue routing *pattern* to be passed to that queue.
3. **headers:** In this exchange the header properties of the arrived message should match the queue properties. It is not used in our environment, so we skip more details. (Check [17] for more information.)
4. **fanout:** This exchange broadcasts incoming messages to all queues that are interested in this specific exchange (i.e. Queues that are binded to this exchange).

An exchange can be manipulated in different ways in order to provide a filter mechanisms for our cloud environment.

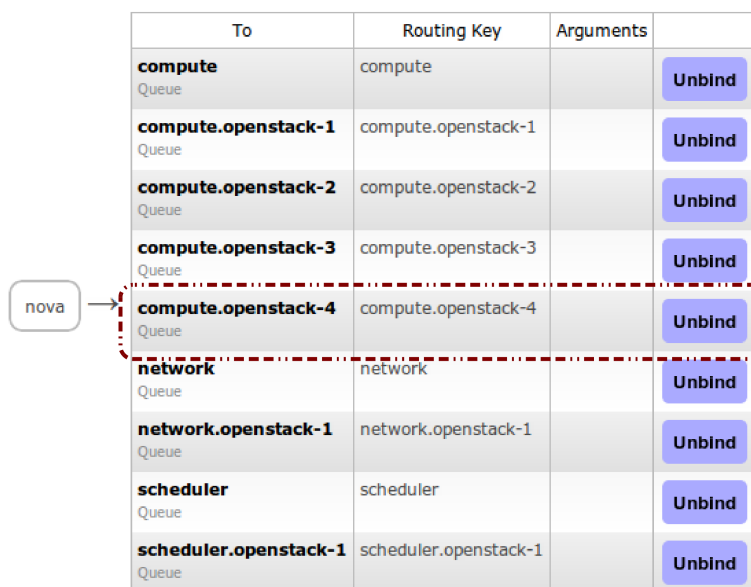
- **Unbinding a queue from the exchange**

The compromised component queue won't receive messages from the unbinded exchange.

As an example, we assume that the compute service of the OpenStack-4 host is compromised. Now, we want to block nova traffic to and from the compromised compute service; so, we unbind the NOVA topic exchange from the queue COMPUTE.OPENSTACK-4.

1. Browse management page (i.e. <http://openstack-1:55672/mgmt/>).
2. Go to "*Exchanges*".
3. Select the target exchange based on its name and type.
4. In the "*Bindings*" section, unbind the target queue (e.g. COMPUTE.OPENSTACK-4). (Check Figure 7.2)

7.1. Restriction of Infected Components



To	Routing Key	Arguments	
compute Queue	compute		Unbind
compute.openstack-1 Queue	compute.openstack-1		Unbind
compute.openstack-2 Queue	compute.openstack-2		Unbind
compute.openstack-3 Queue	compute.openstack-3		Unbind
compute.openstack-4 Queue	compute.openstack-4		Unbind
network Queue	network		Unbind
network.openstack-1 Queue	network.openstack-1		Unbind
scheduler Queue	scheduler		Unbind
scheduler.openstack-1 Queue	scheduler.openstack-1		Unbind

Figure 7.2: Unbinding a queue from an exchange using the Queues Management page of the RabbitMQ

- **Publishing a warning message**

Publishing an alert message to that exchange, so all clients using that exchange will be informed about the compromised component. Thus, by specifying the compromised component, other clients can avoid communicating with it. The main obstacle in this technique is the requirement for implementing new functionalities in clients.

Publishing a new message to the exchange contains following steps:

1. Browse management page (i.e. <http://openstack-1:55672/mgmt/>).
2. Go to "Exchanges".
3. Select the target exchange based on its name and type.
4. In the "Publish Message" section, enter the same routing key a (e.g. COMPUTE.OPENSTACK-1), fill other fields and publish the message. Then repeat the same procedure for the rest of related queues (e.g. COMPUTE.OPENSTACK-2, COMPUTE.OPENSTACK-3)(Check Figure 7.3)

- **Deleting the exchange**

Deleting an exchange will stop routing of messages related to it. It

The screenshot shows the 'Publish Message' section of the RabbitMQ management interface. It includes the following fields and controls:

- Routing key:** A text input field.
- Delivery mode:** A dropdown menu currently set to '1 - Non-persistent'.
- Headers:** A label with a question mark icon, followed by a text input field, an equals sign, and another text input field.
- Properties:** A label with a question mark icon, followed by a text input field, an equals sign, and another text input field.
- Payload:** A large, empty text area for entering the message content.
- Publish Message:** A blue button at the bottom left of the form.

Figure 7.3: Publishing a message to a queue using RabbitMQ management

may have multiple side effects, such as memory overflow and queue exhaustion.

In the *"Delete This Exchange"* section, click *"Delete"* (Check Figure 7.4).

Queue

A queue is called a "weak First In First Out (FIFO)" buffer, that each message in it can be delivered only to a single client unless re-queuing the message [66].

- **Unbinding**

Unbinding a queue from an exchange avoids further routing of messages from that exchange to the unbind-ed queue. We can unbind the queue which is connected to the compromised component and stop receiving messages by the infected client. (Check Figure 7.5)

- **Deleting**

Deleting a queue not only removes the queue itself, but also remove all

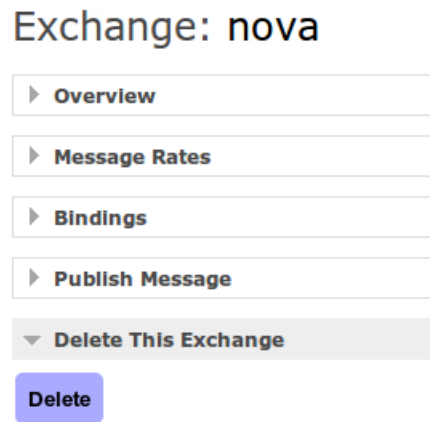


Figure 7.4: Deleting an exchange using RabbitMQ management

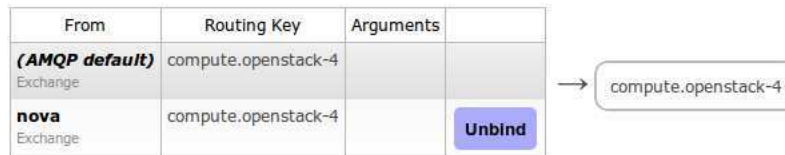


Figure 7.5: Unbinding a queue from an exchange using RabbitMQ management



Figure 7.6: Deleting or purging a queue using RabbitMQ management

messages in the queue and cancel all consumers on that queue. (Check Figure 7.6)

- **Purging**

Purging a queue removes all messages in the queue that do not need acknowledgment. Although it may be useful in some cases, it may not be as effective as required in occurrence of an incident. (Check Figure 7.6)

Figure 7.7²² depicts an overview of a messaging server internal entities and the application points of our approaches.

It should be noted that each of previously mentioned procedures can also be done using either the CLI or HTTP API interfaces. You can find more information about them on our management server, <http://openstack-1:55672>.

7.1.2 Filtering in each component

Applicable filtering mechanisms in the messaging server have been studied in the previous section. This section discusses mechanisms that are appropriate for other components. These components are not essentially aware of messaging technique details and specifications.

Advantages

- The implementation of the filtering mechanism in each component avoids added complexity to the messaging server and cloud controller.
- This approach is a distributed solution without a single point of failure in contrast to the previous one with a centralized filtering mechanism.

²²Multiple details have been avoided in this figure to make it more readable, such as Virtual Host.

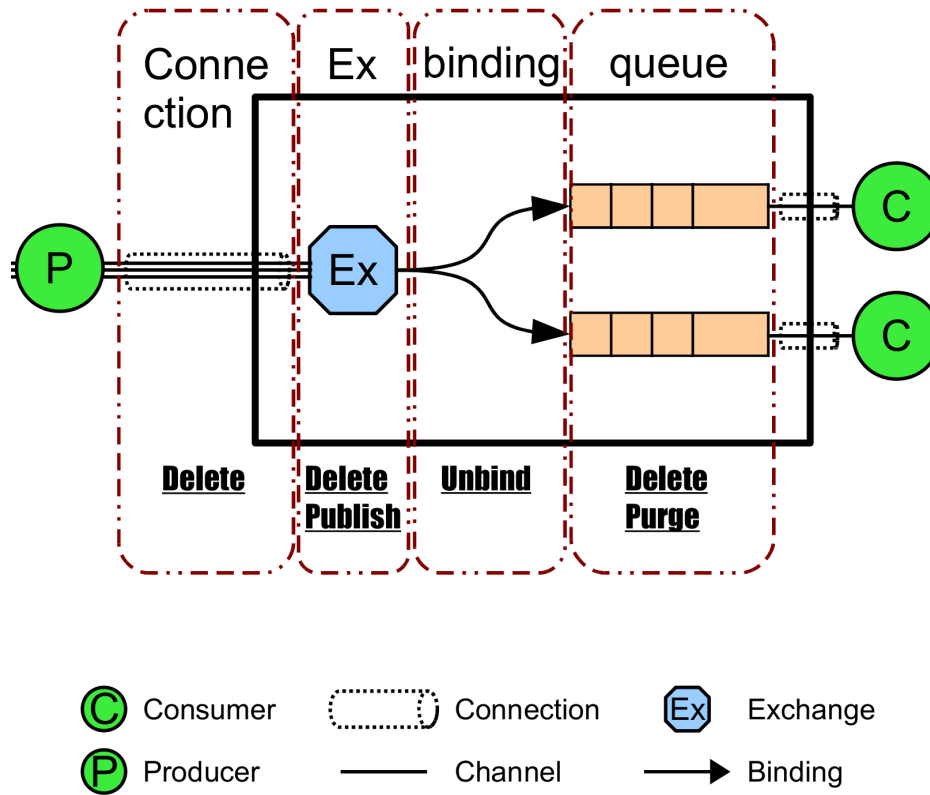


Figure 7.7: Overview of RabbitMQ messaging server and applicable containment approaches.

- Assuming locality principle in the cloud, wasted bandwidth is limited into a cluster/rack which host the infected components. Network connections have much higher speed in a rack or cluster.
- This approach does not require a correlation/coordination entity for filtering messages. Each component behaves independently and autonomously upon receiving an alarm message, that announces a compromised node.

As there is no boundary in the cloud, performing security enforcement at each component is a more reliable approach. Traditionally, most security mechanisms have been employed at the organization/system boundaries. However, as the realization of boundaries is becoming weaker in a cloud environment, this approach is a reasonable one to fulfill the new requirements.

Disadvantages

- When the filtering must be performed in each component, all interacting components must be modified to support the filtering mechanism. However, this issue can be relaxed by using a unified version of messaging client (e.g. pika python client) and modifying the client in case of new requirements.
- The message which should be discarded traverses all the way down to the destination, and wastes the link bandwidth on its route.
- Dropping a message without notifying upper layers, may lead to timeout trigger and resend requests from waiting entities. It can also cause more wasted bandwidth.

Implementation

This approach can be implemented at two different levels: blocking at either the messaging client level (e.g. AMQP messaging client) or the OpenStack component/service level, Figure 7.8.

First, the responsible client can be modified to drop messages with specific properties (e.g. infected source/destination). As an example, the responsible client for AMQP messaging in the OpenStack is `amqp-lib/pika`; we must implement the mechanism in this AMQP client (or its wrapper in the OpenStack) to filter malicious AMQP messages. Using this method, more interaction between the OpenStack and clients may be required to avoid

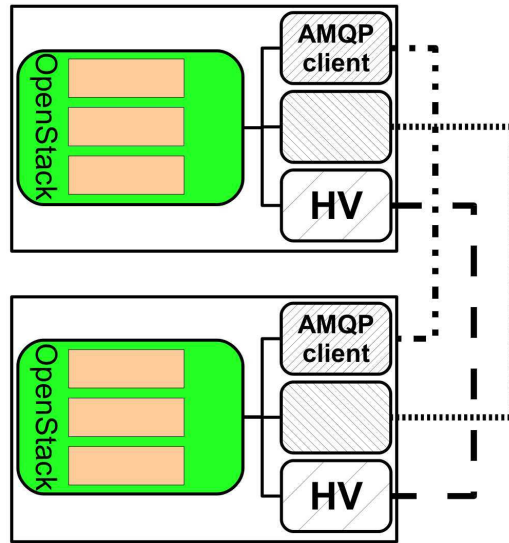


Figure 7.8: Overview of possible filtering points in each component

resend requests. Because of using the same AMQP client in all components, the implementation is easier and its modification process needs less effort.

The second method is to develop the filtering in each of the OpenStack components, such as nova-compute, nova-network, nova-scheduler, etc. This method adds more complexity to those components and it may not be part of their responsibilities.

We propose a combination of these methods. Implementing the filtering mechanism in the carrot/amqplib wrapper of the OpenStack has advantages of both methods and avoids unnecessary complexity. The OpenStack wrapper for managing AMQP messaging is implemented in `src/nova/rpc.py`. In order to identify the malicious message, we use the message address which is part of its context. Then, the actual dropping happens in the *Adapter-Consumer* method. Assuming that the source address is set in the context variable, filtering is straight forward. By checking the message address and avoiding the method call, most of the task is done. The only remaining part is to inform the sender about the problem, that can be implemented by means of the existing message reply functionality.

In addition to this modification another feature should be added to handle the list of compromised components.

7.1.3 Disabling services

Disabling services is a strategy for containing the incident. The disabled service can be either the infected or the communicator one. The communicator service handles tasks distribution and delegation.

This method can be used only by the cloud provider, and is at the application layer.

Disabling an infected service

An incident can be contained by disabling the infected service. It has several advantages, including:

- After stopping the nova-compute service, running instances will continue to work. Thus, as a result consumers' instances will not be terminated nor disrupted.
- All communications to and from the compromised node will be stopped. So, the wasted bandwidth will be reduced massively.
- Shutting down a service gracefully, avoids an extra set of failures. When the service is stopped by Nova interfaces, all other components will be notified and the compromised node will be removed from the list of available compute workers.

Like any other solution, it has multiple drawbacks as well, including:

- Keeping instances in the running status can threaten cloud consumers. The attacker may gain access to running instances on the compromised node.
- The live migration feature will not work anymore. Thus, the threatened consumers cannot migrate running instances to a safe or quarantine compute worker node.
- Neither the cloud provider nor consumers can manage running instances through the OpenStack platform.

In order to stop the nova-compute service, we must have an interface to the compromised node operating system (i.e. either a ssh connection or access to the physical node). Then we can stop the service using the following command (Listing 7.2) :

Listing 7.2: Disabling the nova-compute service

```
root@<COMPROMISED NODE>:~# service nova-compute stop
```

This approach requires no further implementation, although we may like to add a mechanisms to turn services on and off remotely.

Disabling a communicator service

An incident can be contained by disabling or modifying its corresponding communicator service. An example of a communicator service in an OpenStack deployment nova-scheduler service. The nova-scheduler decides that which worker should handle the newly arrived request, such as running an instance.

By adding new features to the scheduler service, the platform can avoid forwarding request to the compromised node.

Advantages of this approach are:

- No more requests will be forwarded to the compromised node.
- Consumers' instances remain in the running status on the compromised node. So, consumers will have enough time to migrate their instances to a quarantine worker node or dispose their critical data. Even estimate impacts of the incident.
- This approach can be used to identify the attackers, hidden system vulnerabilities, and the set of employed exploits. In other words, it can be used for forensic purposes.

And its disadvantages are:

- New features should be implemented. These new features are more focused on the decision algorithm of the scheduler service.
- This approach will not secure the rest of our cloud environment, but it avoids forwarding new requests to the compromised node. However, this drawback can be seen as an opportunity. We can apply this approach and also move the compromised node to a **HoneyCloud**. In the HoneyCloud we don't restrict the compromised node, instead analyze the attack and attacker's behavior. But even by moving the compromised node to a HoneyCloud, hosted instances on that node are still in danger.

It is possible that consumers' instances are all interconnected. Thus, those running instances, on the compromised node in the HoneyCloud,

threaten the rest of consumers' instances. The rest of instances may even be hosted on a secure worker node. The next proposed approach is a solution for this issue.

7.1.4 Removing instances from the project VLAN

This approach does not contain the compromised node, instead focuses on containing instances hosted by the compromised worker node. This is important because those instances may have been compromised as well. The first step toward securing the consumer's service is to disconnect potentially infected instances.

The main usecase of this approach is when the attacker disrupts other solutions (i.e. disabling nova-compute management functionalities, escalated privileges at the OS layer), or when instances and the consumer's service security is very important (e.g. eGovernment services).

It can be done by either the cloud provider or cloud consumers. Although it can also be done at the network layer, we focus on the cloud platform VLAN management capabilities.

It has several advantages specifically for cloud consumers, including:

- Disconnect potentially infected instances from the rest of consumer's instance.
- It does not require implementation of new features.
- The attacker cannot disrupt this method.

And its disadvantages are as follows:

- This method only works in a specific OpenStack networking mode (i.e. VLANManager networking mode).
- The consumer completely loses control over isolated instances, that may lead to data loss or disclosure, service unavailability, etc.

7.1.5 Locking down instances' live migration

Live migration can cause wide-spread infection, or can be a mechanism for further intrusion to a cloud environment. It may take place intentionally or unintentionally (e.g. an affected consumer may migrate instances to resolve the attack side effects, or the attacker that has the consumer privileges migrates instances to use a hypervisor vulnerability and gain control over more nodes).

Disabling this feature helps the cloud provider to contain the incident more easily, and keep the rest of the environment safer.

7.1.6 Quarantining instances

When we migrate instances from a compromised node, we cannot accept the risk of spreading infection along instance migration. Thus, we should move them to a quarantine worker node first. The quarantine worker node has specific functionalities and tasks, including:

- This worker node limits instances connectivity with the rest of cloud environment. As an example, only cloud management requests/responses are delivered by the quarantine host.
- It has a set of mechanisms to check instances' integrity and healthiness. These mechanisms can be provided by the underlying hypervisor, cloud platform, or third parties' services.

In order to deploy a quarantine node, we should study and employ a set of mechanisms. In most cases, we will introduce the appropriate tool that has implemented the mechanism.

1. Virtual Machine Introspection

This mechanism simplifies introspecting the memory space of a virtual machine from another virtual machine. The task is fairly complex because of the semantic gap between the memory space of those two virtual machines.

The XenAccess is an example of introspection library. Using XenAccess the privileged domain can monitor another Xen domain.

2. Domain Monitoring

One of the basic methods to identify a compromised instance is by means of profiling and monitoring the instance behavior. Domain monitoring techniques provide an abstract set of data, comparing to the detailed, low level output of a VM introspection tool.

For a virtual machine running over a Linux box we can use the libvirt [7] library to access the suspicious instance and study its behavior. We will mention a set of commands that uses libvirt and provide information about instances and their statistics:

Listing 7.3: libvirt options for monitoring instances

7.1. Restriction of Infected Components

```
# virsh help monitor
Domain Monitoring (help keyword 'monitor'):
  domblkinfo  domain block device size information
  domblkstat  get device block stats for a domain
  domifstat   get network interface stats
  dominfo     domain information
  dommemstat  get memory statistics for a domain
  domstate    domain state
  list        list domains
```

Listing 7.4: Monitoring network interface statistics of an instance

```
# virsh domifstat instance-00000017 vnet0
vnet0 rx_bytes 146252399
vnet0 rx_packets 483266
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 15022675
vnet0 tx_packets 60770
vnet0 tx_errs 0
vnet0 tx_drop 0
```

Another tool provided by the libvirt library is VIRT-TOP, that is a TOP like utility for virtualization stats in a GNU/Linux environment. An illustration of this tool is given in the following list.

Listing 7.5: virt-top sample output

```
# virt-top
virt-top 18:57:05 - x86_64 4/4CPU 1600MHz 3877MB
6 domains, 5 active, 5 running, 0 sleeping, 0 paused
1 inactive D:0 O:0 X:0
CPU: 0.0% Mem: 2560 MB (2560 MB by guests)

  ID S RDRQ WRRQ RXBY TXBY %CPU %MEM TIME NAME
  5 R 0 0 0 0 0.0 13.0 8:39.37 i-17
  6 R 0 0 0 0 0.0 13.0 5:43.32 i-1b
  7 R 0 0 0 0 0.0 13.0 5:47.03 i-1c
  8 R 0 0 0 0 0.0 13.0 5:47.97 i-1e
  9 R 0 0 0 0 0.0 13.0 5:49.40 i-1f
  -                                     (i-03)
```

ID	S	RXBY	TXBY	RXPK	TXPK	DOMAIN	INTERFACE
5	R	86	0	1	0	instance-000	vnet0
6	R	86	0	1	0	instance-000	vnet1
7	R	86	0	1	0	instance-000	vnet2
8	R	86	0	1	0	instance-000	vnet3
9	R	86	0	1	0	instance-000	vnet4

3. Intrusion Detection

Having an intrusion detection system in the hypervisor or cloud platform layer not only provide better visibility for security mechanisms but also is more resistant against a targeted attack from an unauthorized access to an instance. Livewire [37] is a prototype implementation of an intrusion detection system in a hypervisor.

Another way to benefit from intrusion detection system is the same Amazon's approach. They offer you a standalone Amazon Machine Image (AMI) that contains Snort and Sourcefire Vulnerability Research Team rules. Then the consumer can forward its instances traffic to the virtual machine with intrusion detection capabilities. Same approach can be utilized in our deployment. The main issue is the approach performance and utilization.

4. Utilizing trusted computing concepts

Trusted computing is a technology for ensuring the confidentiality and integrity of a computation. Moreover it is useful for remote attestation. Thus, we can use the technology not only for securing our deployment but also to build a better quarantine and infection analysis mechanism.

There are several debates about its implications. The technology can be used against software and hardware owners. Richard M. Stallman says:

*"Trusted computing is the proponents' name for a scheme to redesign computers so that application developers can trust your computer to obey them instead of you. From their point of view, it is **trusted**; from your point of view, it is **treacherous**."*

There is a Frequently Asked Questions page [18] by Ross Anderson that covers most issues about trusted computing.

In the following we will introduce multiple techniques that uses the trusted computing as their core technology and are useful for our environment.

vTPM: Virtualizing the Trusted Platform Module [25]

Berger et al. propose the idea of a virtual trusted platform module. The vTPM not only provides trusted computing services for an unlimited number of instances, but also deliver extra security services such as remote attestation of an instance integrity [25].

”In this context, attestation means to affirm that some software or hardware is genuine or correct.”

The vTPM idea has been implemented in the Xen hypervisor by the same team and IBM researchers²³. Nguyen Anh Quynh has been working on the KVM implementation and integration of vTPM, but it is not released yet. Thus, using the Xen hypervisor, we can check an instance in the quarantine node to ensure its integrity.

TCCP: Trusted Cloud Computing Platform [67]

Santos et al. introduce the Trusted Cloud Computing Platform (TCCP). The initial purpose of TCCP is to ensure the confidentiality and integrity of a cloud consumer’s computations. It is also possible to do the remote attestation procedure using this technique, thus it is useful to be considered as an effective method for ensuring an instance integrity [67]. However, it has severe drawbacks that make the application absurd, including: it is based on a trusted hypervisor (Terra is used in their prototype implementation); and an external trusted entity is required. Clearly, these obstacles can be relaxed by implementing a Trusted Virtual Machine Monitor (TVMM) and using a CA as the external trusted entity.

TVDC: IBM Trusted Virtual Datacenter [26]

The research of Berger et al. on trusted virtual datacenters are also worth noting. Its main feature for a cloud environment is the strong isolation among resources and instances. This isolation is realized by means of the Trusted Virtual Domain (TVD) [26]. It can be used not only for isolating consumers’ projects from each other but also to implement a reliable HoneyCloud.

²³They are also authors of the original paper [25].

Implementing this approach in our cloud environment requires an extensive work. Moreover, the performance of this technique should be studied in a large scale cloud environment to understand its implications.

It should be noted that although cloud providers or third party service providers can offer an IDS agent service inside each instance, they cannot force the consumer for accepting it. It is a reasonable argument due to consumer's organization internal security policies and resource overhead because of the security agent. Thus, applying security services to underlying layer (i.e. hypervisor, cloud platform) is a preferred solution.

Detailed specifications of such a compute worker node is a great opportunity for future work.

7.2 Replication of Services

An approach to overcome the implications of an incident is replicating services. A service in this section is a service which is delivered and maintained by the cloud provider. It can be a cloud platform service (e.g nova-compute) or any other services that concerns other stakeholders. The replication can be done passively or actively, and that is due to new characteristics of the cloud model.

7.2.1 Replication Layers

The replication of a cloud service can be done either at the physical or virtual machine layer.

Replicate services on physical machines

Replicating service on physical machines is already done in a platform such as the OpenStack. The provider can replicate cloud services either passively or actively when facing an issue in the environment.

Replicate services on virtual machines

Replication of service on virtual machines has multiple benefits, including:

1. Virtual machines can be migrated while running (i.e. live migration), this is a practical mechanism for stateful services that use memory.

2. Replication at the instance layer is helpful for forensics purposes. It is also possible to move the compromised service in conjunction with the underlying instance to a HoneyCloud. This is done instead of moving the physical node, ceasing all services on it, and changing the network configuration in order to restrict the compromised node communication.
3. Using virtual machines in a cloud environment we can also benefit from the cloud model elasticity and on demand access to computing resources.

This approach is also the main idea behind the CC-VIT (Section 6.1.1). By applying the CC-VIT to our environment, the preferred hybrid fault model will be REMH, and the group communication is handle using the AMQP messaging.

We can use physical-to-virtual converters to have the advantages of both approaches. These tools convert a physical machine to a virtual machine image/instance that can be run on top of a hypervisor.

Moreover, each of these replicas can be either active or passive. This will have a great impact on the system availability.

7.3 Disinfection of Infected Components

Disinfecting an infected component is a crucial task in handling an incident and securing the system. It can be accomplished with multiple methods having a variety of specifications.

None of the following approaches will be used for cleaning the infected binary files, instead less complex techniques are employed that can be applied in a highly distributed environment. Cleaning a binary file can be offered by a third party security service provider, that has focused on large scale antivirus software.

1. **Updating the code**

The service code can be updated to the latest, patched version. This process should be done in a smooth way so all components will be either updated or remain compatible with each other after partial components update.

Several tools has been developed with this purpose. One of the best examples is the Puppet project [11].

2. **Purging the infected service**

Assuming that the attacker has stopped at the cloud platform layer, by removing the service completely we can assure containment of the incident.

3. **Replacing the service**

Another method which is not as strong as others, is achieved by replacing the infected service with another one that uses a different set of application layer resources, such as configuration files, binaries, etc. Thus, we can be sure that the infected resources have no effect on the new service.

7.4 Consumer Approaches

This section will introduce a couple of approaches that can be used by a consumer in case of an incident. They are categorized in two groups, reactive and proactive approaches.

7.4.1 Reactive

Reactive approaches are applied when the consumer become aware of an incident, and they don't require a previous action. The instance migration and component disinfection are two reactive methods that will be discussed.

Migration

The affected consumer can migrate an specific instance or a set of instances to another compute worker or even another cloud environment. The migration among different provider is an open challenge nowadays, because of the weak interoperability of cloud systems and lack of standard interfaces for cloud services.

In our deployment, both Amazon Elastic Compute Cloud (EC2) APIs and RackSpace APIs are supported. Thus, in theory a consumer can move between any cloud environment provided by the Amazon EC2, RackSpace, and any open deployment of OpenStack without any problem.

Disinfection

The same methods as discussed in Section 7.3 can be applied here.

7.4.2 Proactive

Proactive approaches require a set of previously done actions. Two proactive methods are mentioned in the following, comprising: services replication and application of trusted computing technology.

Replicate services

As discussed in Section 7.2, a consumer can also replicate its services. This replication may be done in either a single cloud environment or among several cloud providers' environments.

Trusted Computing

Assuming that the underlying platform support trusted computing, a consumer can use an external trusted entity to ensure the confidentiality and integrity of computation, as explained in [67] and [25] (Section 7.1.6).

Chapter 8

Lab Setup

As we discussed before, we have found the OpenStack cloud platform as the best choice for a real case study in our research. But choosing the OpenStack has its own disadvantages as well. The fact that our first experience was only six months after its first release, and lack of documentation, introduces several challenges in our study.

Installation, configuration and administration of the OpenStack in a useful way for our research is a part of our thesis contributions. It should be noted while experimenting with the OpenStack, several bugs and issues have been identified in the platform and reported to the community. Additionally, many other design improvements have been discussed in the community which are direct outcomes of our deep study in the OpenStack platform and previous identification of a cloud platform requirements.

8.1 OpenStack Compute Deployment

This section will study installation and configuration of OpenStack Compute project (Nova). We will use a variety of references, including: *OpenStack Compute Administration Manual*[56], OpenStack Wiki page[85] and experiences from communicating with OpenStack community at IRC channel #openstack (irc.freenode.org).

We explain a few terms that are going to be used in the rest of our report and may be ambiguous for readers:

In the OpenStack cloud environment each cloud consumer has its own *project*. Access to VM images is restricted by these projects. A project defines the territory of a cloud consumer which can be file system quota, bandwidth quota, etc. A project can have several *users* with different roles. A *role* restricts users' activity and define her/his privileges. *Key-pairs* are assigned to each user and facilitate the user's access to the cloud resources. A *VM image* can have a kernel and a ramdisk (optional); this image contains the operating system and applications which will be used or have been requested by the cloud consumer. The VM image is required to start its

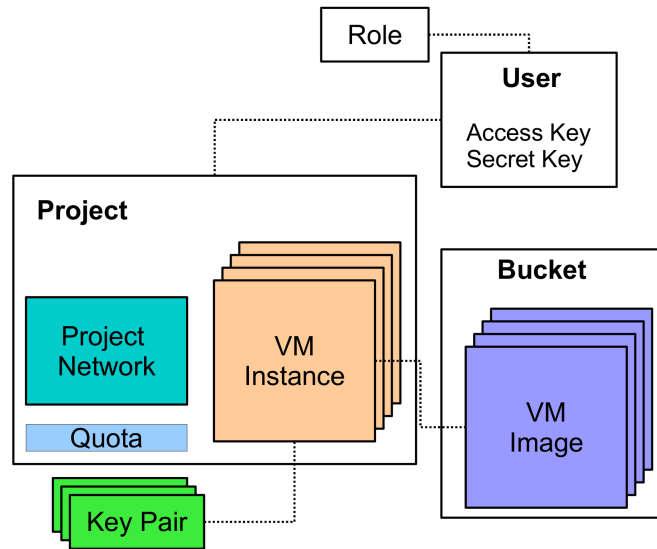


Figure 8.1: Logical relation of entities in an OpenStack environment

corresponding VM instance. *Bucket* is a container for images. Their logical relation is depicted in Figure 8.1.

8.1.1 System Requirements

The hardware requirements of a cloud controller in an experimental deployment of the OpenStack is shown in Table 8.1. We will use same configuration for all host machines. These host machines will run the compute controller, network controller, volume controller, scheduler, and object store services.

MySQL is recommended as the database, because it is easier to do high availability with it. Moreover, Master/Slave replication of MySQL is recommended for high availability setup. The more slaves, the better it would be, also we should consider heartbeat to change a slave to master in case of a master failure.

Heartbeat manages the configuration of different resources to manage the switching between two servers in the event of a failure. The resource configuration defines the individual services that should be brought up (or taken down) in the event of a failure. [16]

Cloud Controller	Minimum	Suggested	Actual
CPU	1GHz	Two 2GHz	Four 2.80GHz
HVM Support	Not essential	Recommended	Available
Memory	1GB	8GB	8GB
Disk Space	40GB	200GB	1TB
Networking	100Mbps	1Gbps	1Gbps ToR, 100Mbps Uplink
Operating System	Ubuntu, CentOS, RHEL	Ubuntu 10.04 LTS	Ubuntu 10.04.1 LTS
Database	SQLAlchemy Compatible	MySQL, Post- greSQL	MySQL

Table 8.1: Hardware Requirement for Cloud Controller

Using MySQL proxy will even improve the performance more, by splitting read/write requests.

There exists a variety of methods for scaling out OpenStack, one way is to add more compute nodes, under *nova-compute* service name. We have a single cloud controller and four compute nodes.

8.1.2 Architecture/Structure

In our laboratory configuration, we used the simple flat structure. This will avoid further complexity which is caused by the hierarchical or peer to peer architecture. We have four physical machines, one of them will be the cloud controller and eight services will run on it. Services are *nova-api*, *nova-network*, *nova-scheduler*, *nova-objectstore*, *nova-compute*, *glance*, *dnsmasq*, and *mysql server*. The other three nodes are compute workers which have only the *nova-compute* service. The abstract diagram of our lab setup is depicted in Figure 8.2.

Each machine has two network interfaces, which are connected to different switches (i.e. SW1 and SW2). The first interface is for communicating with the outside world and is used mainly for the platform management. The second interface is used to inter-connect virtual machines. A Linux bridge is created using the second interface on each machine. Virtual interfaces for instances are added to the bridge on the physical machine. Thus, the bridge in each host works as a layer 2 switch. All these bridges are connected to each other using another switch (i.e. SW1). Figure 8.3 gives an abstract view of the inner structure of a compute node.

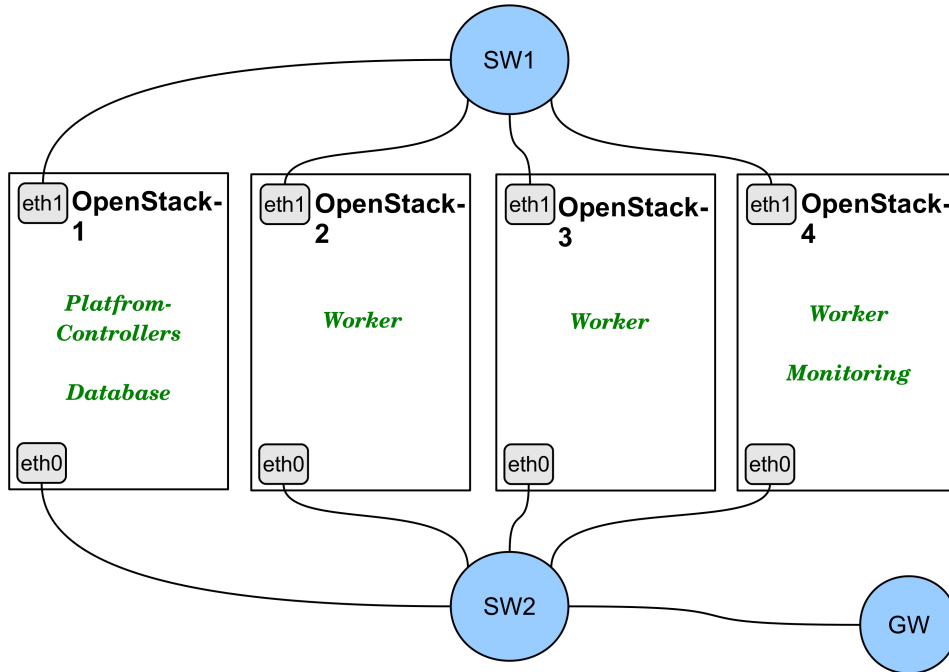


Figure 8.2: Hosts structure in our laboratory configuration

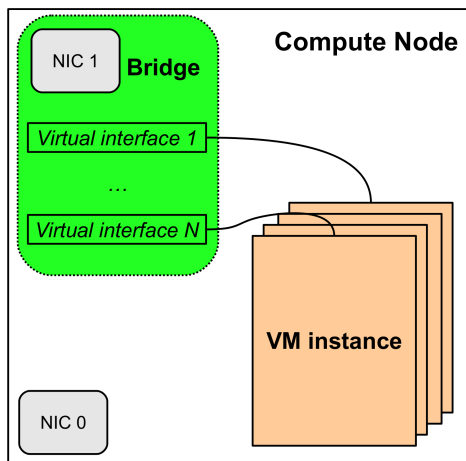


Figure 8.3: A compute node in our laboratory configuration

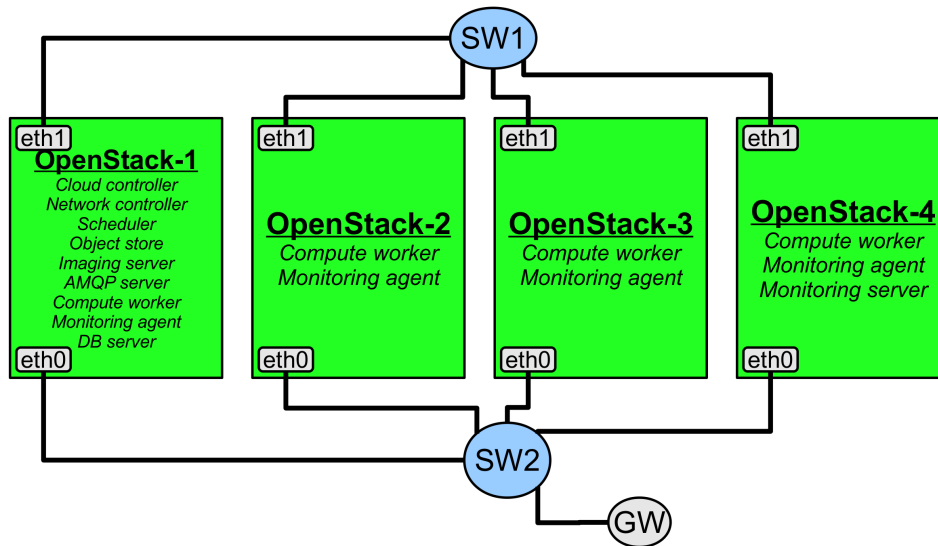


Figure 8.4: Running services on each physical host

8.1.3 Component Distribution

We have four compute nodes that run the nova-compute service. The host with nova-compute service can host virtual machine instances. Thus, all four physical machines in our configuration are capable of hosting VM instances.

The first host (i.e. OpenStack-1) is also working as the cloud controller. The cloud controller is responsible for running the messaging service (i.e. RabbitMQ). Additionally, our cloud controller node handles the networking service, database functionality, task scheduling, and provide storage service.

Moreover, a Simple Network Management Protocol (SNMP) agent is installed on each host for monitoring purposes. The monitoring server is running on OpenStack-4. A detailed figure of the lab deployment is in Figure 8.4, also running services on each host are listed in Tables 8.2, 8.3, 8.4, 8.5.

8.1.4 Installation

Cloud Controller Node

Initially we need to add the Nova and Glance Personal Package Archives (PPA) repository, and update the local list of packages to include nova files.

Listing 8.1: Adding Nova package repository

8.1. OpenStack Compute Deployment

Specification	
IP	129.241.252.119
OS	Linux openstack-1 2.6.32-24-server #39-Ubuntu SMP Wed Jul 28 06:21:40 UTC 2010 x86_64 GNU/Linux
HW	4 * (Intel(R) Core(TM) i5-2300 CPU), 8GB RAM, 1 TB HDD, 2 * 1 GB NIC
Services	
Cloud Controller	RabbitMQ
Networking	nova-network, Dnsmasq
Scheduling	nova-scheduler
Storage	nova-objectstore, glance
Computing	nova-compute
Database	MySQL, SQLite (glance)
Monitoring	SNMP Agent

Table 8.2: OpenStack-1 specification and services

Specification	
IP	129.241.252.118
OS	Linux openstack-1 2.6.32-24-server #39-Ubuntu SMP Wed Jul 28 06:21:40 UTC 2010 x86_64 GNU/Linux
HW	4 * (Intel(R) Core(TM) i5-2300 CPU), 8GB RAM, 1 TB HDD, 2 * 1 GB NIC
Services	
Computing	nova-compute
Monitoring	SNMP Agent

Table 8.3: OpenStack-2 specification and services

Specification	
IP	129.241.252.117
OS	Linux openstack-1 2.6.32-24-server #39-Ubuntu SMP Wed Jul 28 06:21:40 UTC 2010 x86_64 GNU/Linux
HW	4 * (Intel(R) Core(TM) i5-2300 CPU), 8GB RAM, 1 TB HDD, 2 * 1 GB NIC
Services	
Computing	nova-compute
Monitoring	SNMP Agent

Table 8.4: OpenStack-3 specification and services

8.1. OpenStack Compute Deployment

Specification	
IP	129.241.252.116
OS	Linux openstack-1 2.6.32-24-server #39-Ubuntu SMP Wed Jul 28 06:21:40 UTC 2010 x86_64 GNU/Linux
HW	4 * (Intel(R) Core(TM) i5-2300 CPU), 8GB RAM, 1 TB HDD, 2 * 1 GB NIC
Services	
Computing	nova-compute
Monitoring	Cacti server, SNMP Agent
Web Server	Apache

Table 8.5: OpenStack-4 specification and services

```
root@openstack-1:~# add-apt-repository \  
    ppa:nova-core/trunk  
root@openstack-1:~# add-apt-repository \  
    ppa:glance-core/trunk  
root@openstack-1:~# apt-get update
```

Then we continue to install nova packages and their dependencies.

Listing 8.2: Installing Nova packages

```
root@openstack-1:~# apt-get install python-greenlet\  
    python-mysqldb python-nova nova-common\  
    nova-doc nova-api nova-network \  
    nova-objectstore nova-scheduler \  
    nova-compute rabbitmq-server euca2ools unzip
```

Now we should add the Glance project which provides the Imaging Service.

Listing 8.3: Installing the Glance package

```
root@openstack-1:~# apt-get install glance
```

Finally we install the SNMP Agent and mySQL database server.

Listing 8.4: Installing the SNMP agent

```
root@openstack-1:~# apt-get install snmpd \  
    mysql-server
```

Compute Node

In each of worker node (i.e. OpenStack-2, OpenStack-3, OpenStack-4), the nova-compute and the SNMP Agent should be installed.

Listing 8.5: Installing Nova and SNMP Agent

```
root@openstack-2:~# add-apt-repository \
    ppa:nova-core/trunk
root@openstack-2:~# apt-get update
root@openstack-2:~# apt-get install nova-compute
root@openstack-2:~# apt-get install snmpd
```

We also need to install the Cacti monitoring server on OpenStack-4 using apt-get command.

8.1.5 Configuration

Cloud Controller

We start by configuring our cloud controller. The database, S3 host, rabbitmq server, and EC2 host are all installed and running on OpenStack-1. Moreover, the network range for all instances in the cloud is 192.168.0.0/12 and we limit number of IP addresses in the cloud to 5000. The networking mode which has been used in the deployment is FlatDHCP which will be explained later. Thus, the `/etc/nova/nova.conf` configuration file will be as follows:

Listing 8.6: nova.conf

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--lock_path=/var/lock/nova
--verbose
--sql_connection=mysql://root:nova@129.241.252.119/nova
--s3_host=129.241.252.119
--rabbit_host=129.241.252.119
--ec2_host=129.241.252.119
--ec2_url=http://129.241.252.119:8773/services/Cloud
--network_manager=nova.network.manager.FlatDHCPManager
--flat_network_dhcp_start=192.168.0.2
--flat_interface=eth1
```

```
--flat_injected=False
--public_interface=eth0
--fixed_range=192.168.0.0/12
--network_size=5000
--glance_host=129.241.252.119
--image_service=nova.image.glance.GlanceImageService
```

In the following you may find the description of some useful configuration parameters in *nova.conf*; also the exact value of each parameter in our installation is in Table 8.6:

- **sql_connection:** URI of OpenStack Compute SQL database
- **s3_host:** Location of OpenStack ObjectStore service that has VM images and buckets
- **rabbit_host:** Location of OpenStack Compute database
- **cc_host:** Location of OpenStack nova-api service
- **ec2_url:** nova-api URL
- **network_manager:** Specifies network configuration for communication of Cloud Controller with its Compute nodes and VM instances
- **flat_network_dhcp_start:** The starting IP address for the our flat network
- **flat_interface:** The host interface which is used for bridging
- **public_interface:** The host interface that is accessible publicly and mainly used for management.
- **fixed_range:** IP block that will be used for VM instances in the cloud. This block is shared between all customers' projects in the cloud. While defining a new project, a subnet of this block can be assigned specifically to a project in the cloud.
- **network_size:** Number of all IP addresses that may be assigned to instances across all projects
- **glance_host:** Location of glance imaging service
- **imaging_service:** The type of imaging service which is used, can be either local or glance image service.

8.1. OpenStack Compute Deployment

Parameter	Type	Value
sql_connection	URI	mysql://root:nova@129.241.252.119/nova
s3_host	IP	129.241.252.119
rabbit_host	IP	129.241.252.119
cc_host	IP	129.241.252.119
ec2_url	HTTP URL	http://129.241.252.119:8773/services/Cloud
network_manager	String	nova.network.manager.FlatDHCPManager
flat_network_dhcp_start	IP	192.168.0.2
flat_interface	String	eth1
public_interface	String	eth0
fixed_range	Net prefix	192.168.0.0/12
network_size	Integer	5000
glance_host	IP	129.241.252.119
imaging_service	String	nova.image.glance.GlanceImageService

Table 8.6: Cloud controller parameters and our deployment details [56]

As the *nova.conf* file contains the database administrator's password, we should limit the access to it.

Listing 8.7: Setting *nova.conf* permissions

```
root@openstack-1:~# chown -R root:nova /etc/nova
root@openstack-1:~# chmod 644 /etc/nova/nova.conf
```

Now we should configure the previously installed MySQL server. The MySQL server should be configured in such a way that will be accessible from required machines. Also, we should create nova database by ourselves and grant all privileges to the administrator user.

Listing 8.8: Configuring MySQL for nova

```
root@openstack-1:~# sed -i 's/127.0.0.1/0.0.0.0/g'\
/etc/mysql/my.cnf
root@openstack-1:~# mysql -uroot -pnova
mysql> CREATE DATABASE nova;
mysql> GRANT ALL PRIVILEGES ON *.* TO\
'root'@'0'%' WITH GRANT OPTION;
```

After finishing with the nova components and the database, we should configure the SNMP agent as well. So it will be accessible by the monitoring

server. We add/modify few lines to `/etc/snmp/snmpd.conf` configuration file to grant the access to the monitoring server and unify the SNMPv2c community.

Listing 8.9: Modified lines of `snmpd.conf`

```
# rwuser: a SNMPv3 read-write user
rwuser cacti

# rwcommunity: a SNMPv1/SNMPv2c read-write access
# community name
rwcommunity opencommunity openstack-4

com2sec readwrite openstack-4 opencommunity
```

Our Network Configuration

We have used FlatDHCP mode for network configuration. The IP block of 192.168.0.0/12 is used for all the projects and 5000 IP addresses are available for use in this range. We authorized ICMP and SSH packets to all VM instances by default, using the following commands:

Listing 8.10: Authorizing ICMP and SSH

```
# euca-authorize -P icmp -t -1:-1 default
# euca-authorize -P tcp -p 22 default
```

Finally we should add the following rule to ensure destination natting of the traffic heading towards 169.254.169.254.

Listing 8.11: Adding iptables rule to redirect the traffic

```
# iptables -t nat -A PREROUTING \
    -d 169.254.169.254/32 \
    -p tcp -m tcp --dport 80 -j DNAT \
    --to-destination 129.241.252.119:8773
```

Compute Node

In order to scale out our cloud platform, we installed several compute nodes as explained previously. We should ensure that `nova.conf` has the proper parameters/values on each compute node. Parameters should point to the right IP address of the cloud controller, object store, compute database, nova-api service and imaging service.

8.1.6 Management

This part contains basic management functionalities of the OpenStack Compute including: loading and deleting a VM image, starting and terminating an instance.

In order to execute all the following commands, appropriate credentials should be exported beforehand. Nova provides a set of credentials for a user in an specific project.

So, we should start by add a new user and creating a new project and assigning the user to that project. Finally we need to create the required network and extract user credentials.

Listing 8.12: Generating credentials

```
# nova-manage user admin aryan
# nova-manage project create myproject aryan
# nova-manage network create 192.168.0.0/24 1 255
# nova-manage project zipfile myproject aryan
```

The output of the last command is a zip file named *nova.zip*. It contains credentials and a *novarc* file. Sourcing the *novarc* will export appropriate variables to the current shell environment.

Listing 8.13: Sourcing *novarc*

```
# source /root/creds/novarc
```

VM Images

OpenStack provides some basic VM images for testing the installation. But we can create our own images as well. This procedure becomes easier by means of the following code: <https://code.launchpad.net/smoser/+junk/ttylinux-uec>.

Starting an Instance

We loaded our image using Ubuntu Enterprise Cloud utilities.

Listing 8.14: Uploading a VM image

```
# uec-publish-tarball ${img}.tar.gz mybucket x86_64
```

When the image is loaded correctly, we will get three references, Eucalyptus Machine Image (emi), Eucalyptus Ramdisk Image (eri), Eucalyptus Kernel Image (eki) namely. These

references will be useful for further management/manipulation of VM instances.

When image extraction is completed and the image status changed from "untarring" to "available", we can run the instance using our newly loaded image. `emi` is needed for this step. A key pair should be created first, and will be injected to the instance, so it will be possible to connect to the launched instance using generated key.

Listing 8.15: Creating a key pair and running an instance

```
# euca-add-keypair mykey > mykey.priv
# euca-run-instances <EMI_VALUE> -k mykey -t m1.tiny
```

`mykey` is the user's key for a specific project and `m1.tiny` describes the instance type, which is the smallest one.

We check the status of started instance using:

Listing 8.16: Checking an instance status

```
# euca-describe-instances

RESERVATION      r-a8a9ekp8      myproj default
INSTANCE         i-00000001      ami-00000003
192.168.0.2      192.168.0.2    running mykey
(myproj, openstack-1)  0m1.tiny
2011-04-13T09:37:42Z  nova
```

and when it becomes "running", it would be possible to SSH to its IP address that is extracted from the previous command.

Terminating an Instance

In order to terminate an instance we use the instance ID from the instance description.

Listing 8.17: Terminating an instance

```
# euca-terminate-instances <INSTANCE_ID>
```

Deleting an Image

A bucket can be deleted using `euca` utilities. Bucket is a container for an image.

Listing 8.18: Deleting a bundle

```
# euca-delete-bundle -b mybucket
```

8.1.7 Operation

Instance launching [46]

Starting a new instance requires cooperation of several components, including: API Server, Cloud Controller, Scheduler, Compute Controller and Network Controller. Their functionalities have been explained previously in Section 3.2. As it is depicted in Figure 8.5, the abstract communication of major components, for starting an instance, can be represented in 8 steps. It is assumed that only one Cloud Controller (i.e. Messaging server) exists. Details of each step is as follows: [46]

1. The API server forwards the user request for running an instance to the cloud controller.
 - Check number of running instances for this particular type and avoid hitting the threshold.
 - Check existence of corresponding security group, otherwise create one.
 - Generate networking parameters (i.e. Media Access Control (MAC) Address and Hostname).
 - Communicating with the scheduler to find the proper compute worker.
2. The scheduler chooses a compute worker and sends it the request to start a new instance.

The message delivery type which is used for sending the request to the scheduler is called *RPC Casting*. Contrary to a *call*, in a *cast*, no response is expected. In this delivery mode, the API server publishes the request and a scheduler worker consumes and retrieves it from the queue.

3. A compute worker receives the request. Compute checks if the instance is already started, if not it will request for a fixed IP address and setup the VLAN and bridge configurations. [46].

The compute controller uses a *RPC call* to ask for a fixed IP address. In this technique a specific host is targeted and a response is expected. The target for this request is the network controller component.

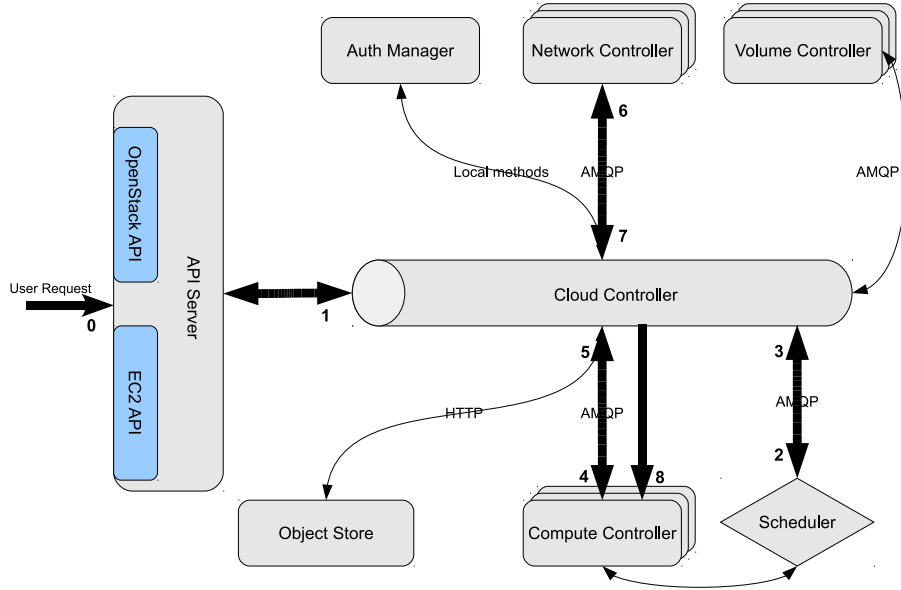


Figure 8.5: Message flow for running an instance [46]

4. The compute spawns the requested VM instance by means of a virtualization driver. In this step proper firewall rules will also be applied.

8.2 Performance Monitoring of the Infrastructure

As we discussed in Chapter 5, detecting an incident and explaining its specifications, requires the knowledge about the normal operation of a system. Using system monitoring technique we understand the expected behavior of a cloud environment and its components.

We have used several tools for monitoring the cloud components, including: RRDTool, Simple Network Management Protocol Daemon (SNMPD), sFlow, collectd, Cacti.

- **RRDTool**

The RRDTool is a package for logging and graphing system characteristics based on time series data [55].

- **SNMPD**

SNMPD is a daemon that responds to SNMP requests. It is a SNMP agent that listen for SNMP request from management software [54].

- **sFlow**

The sFlow [70] is a standard technology which is used for monitoring networks and hosts. The scalability and metrics integration are two important characteristics of sFlow standard [69].

- **Cacti**

Cacti uses the RRDTool package for data storage and data graphing. Cacti has a poller functionality to gather data from different sources. It uses multiple data acquisition methods [4].

8.2.1 Installation and Configuration

SNMPD

Listing 8.19: Installing and configuring SNMP daemon

```
# apt-get install snmpd
# snmpconf -g basic_setup
```

First command will install the SNMPD and all required packages. *snmpconf* will do the basic configuration for the SNMP agent.

Cacti

Listing 8.20: Installing and configuring SNMP daemon

```
# apt-get install cacti cacti-cactid cacti-spine
```

This will install Cacti and all its dependencies. To complete the installation, we should check the following address and proceed the procedure.

Host sFlow

8.2.2 Data Sources and Graphs

We measure and draw corresponding graphs for the following resources/entities:

- CPU utilization (Figure 8.6)

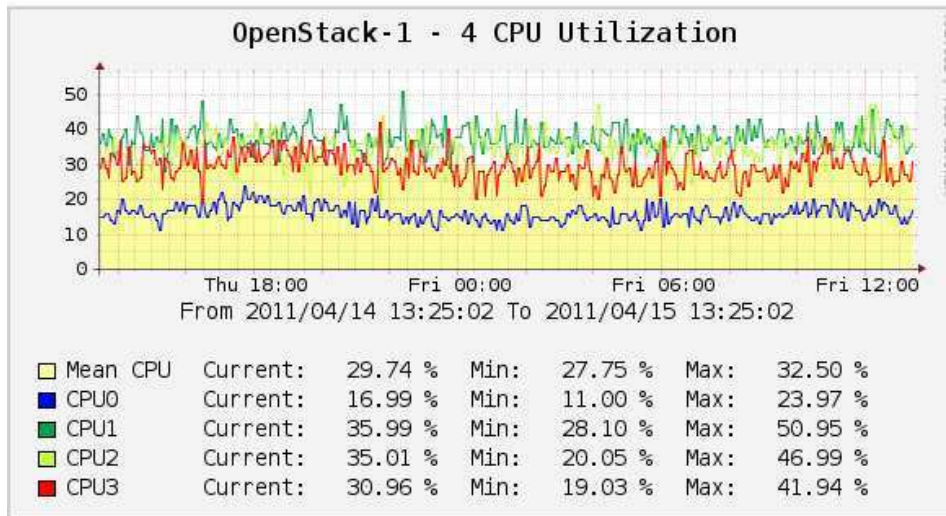


Figure 8.6: CPU Utilization

- Network traffic and protocol statistics (Figures 8.7, 8.9, 8.11)
- Disk input/output (Figure 8.10)
- Load (Figure 8.12)
- Memory (Figure 8.8)
- Number of processes
- Number of logged in users

Each of these statistics are gathered and represented for five different period (i.e. one minute, five minute, 30 minute, two hour, one day)

8.2. Performance Monitoring of the Infrastructure

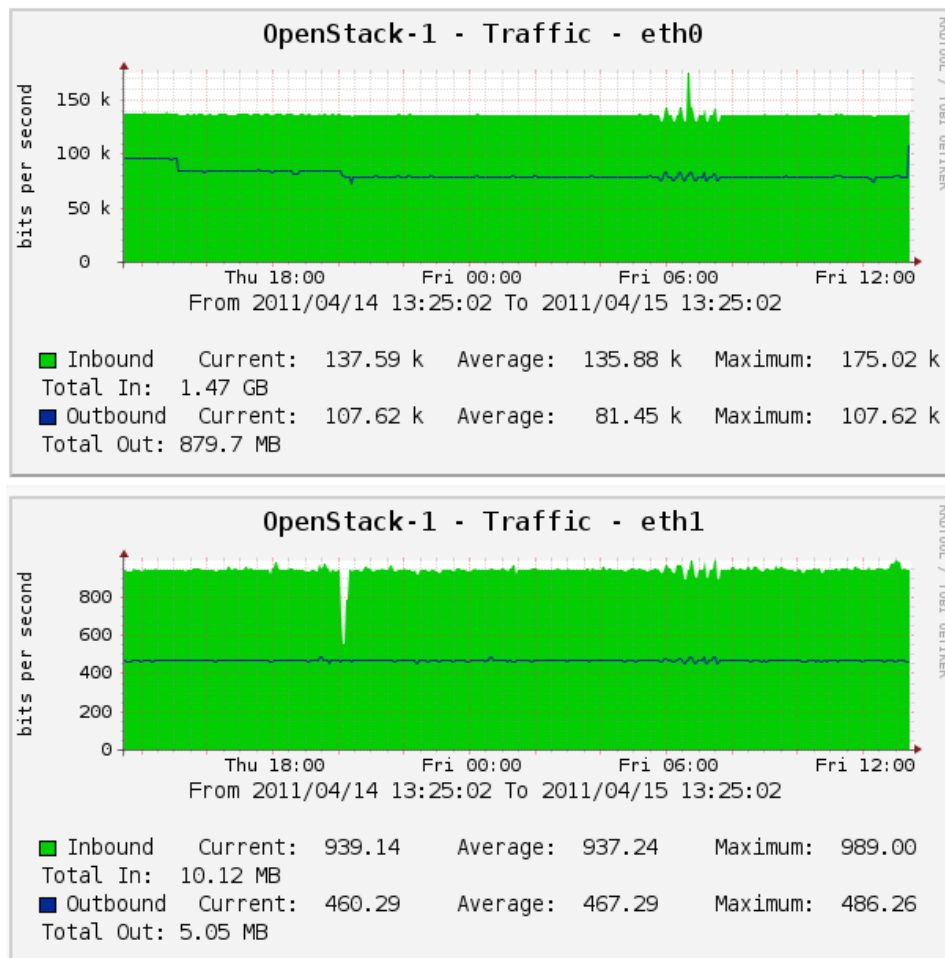


Figure 8.7: NIC Traffic

8.2. Performance Monitoring of the Infrastructure

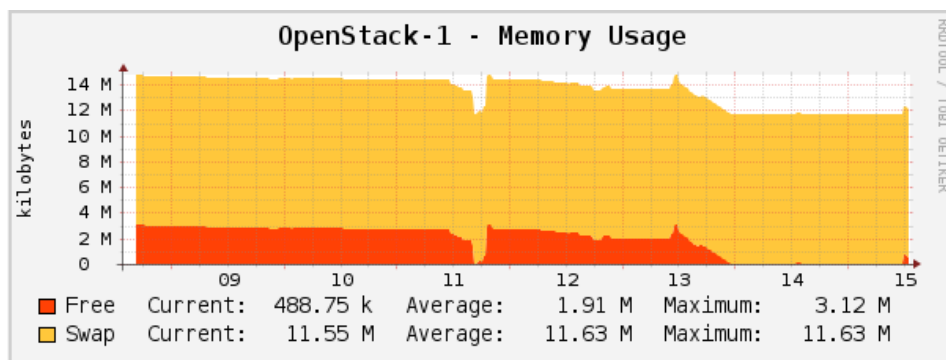


Figure 8.8: Memory Usage

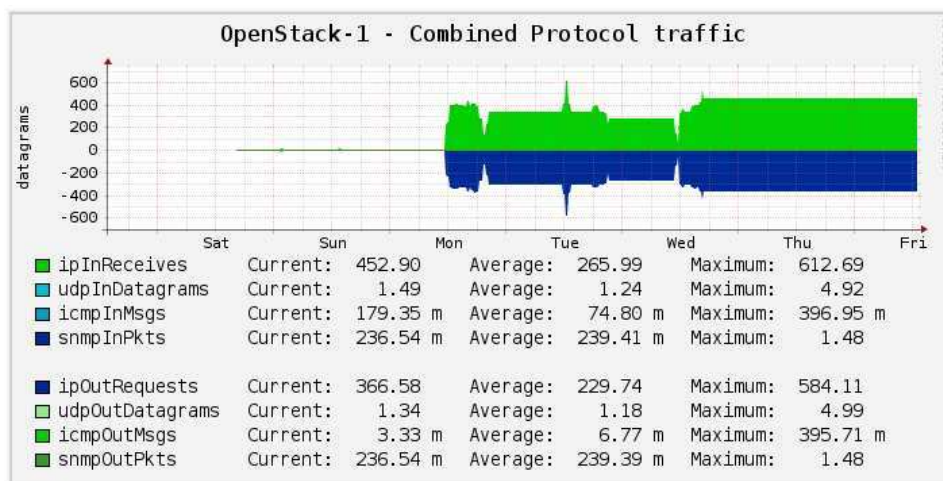


Figure 8.9: Combined Traffic statistics

8.2. Performance Monitoring of the Infrastructure

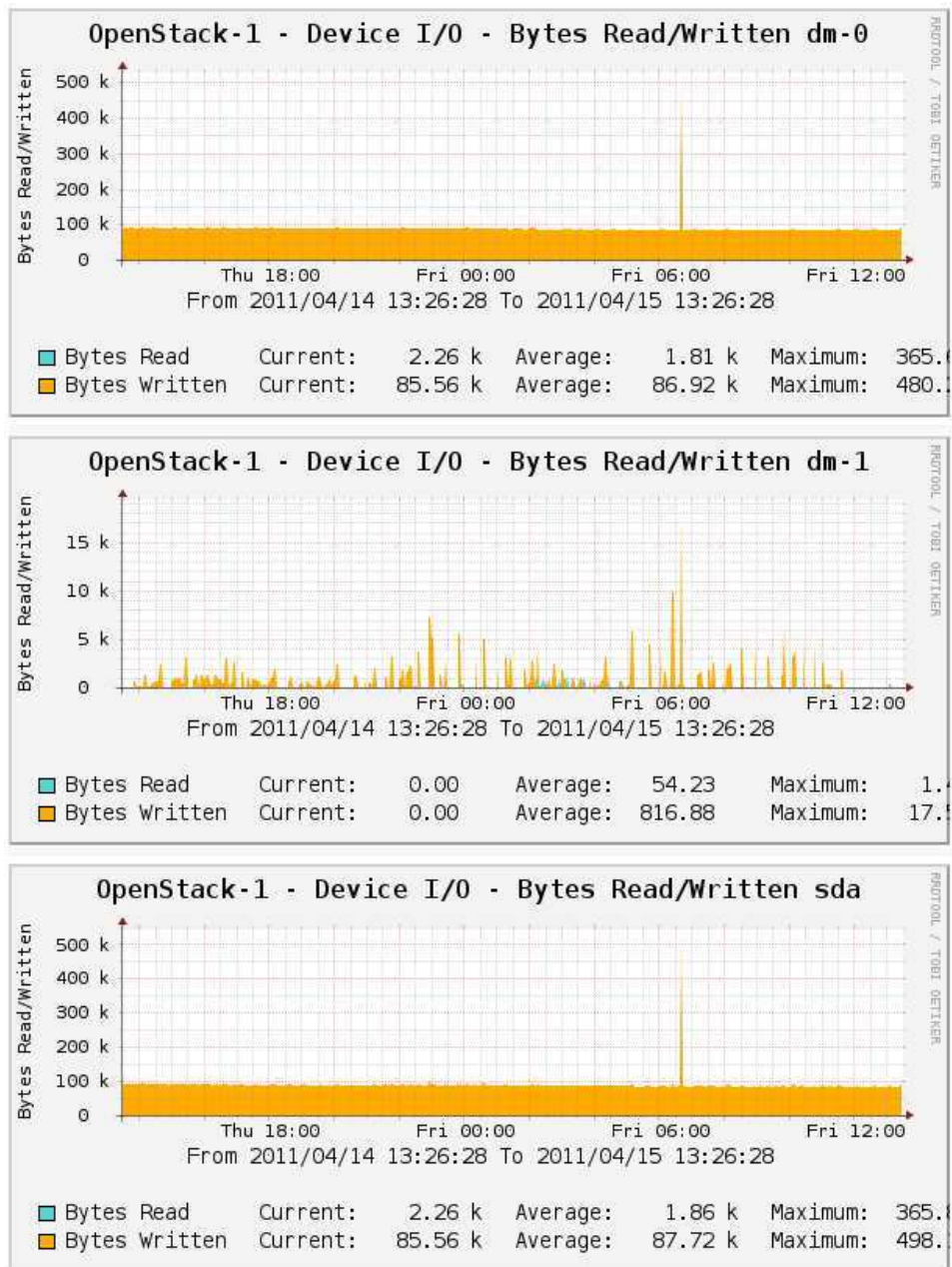


Figure 8.10: Disk Input/Output

8.2. Performance Monitoring of the Infrastructure

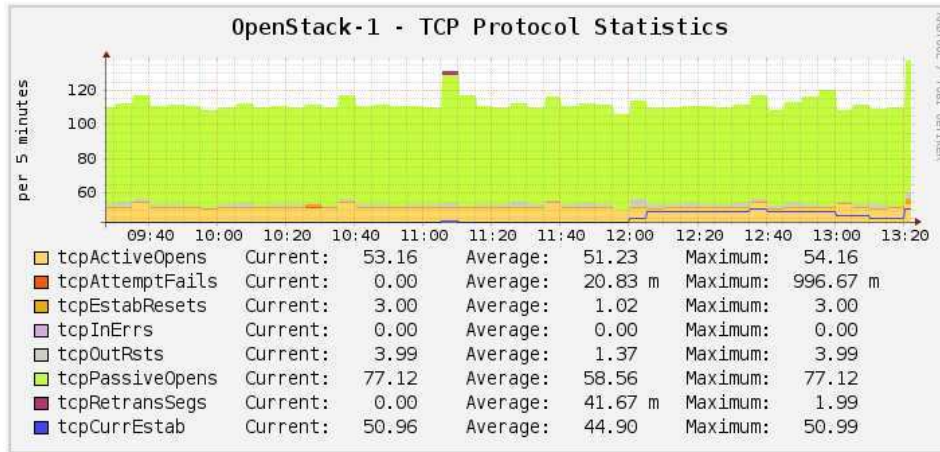


Figure 8.11: TCP Protocol statistics

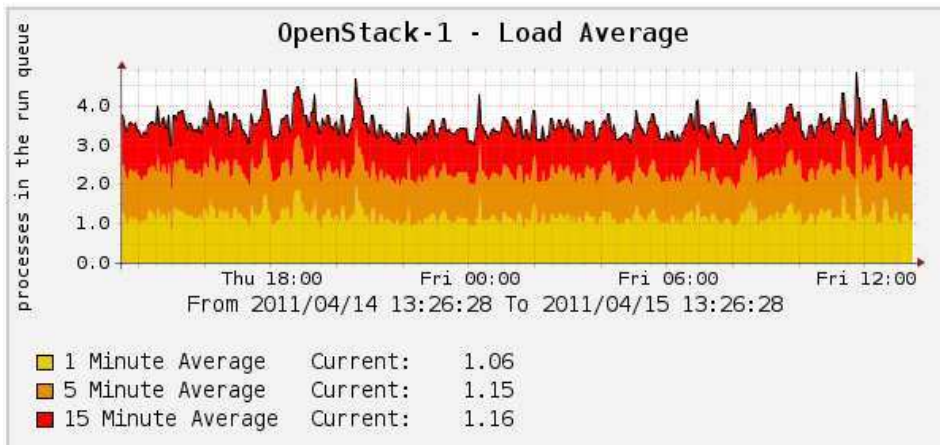


Figure 8.12: Load Average

8.3 Messaging Server Management and Monitoring

The OpenStack uses RabbitMQ as the messaging server in the cloud controller machine. One of the main approaches which we have proposed for containment and eradication was blocking the compromised node **at the messaging server**. This section will explain how we changed the messaging server and its dependencies so it will be capable of having the monitoring and management plug-in.

8.3.1 Installation

The RabbitMQ server which has been installed by default during the OpenStack installation will not work with RabbitMQ management plug-in. Thus, you have to make sure that the latest version of RabbitMQ and Erlang is installed. We start by removing the already installed RabbitMQ server and Erlang package²⁴.

Listing 8.21: Removing the existing RabbitMQ server and Erlang package

```
# apt-get remove --purge rabbitmq-server
# apt-get remove --purge erlang
```

Installing Erlang from source requires two libraries which will be installed first.

Listing 8.22: Installing Erlang dependencies

```
# apt-get install libncurses5-dev xmlto
```

Listing 8.23: Installing Erlang from its source

```
# wget http://www.erlang.org/download/\
    otp_src_R14B02.tar.gz
# tar xvfz otp_src_R14B02.tar.gz
# cd otp_src_R14B02
# ./configure && make && make install
```

Now we continue by installing the RabbitMQ server from its source

²⁴Nova services must be stopped before removing the messaging server, otherwise you will see lots of error messages in log files.

Listing 8.24: Installing RabbitMQ

```
# wget http://www.rabbitmq.com/releases/\
      rabbitmq-server/v2.4.1/rabbitmq-\
      server-2.4.1.tar.gz
# tar xvfz rabbitmq-server-2.4.1.tar.gz
# cd rabbitmq-server-2.4.1
# make && sudo make install \
      SBIN_DIR=/usr/local/sbin \
      MAN_DIR=/usr/local/man/ \
      TARGET_DIR=/usr/local/lib/rabbitmq
```

Finally we should install the RabbitMQ management plug-in with its dependencies.

Listing 8.25: Installing RabbitMQ Management and Monitoring plug-in

```
# cd /usr/local/lib/rabbitmq/plugins
# wget http://www.rabbitmq.com/releases/plugins/\
v2.4.1/rabbitmq-management-2.4.1.ez
# wget http://www.rabbitmq.com/releases/plugins/\
v2.4.1/rabbitmq-management-agent-2.4.1
# wget http://www.rabbitmq.com/releases/plugins/\
v2.4.1/rabbitmq-mochiweb-2.4.1.ez
# wget http://www.rabbitmq.com/releases/plugins/\
v2.4.1/amqp_client-2.4.1.ez
# wget http://www.rabbitmq.com/releases/plugins/\
v2.4.1/webmachine-2.4.1.ez
# wget http://www.rabbitmq.com/releases/plugins/\
v2.4.1/mochiweb-2.4.1.ez
# killall -9 rabbitmq-server
# rabbitmq-server -detach
```

8.3.2 Operating

After installing the plug-in, we can start managing and monitoring the messaging server by browsing <http://openstack-1:55672/mgmt/>

8.3. Messaging Server Management and Monitoring

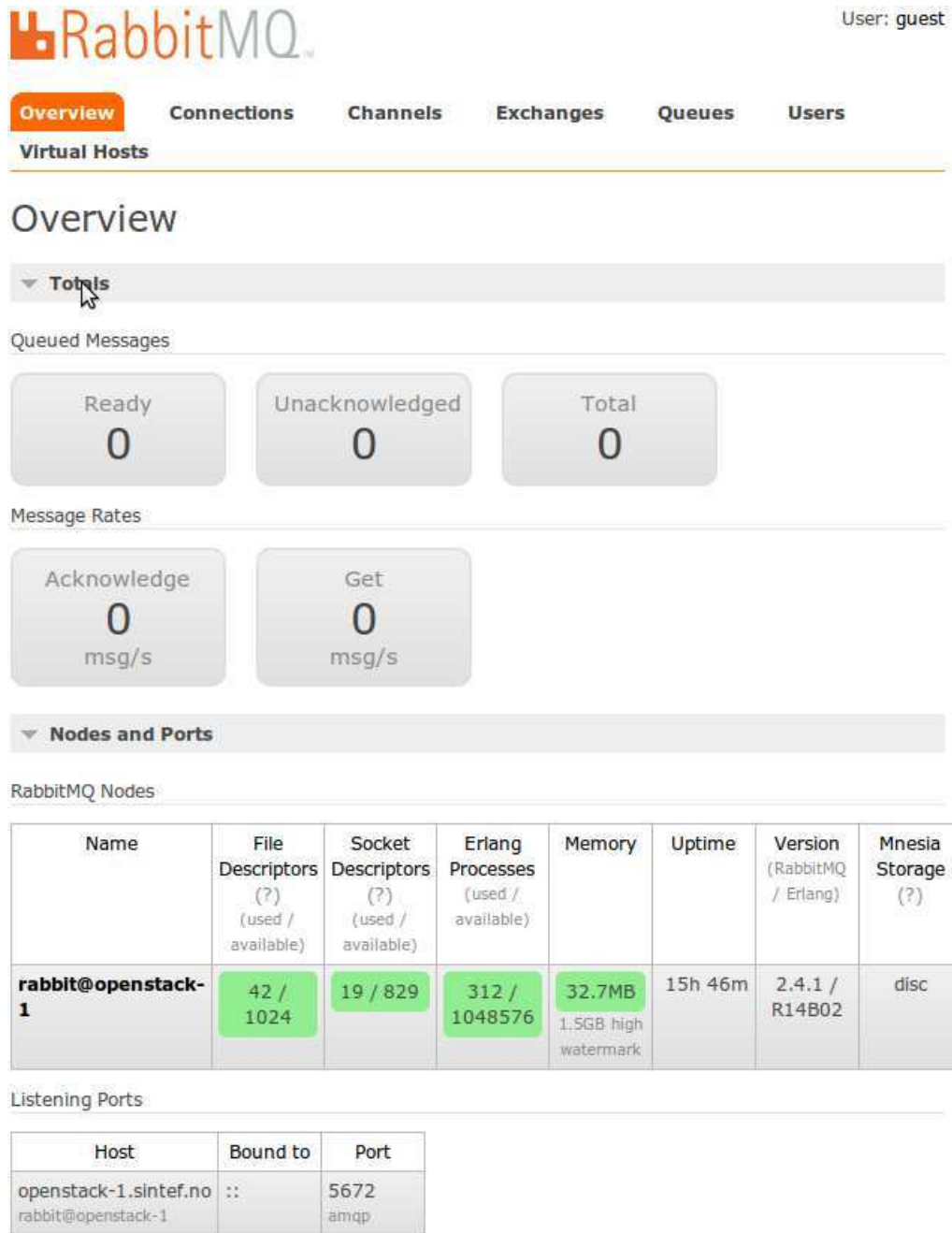


Figure 8.13: RabbitMQ Management Overview

8.3. Messaging Server Management and Monitoring

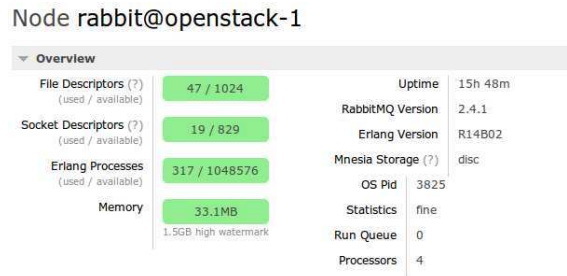


Figure 8.14: RabbitMQ Node rabbit@openstack-1 Overview

Connections

Network			Overview		
Peer address	From client	To client	Channels	User name	State
129.241.252.116:41057	604B/s <small>(32.4MB total)</small>	125B/s <small>(6.7MB total)</small>	1	guest	running
129.241.252.116:41058	220B/s <small>(11.8MB total)</small>	125B/s <small>(6.7MB total)</small>	1	guest	running
129.241.252.116:41059	330B/s <small>(18.0MB total)</small>	122B/s <small>(6.7MB total)</small>	1	guest	running
129.241.252.117:33649	227B/s <small>(12.1MB total)</small>	128B/s <small>(6.9MB total)</small>	1	guest	running
129.241.252.117:33650	347B/s <small>(18.5MB total)</small>	129B/s <small>(6.9MB total)</small>	1	guest	running
129.241.252.117:33651	623B/s <small>(33.3MB total)</small>	128B/s <small>(6.9MB total)</small>	1	guest	running
129.241.252.118:49885	585B/s <small>(32.0MB total)</small>	121B/s <small>(6.6MB total)</small>	1	guest	running
129.241.252.118:49886	325B/s <small>(17.8MB total)</small>	121B/s <small>(6.6MB total)</small>	1	guest	running
129.241.252.118:49887	214B/s <small>(11.7MB total)</small>	121B/s <small>(6.6MB total)</small>	1	guest	running
129.241.252.119:48262	229B/s <small>(12.2MB total)</small>	129B/s <small>(6.9MB total)</small>	1	guest	running
129.241.252.119:48263	347B/s <small>(18.5MB total)</small>	129B/s <small>(6.9MB total)</small>	1	guest	running
129.241.252.119:48264	626B/s <small>(33.3MB total)</small>	129B/s <small>(6.9MB total)</small>	1	guest	running
129.241.252.119:49251	249B/s <small>(13.5MB total)</small>	129B/s <small>(7.0MB total)</small>	1	guest	running
129.241.252.119:49252	367B/s <small>(20.0MB total)</small>	129B/s <small>(7.0MB total)</small>	1	guest	running
129.241.252.119:49253	646B/s <small>(35.1MB total)</small>	129B/s <small>(7.0MB total)</small>	1	guest	running
129.241.252.119:49254	229B/s <small>(12.4MB total)</small>	129B/s <small>(7.0MB total)</small>	1	guest	running
129.241.252.119:49255	348B/s <small>(19.1MB total)</small>	129B/s <small>(10.8MB total)</small>	1	guest	running
129.241.252.119:49256	626B/s <small>(34.0MB total)</small>	129B/s <small>(7.0MB total)</small>	1	guest	running

Figure 8.15: RabbitMQ Connections

8.3. Messaging Server Management and Monitoring



Figure 8.16: RabbitMQ a Connection Details

Channels

Channel	Details					
	User name	Mode (?)	Prefetch	Unacked	Unconfirmed	Status
129.241.252.116:41057:1	guest		0	0	0	Active
129.241.252.116:41058:1	guest		0	0	0	Active
129.241.252.116:41059:1	guest		0	0	0	Active
129.241.252.117:33649:1	guest		0	0	0	Active
129.241.252.117:33650:1	guest		0	0	0	Active
129.241.252.117:33651:1	guest		0	0	0	Active
129.241.252.118:49885:1	guest		0	0	0	Active
129.241.252.118:49886:1	guest		0	0	0	Active
129.241.252.118:49887:1	guest		0	0	0	Active
129.241.252.119:48262:1	guest		0	0	0	Active
129.241.252.119:48263:1	guest		0	0	0	Active
129.241.252.119:48264:1	guest		0	0	0	Active
129.241.252.119:49251:1	guest		0	0	0	Active
129.241.252.119:49252:1	guest		0	0	0	Active
129.241.252.119:49253:1	guest		0	0	0	Active
129.241.252.119:49254:1	guest		0	0	0	Active
129.241.252.119:49255:1	guest		0	0	0	Active
129.241.252.119:49256:1	guest		0	0	0	Active

Figure 8.17: RabbitMQ Channels

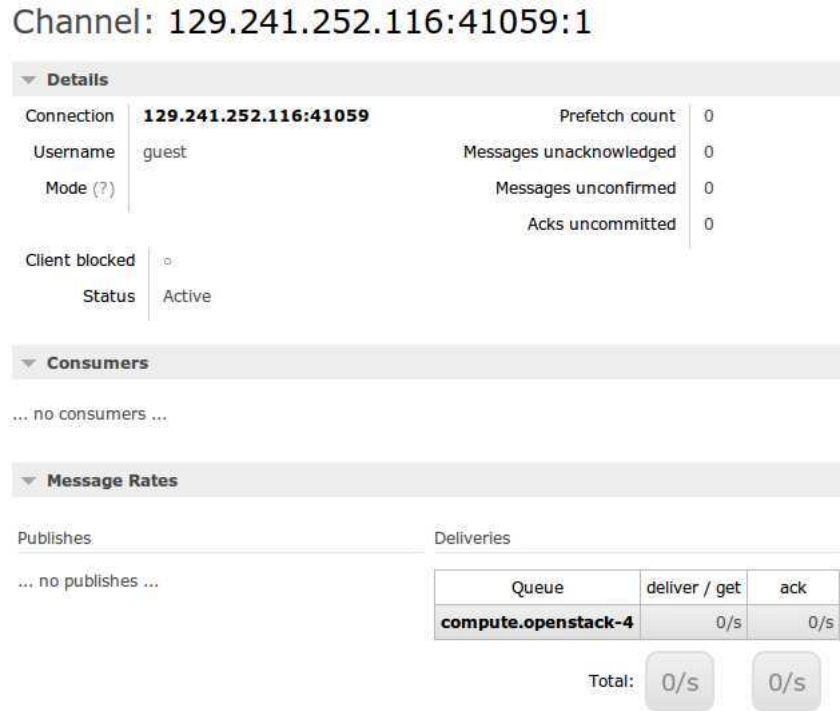


Figure 8.18: RabbitMQ a Channel Details

Exchanges

All exchanges			
Name	Type	Parameters	Message rate
(AMQP default)	direct	D	
amq.direct	direct	D	
amq.fanout	fanout	D	
amq.headers	headers	D	
amq.match	headers	D	
amq.rabbitmq.log	topic	D	
amq.topic	topic	D	
compute_fanout	fanout		
network_fanout	fanout		
nova	topic		
scheduler_fanout	fanout		

Figure 8.19: RabbitMQ Exchange

Exchange: nova

Overview
 Message Rates
 Bindings

Outgoing from nova

To	Routing Key	Arguments	
compute Queue	compute		Unbind
compute.openstack-1 Queue	compute.openstack-1		Unbind
compute.openstack-2 Queue	compute.openstack-2		Unbind
compute.openstack-3 Queue	compute.openstack-3		Unbind
compute.openstack-4 Queue	compute.openstack-4		Unbind
network Queue	network		Unbind
network.openstack-1 Queue	network.openstack-1		Unbind
scheduler Queue	scheduler		Unbind
scheduler.openstack-1 Queue	scheduler.openstack-1		Unbind

Add Binding

nova →

queue :
 Routing Key:
 Arguments: =

Bind

Figure 8.20: RabbitMQ an Exchange Details

Queues

▼ All queues

Overview			
Name	Exclusive	Parameters	Status
compute			Active
compute.openstack-1			Active
compute.openstack-2			Active
compute.openstack-3			Active
compute.openstack-4			Active
compute_fanout_10809d7fbcf845e8b441c6e5c9ab2736			Active
compute_fanout_7545bd713a884336a681eb9f8c5a2aeb			Idle
compute_fanout_776b58a98056423a9b304dbbe640ed62			Active
compute_fanout_90fdbcccef9240d88e8e7e4838f5e22c			Active
compute_fanout_b67a2436a53343d19e9d25748e6d14a8			Active
compute_fanout_cfd7d0326d6c4c63bebee5944bde3a10			Idle
network			Active
network.openstack-1			Active
network_fanout_aa6447787968424aa0e1022686f093d0			Active
scheduler			Active
scheduler.openstack-1			Active
scheduler_fanout_ac0b82dc83e44b6696e26a186b473b0b			Active

► Add a new queue

Figure 8.21: RabbitMQ Queues

Queue compute

▼ Overview

Messages

Ready
0

Unacknowledged
0

Total
0

Details

Parameters		Consumers	
Exclusive owner	None	Memory	53.8kB
Status	Active		

► Message Rates

► Consumers

▼ Bindings

Incoming to compute

From	Routing Key	Arguments	
(AMQP default) Exchange	compute		
nova Exchange	compute		Unbind

→ compute

Add Binding

Exchange:

Routing Key:

Arguments: =

compute

Bind

Figure 8.22: RabbitMQ a Queue Details

Chapter 9

Conclusion

Cloud computing is a new computing model. Its definitions and realizations have new characteristics compared to other computing models. New characteristics hinder the application process of existing mechanisms. In some cases, existing approaches are not applicable and in other cases adaptation is required.

Initially, we studied different aspects of a real cloud environment. We have been working on a deployed environment instead of focusing on an imaginary computing model. Experimenting on a deployed environment is helpful in reducing the gap between academic research and industrial deployment/requirements. We should understand that many questions that are discussed in an academic environment are already solved in industry or are not the right questions at all. A good blog post on this issue can be found in [83].

Although our lab setup was not big enough to be realistic, it was useful for understanding the ecosystem of the cloud model, and observing possible weaknesses in it. Obviously, deploying a larger infrastructure reveals more information about the exact behavior of the environment and the result will be more accurate. However, that may not be feasible as a university project unless big players in the cloud are willing to contribute. Some of those efforts are as follows: OpenCirrus [8] (supported by HP, Intel, and Yahoo!), Google Exacycle [24] program, and Amazon grants for educators, researchers and students [68].

In our thesis we have decided to use the OpenStack cloud software. There were multiple reasons behind this decision, such as:

- Working on an open source project helps its community, and pushes the open source paradigm forward.
- Analysis of the platform and experimenting different approaches are easier and more efficient when we can access the source code.
- Big companies are involved in the OpenStack project, and many of them are using the platform in their own infrastructure. Thus, Open-

Stack can become a leading open source cloud platform in the near future.

When we started our thesis, it was only 4 months after the first release of OpenStack; documentations were not good enough even if they were available. We studied its components and identified their functionalities and other specifications. Moreover, working with a platform which is under heavy development, has its own challenges.

In order to secure the environment against a compromised component, we have to handle the corresponding incident. The NIST incident handling guideline has been studied and applied to our experimental cloud environment. During the application process we did not limit ourselves to the lab setup, because it was not large/distributed enough. So, in propose approaches we considered a large scale, highly distributed target environment; and made those approaches compatible with such an environment.

Moreover, the NIST guideline recommends a set of actions for each handling phase. These actions can be realized using a variety of mechanisms. We have studied several mechanisms and discussed their compatibilities with the cloud model. Additionally, we have proposed new approaches that are helpful in fulfilling incident handling requirements.

Furthermore, in this process multiple questions and challenges were raised that can be interesting topics for future work in cloud incident handling and in general security of a cloud environment. We itemize a few of them in the following:

- Statistical measurement and analysis of each approach and study of the exact performance overhead.
- Large scale deployment of OpenStack with its latest release.
- Implementation of proposed approaches as a set of security services, and study their effectiveness for a cloud consumer and the cloud environment in general.
- Study the compatibility of approaches and guidelines to other cloud environments, specifically with those operated by industry or commercial cloud providers (e.g. Amazon, Rackspace, Google App Engine, Azure).

References

- [1] European Network and Information Security Agency. www.enisa.europa.eu/about-enisa, Visited: 15 Feb 2011.
- [2] National information assurance (ia) glossary. CNSS Instruction 4009, Committee on National Security Systems, April 2010.
- [3] About RabbitMQ. <http://www.rabbitmq.com/about.html>, February 2011.
- [4] Cacti, the Complete RRDTool-based graphing solution. <http://www.cacti.net/>, April 2011.
- [5] Cloud Security Alliance. <http://www.cloudsecurityalliance.org/>, February 2011.
- [6] Cloudfit project summary. <http://cloudfit.di.fc.ul.pt/index.php?title=Public>About>, March 2011.
- [7] libvirt Wiki. http://wiki.libvirt.org/page/Main_Page#libvirt_Wiki, March 2011.
- [8] Open cirrus. <https://opencirrus.org/>, June 2011.
- [9] Open vSwitch, An Open Virtual Switch. <http://openvswitch.org/>, March 2011.
- [10] Openstack community. <http://www.openstack.org/community/>, May 2011.
- [11] Puppet labs. <http://www.puppetlabs.com/>, May 2011.
- [12] RabbitMQ and Nova. <http://nova.openstack.org/devref/rabbit.html>, February 2011.
- [13] Rabbitmq core api guide. <http://www.rabbitmq.com/api-guide.html>, May 2011.

- [14] Rabbitmq ssl. <http://www.rabbitmq.com/ssl.html>, May 2011.
- [15] The Committee on National Security Systems. <http://www.cnss.gov/>, March 2011.
- [16] Using heartbeat with mysql and drbd. <http://dev.mysql.com/doc/refman/5.0/en/ha-heartbeat-drbd.html>, March 2011.
- [17] Sanjay Aiyagari, Shahrokh Sadjadi, Matthew Arrot, Rafael Schloming, Mark Atwell, Steven Shaw, Jason Brome, Gordon Sim, Alan Conway, Martin Sustrik, Robert Greig, Carl Trieloff, Pieter Hintjens, Kim van der Riet, John O’Hara, Steve Vinoski, and Martin Ritchie. Advanced message queuing protocol specification. amq-spec, AMQP.org, December 2006. Version 0.9.
- [18] Ross Anderson. ‘trusted computing’ frequently asked questions. <http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html>, August 2003.
- [19] Ruo Ando, Kang Byung, and Youki Kadobayashi. Log analysis of exploitation in cloud computing environment using automated reasoning. In *Proceedings of the 17th international conference on Neural information processing: models and applications - Volume Part II*, ICONIP’10, pages 337–343, Berlin, Heidelberg, 2010. Springer-Verlag.
- [20] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.
- [21] AWS Security Team. Vulnerability Reporting. <http://aws.amazon.com/security/vulnerability-reporting/>, March 2011.
- [22] Paolo Balboni, Kieran McCorry, and W. David Snead. Cloud Computing – Benefits, risks and recommendations for information security. Technical report, European Network and Information Security Agency, November 2009. <http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment/>.
- [23] Stephen G. Batsell, Nageswara S. Rao, and Mallikarjun Shankar. Distributed Intrusion Detection and Attack Containment for Organizational Cyber Security. <http://www.ioc.ornl.gov/projects/documents/containment.pdf>.

References

- [24] Dan Belov. 1 billion core-hours of computational capacity for researchers. <http://googleresearch.blogspot.com/2011/04/1-billion-core-hours-of-computational.html>, June 2011.
- [25] Stefan Berger, Ramon Caceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vtpm: Virtualizing the trusted platform module. Research Report RC23879, IBM Research Division, February 2006.
- [26] Stefan Berger, Ramon Caceres, Dimitrios Pendarakis, Reiner Sailer, Enriquillo Valdez, Ronald Perez, Wayne Schildhauer, and Deepa Srinivasan. Tvdc: Managing security in the trusted virtual datacenter. Research Report RC24441, IBM Research Division, November 2007.
- [27] Yanpei Chen, Vern Paxson, and Randy H. Katz. What's New About Cloud Computing Security? Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley, Jan 2010.
- [28] Ben Tudor Christy Pettey. Gartner Says 60 Percent of Virtualized Servers Will Be Less Secure Than the Physical Servers They Replace Through 2012. <http://www.gartner.com/it/page.jsp?id=1322414>, March 2010.
- [29] Holly Stevens Christy Pettey. Gartner highlights key predictions for it organizations and users in 2010 and beyond. <http://www.gartner.com/it/page.jsp?id=1278413>, January 2010.
- [30] Cloud Security Alliance. Top Threats to Cloud Computing V1.0. Technical report, March 2010. <http://www.cloudsecurityalliance.org/topthreats/>.
- [31] OpenStack Community. Bexar, Release Notes. <http://wiki.openstack.org/ReleaseNotes/Bexar>, March 2011.
- [32] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud Computing: Issues and Challenges. *Advanced Information Networking and Applications, International Conference on*, 0:27–33, 2010.
- [33] Data Privacy Protection Directive. http://ec.europa.eu/justice/policies/privacy/index_en.htm.
- [34] Vish Ishaya Eric Day, Paul Voccio. Distributed scheduler. <http://wiki.openstack.org/DistributedScheduler>, February 2011.

References

- [35] FinCEN. USA PATRIOT Act. http://www.fincen.gov/statutes_regs/patriot/index.html.
- [36] Fortify Software. DEFCON survey reveals vast scale of cloud hacking - and the need to bolster security to counter the problem. <https://www.fortify.com/news-and-events/press-releases/2010/2010-08-24.html>, August 2010.
- [37] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.
- [38] Anne Gentle. Clustering, globalization, and scale-out architecture overview. <http://wiki.openstack.org/Overview>, May 2011.
- [39] G. Golovinsky, S. Johnston, and D. Birk. Syslog Extension for Cloud Using Syslog Structured Data. Internet-Draft, March 2011.
- [40] Bernd Grobauer and Thomas Schreck. Towards incident handling in the cloud: challenges and approaches. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop, CCSW '10*, pages 77–86, New York, NY, USA, 2010. ACM.
- [41] Ask Solem Hoel. Carrot - AMQP Messaging Framework for Python. <https://github.com/ask/carrot/>, February 2011.
- [42] Adam Kliarsky Jeff Reed. Following Incidents into the Cloud. Reading room, The SANS Institute, September 2010.
- [43] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. Lipsin: line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication, SIGCOMM '09*, pages 195–206, New York, NY, USA, 2009. ACM.
- [44] Paul Hoffman Karen Scarfone, Murugiah Souppaya. Guide to Security for Full Virtualization Technologies. Special Publications SP 800-125, NIST, January 2011. <http://csrc.nist.gov/publications/nistpubs/800-125/SP800-125-final.pdf>.
- [45] Tim Grance Karen Scarfone and Kelly Masone. Computer Security Incident Handling Guide. Special Publications SP 800-61 Rev. 1, NIST, March 2008. <http://csrc.nist.gov/publications/nistpubs/800-61-rev1/SP800-61rev1.pdf>.

References

- [46] Laurent Luce. OpenStack Nova internals of instance launching. <http://www.laurentluce.com/?p=227>, January 2011.
- [47] Simon MacMullen. Who are you? authentication and authorisation in rabbitmq. <http://www.rabbitmq.com/blog/2011/02/07/who-are-you-authentication-and-authorisation-in-rabbitmq-231/>, May 2011.
- [48] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical Report SP 800-145 Draft, National Institute of Standards and Technology, Information Technology Laboratory, January 2011.
- [49] M. Michael, J.E. Moreira, D. Shiloach, and R.W. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, march 2007.
- [50] Armando Migliaccio. RabbitMQ High Availability. <http://wiki.openstack.org/RabbitmqHA>, February 2011.
- [51] Barbara Liskov Miguel Castro. Byzantine Fault Tolerance. <http://www.google.no/patents/about?id=6EYPAAAAEBAJ>, 12 2003. US Patent 6671821 B1.
- [52] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.
- [53] Derek Gordon Murray, Grzegorz Milos, and Steven Hand. Improving xen security through disaggregation. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, pages 151–160, New York, NY, USA, 2008. ACM.
- [54] NetSNMP. SNMPD. <http://www.net-snmp.org/docs/man/snmpd.html>, June 2005.
- [55] Tobias Oetiker. Round-Robin Database Tool. <http://www.mrtg.org/rrdtool>, April 2011.
- [56] OpenStack. *OpenStack Compute Administration Manual*, bexar release edition, Feb 2011. docs.openstack.org.
- [57] OpenStack. *OpenStack Object Storage Administration Manual*, bexar release edition, Feb 2011. docs.openstack.org.

References

- [58] Bjarne E. Helvik Peder J. Emstad, Poul E. Heegaard and Laurent Paquereau. *Dependability and performance in information and communication systems - Fundamentals*. Tapir akademisk forlag, 2010.
- [59] Tim Grance Peter Mell. Effectively and securely using the cloud computing paradigm, July 2009.
- [60] Personal Information Protection and Electronic Documents Act. <http://laws.justice.gc.ca/en/P-8.6/>.
- [61] Larry Ponemon. Security of Cloud Computing Users: A Study of U.S. and Europe IT Practitioners. Industry research, CA Technologies, May 2010. <http://www.ca.com/us/collateral/industry-research/na/security-of-cloud-computing-users/a-study-of-us-and-europe-it-practitioners.aspx>.
- [62] IDC Press. Virtualization Market Accelerates Out of the Recession as Users Adopt "Virtualize First" Mentality. <http://www.idc.com/getdoc.jsp?containerId=prUS22316610>, Apr 2010.
- [63] Jeff Reed. Following Incidents into the Cloud. Security reading room, SANS Institute, 2011. http://www.sans.org/reading_room/whitepapers/incident/incidents-cloud_33619.
- [64] IBM X-Force's research and development teams. IBM X-Force's 2010 Mid-Year Trend and Risk Report. Technical report, IBM, August 2010.
- [65] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security, CCS '09*, pages 199–212, New York, NY, USA, 2009. ACM.
- [66] Dmitriy Samovskiy. Introduction to amqp messaging with rabbitmq, July 2008.
- [67] Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. Towards trusted cloud computing. In *HOTCLOUD*. USENIX, 2009.
- [68] Amazon Web Services. AWS in Education. <http://aws.amazon.com/education/>, June 2011.
- [69] SFlow. Cloud-scale Performance Monitoring. <http://blog.sflow.com/2010/09/cloud-scale-performance-monitoring.html>, September 2010.

References

- [70] sFlow. sFlow. <http://www.sflow.org/>, April 2011.
- [71] Andrew Clay Shafer. "OpenStack: Philosophy & Implementation". Presented as the Cloud Services SIG: An overview of OpenStack by Andrew Shafer of CloudScaling, 2010.
- [72] Gary Stoneburner, Alice Goguen, and Alexis Feringa. Risk Management Guide for Information Technology Systems. Special publication 800-30, National Institute of Standards and Technology, July 2002.
- [73] Jamie Strandboge. AppArmor. <https://wiki.ubuntu.com/AppArmor>, March 2011.
- [74] Aryan TaheriMonfared. Monitoring Intrusions and Security Breaches in Highly Distributed Cloud Environments. March 2011.
- [75] Takabi, H. and Joshi, J.B.D. and Gail-Joon Ahn. SecureCloud: Towards a Comprehensive Security Framework for Cloud Computing Environments. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*, pages 393 –398, July 2010.
- [76] Yuesheng Tan, Dengliang Luo, and Jingyu Wang. Cc-vit: Virtualization intrusion tolerance based on cloud computing. In *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*, pages 1 –6, December 2010.
- [77] The MITRE Corporation. Common Event Expression. <http://cee.mitre.org/>, April 2011.
- [78] Fang Liu Tong, Jian Mao, Robert Bohn, John Messina, and Dawn Leaf. Nist cloud computing reference architecture. http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/ReferenceArchitectureTaxonomy/NIST_CC_Reference_Architecture_v1_March_30_2011.pdf, March 2011. Version 1.
- [79] Carl Trieloff, Ciaran McHale, Gordon Sim, Harold Piskiel, John O'Hara, Jason Brome, Kim van der Riet, Mark Atwell, Martin Lucina, Pieter Hintjens, Robert Greig, Sam Joyce, and Sanjay Shrivastava. Advanced message queuing protocol protocol specification. amq-spec, AMQP.org, July 2006. Version 0.8.

References

- [80] Toshikazu Uemura, Tadashi Dohi, and Naoto Kaio. Availability analysis of a scalable intrusion tolerant architecture with two detection modes. In Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong, editors, *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, volume 5931 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2009.
- [81] Lennert Buytenhenk Uwe BÄhme. Linux BRIDGE-STP-HOWTO. <http://www.faqs.org/docs/Linux-HOWTO/BRIDGE-STP-HOWTO.html>, January 2001.
- [82] Sandy Walsh. Multiple cluster zones. <http://wiki.openstack.org/MultiClusterZones>, February 2011.
- [83] Matt Welsh. How can academics do research on cloud computing? <http://matt-welsh.blogspot.com/2011/05/how-can-academics-do-research-on-cloud.html>, May 2011.
- [84] OpenStack Wiki. "Architectural Overview", 2010. [Online; accessed 08-Feb-2010].
- [85] OpenStack Wiki. "Nova System Architecture", 2010. [Online; accessed 08-Feb-2010].
- [86] Gansen Zhao, Chunming Rong, Martin Jaatun, and Frode Sandnes. Reference deployment models for eliminating user concerns on cloud security. *The Journal of Supercomputing*, pages 1–16, 2010. 10.1007/s11227-010-0460-9.

Glossary

AppArmor AppArmor is a Mandatory Access Control (MAC) system which is a kernel (LSM) enhancement to confine programs to a limited set of resources. AppArmor's security model is to bind access control attributes to programs rather than to users.

SOURCE: WIKI.UBUNTU.COM/APPARMOR. 66

BLOB binary large object, "a collection of binary data stored as a single entity in a database management system. Blobs are typically images, audio or other multimedia objects, though sometimes binary executable code is stored as a blob." SOURCE: WIKIPEDIA. 38

Cacti Cacti is a complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. SOURCE: WWW.CACTI.NET. 67, 132

HoneyCloud We have used this term to address a cloud specific trap for detecting, deflecting and counteracting an attack. 108, 113, 115

Indication A sign that an incident may have occurred or may be currently occurring. SOURCE: SP 800-61. 65

KVM Kernel-based Virtual Machine, a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). SOURCE: WWW.LINUX-KVM.ORG. 26, 46

libvirt Libvirt is collection of software that provides a convenient way to manage virtual machines and other virtualization functionality, such as storage and network interface management. These software pieces include an API library, a daemon (libvirtd), and a command line utility (virsh). SOURCE: WIKI.LIBVIRT.ORG. 66

Precursor A sign that an attacker may be preparing to cause an incident. SOURCE: SP 800-61. 65

QEMU A generic and open source machine emulator and virtualizer. SOURCE: WIKI.QEMU.ORG. 46

rpc.call Request-Response implementation of the RPC. 27, 28

rpc.cast One-way implementation of the RPC. 27, 28

RRDTool (Round-Robin Database Tool) RRDtool is the OpenSource industry standard, high performance data logging and graphing system for time series data. RRDtool can be easily integrated in shell scripts, perl, python, ruby, lua or tcl applications. SOURCE: WWW.MRTG.ORG/RRDTOOL. 67, 132

S3 (Simple Storage Service) S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, secure, fast, inexpensive infrastructure that Amazon uses to run its own global network of web sites. SOURCE: AWS.AMAZON.COM. 38, 125

Side Channel A specific type of covert channels that passively observing information and is based on timing channels.. 13

Acronyms

AMI Amazon Machine Image. 112

AMQP Advanced Message Queuing Protocol. 11, 22, 24, 26, 27, 30–36, 95, 97, 105, 106, 115

BFT Byzantine Fault Tolerance. 73

CLI Command Line Interface. 96, 103

DHCP Dynamic Host Configuration Protocol. 40, 41

DoS Denial of Service. 49, 50, 56

EC2 Elastic Compute Cloud. 116, 125

eki Eucalyptus Kernel Image. 129

emi Eucalyptus Machine Image. 129, 130

ENISA European Network and Information Security Agency. 17

eri Eucalyptus Ramdisk Image. 129

FIFO First In First Out. 101

HI Host Identity. 40

HTTP Hypertext Transfer Protocol. 29, 30

IaaS Infrastructure as a Service. 8, 18, 25, 26, 47, 51, 56, 59, 69

IDPS Intrusion Detection Prevention System. 65

IDS Intrusion Detection System. 57, 114

IPC Inter-process communication. 25

MAC Media Access Control. 131

NaaS Network as a Service. 22

NIC Network Interface Controller. 40, 54

NIST National Institute of Standards and Technology. 1, 3, 4, 6, 8, 10, 13, 56, 65, 74–76, 149

NTP Network Time Protocol. 68

PPA Personal Package Archives. 122

QoS Quality of Service. 51–54, 59, 64, 70, 92, 97

RBAC Role-based Access Control. 35

REMH Redundant Execution on Multiple Host. 71, 115

RESH Redundant Execution on Single Host. 71

RPC Remote Procedure Call. 27, 29, 131

SaaS Software as a Service. 22

SLA Service Level Agreement. 54, 78

SN Shared Nothing. 22

SNMP Simple Network Management Protocol. 122, 124, 133

SNMPD Simple Network Management Protocol Daemon. 132

SOC Service Oriented Computing. 4

TCCP Trusted Cloud Computing Platform. 113

TVD Trusted Virtual Domain. 113

TVMM Trusted Virtual Machine Monitor. 113

UML User-mode Linux. 26, 46

UUID Universally Unique Identifier. 28

Appendix A - Paper

As strong as the weakest link: Handling compromised components in OpenStack

Aryan TaheriMonfared
Department of Telematics

Norwegian University of Science and Technology
taherimo@stud.ntnu.no

Martin Gilje Jaatun
SINTEF ICT

Trondheim, Norway
Martin.G.Jaatun@sintef.no

Abstract—This paper presents an approach to handle compromised components in an Infrastructure-as-a-Service Cloud Computing platform. Our experiments show that traditional incident handling procedures are applicable for cloud computing, but need some modification to function optimally.

I. INTRODUCTION

One of the main obstacles in the movement toward Cloud Computing is its security challenges. Although it has been argued [1] that most of the security issues in Cloud Computing are not fundamentally novel, a new computing model invariably brings its own security doubts and issues to the market.

In a distributed environment with several stakeholders, there will always be numerous ways of attacking and compromising a component, and it is not possible to stop all attacks or to ensure that the system is secure against all threats.

Thus, the best approach is to understand impacts and assess the risk of a compromised component. So, we don't study attack methods, instead impacts of a compromised component on the provided service and other components will be analyzed. In order to study impacts of a successful attack, exact functionalities of each component are extracted.

After identifying impacts of a successful attack, we should find efficient approaches to tolerate such an attack and its damages. In this process, the incident should be detected and analyzed first. Detecting and analyzing an incident have a standard procedure that requires knowledge about the normal behavior and operation of the system. The next step is containing the incident. ...

There are currently several public cloud providers, however none of them disclose their security mechanisms. Thus, we should study applicable mechanisms and introduce new ones to fulfill security requirements of our experimental cloud environment. Publishing these approaches, other researchers can also analyze them and make them more robust.

When we talk about a compromised component in this document, we mean those components in a cloud environment that are disclosed, modified, destroyed or even lost. Finding compromised components and identifying their impacts on a cloud environment is crucial.

We have found the OpenStack cloud platform as the best choice for a real case study in our research. In our laboratory configuration, we used the simple flat structure. This will avoid

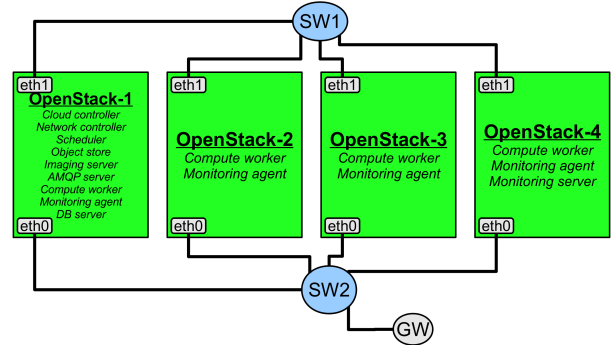


Fig. 1. Lab setup

further complexity which is caused by the hierarchical or peer to peer architecture. We have four physical machines, one of them will be the cloud controller, and other three are compute worker nodes. The abstract diagram of our lab setup is depicted in Figure 1.

It should be noted that although we focus on the OpenStack as a specific cloud software in our study, more or less same components and processes may be identified in other cloud platform implementations.

II. INCIDENT HANDLING

We will in the following focus on cloud platform components, functionalities, connected components, access methods and their impacts in case of being compromised. The symptoms of a compromised component are useful in detecting security breaches and must be considered when performing further analysis.

A. Actors' Requirements

Studying the detection and analysis phase of the NIST incident handling guideline [2], and applying new characteristics of Cloud Computing model, we identified several requirements for a cloud provider and a cloud consumer.

1) Cloud providers' requirements:

- **Security APIs:** The cloud provider should develop set of APIs that deliver event monitoring functionalities and also provide forensic services for authorities. Event monitoring APIs ease systematic incident detection for cloud consumers and even third parties. Forensic services

at virtualization level can be implemented by means of virtual machine introspection libraries. An example of an introspection library is XenAccess that allows a privileged domain to access live states of other virtual machines. A cross-layer security approach seems to be the best approach in a distributed environment [3].

- **Precursor or Indication Sources:** The cloud provider deploys, maintains and administrates the cloud infrastructure. The provider also develops required security sensors, logging and monitoring mechanisms to gather enough data for incident detection and analysis at the infrastructure level. As an example, security agents, intrusion monitoring sensors, application log files, report repository, firewall statistics and logs are all part of security relevant indication sources. In case of a security incident, the cloud provider should provide raw data from these sources to affected customers and stakeholders. Thus they will be capable of analyzing raw data and characterizing incident properties.
- **External reports:** The cloud provider should provide a framework to capture external incident reports. These incidents can be reported by cloud consumers, end users or even third parties. This is not a new approach in handling an incident, however finding the responsible stakeholders for that specific incident and ensuring correctness of the incident¹ require extensive research. An illustration, Amazon has developed "Vulnerability Reporting Process"[4] which delivers same functionalities as described before.
- **Stakeholder interaction:** A timely response to an incident requires heavy interaction of stakeholders. In order to ease this interaction at the time of crisis, responsibilities of each stakeholder should be described in detail.
- **Security services:** Cloud consumers may not be interested in developing security mechanisms. The cloud provider can deliver a security service to overcome this issue. Security services which are delivered by the provider can be more reliable in case of an incident and less challenging in the deployment and the incident detection/analysis.
- **Infrastructure information:** When the cloud consumer or another third party wants to develop an incident detection and analysis mechanisms, they may need to understand the underlying infrastructure and its architecture. However, without cloud provider cooperation that won't be feasible. So, the cloud provider should disclose enough information to responsible players to detect the incident in a timely fashion and study it to propose the containment strategy.

2) *Cloud consumers' requirements:* A cloud consumer must fulfill requirements to ensure effectiveness of the incident detection and analysis process.

- **Consumer's security mechanisms:** The cloud consumer might prefer to develop its own security mechanisms

(e.g. incident detection and analysis mechanisms). The customer's security mechanisms can be based on either the cloud provider's APIs or reports from a variety of sources, including: provider's incident reports, end-users' vulnerability reports, third parties' reports.

- **Provider's agents in customer's resources:** By implementing provider's agents, the cloud consumer will facilitate approaching a cross-layer security solution. In this method, the cloud consumer will know the exact amount and type of information that has been disclosed. Moreover, neither the cloud consumer nor the provider needs to know about each others' architecture or infrastructure design.
- **Standard communication protocol:** In order to have a systematic incident detection and analysis mechanisms, it is required to agree on a standard communication protocol that will be used by all stakeholders. This protocol should be independent of a specific provider/customer.
- **Report to other stakeholders:** If the customer cannot implement the provider's agent in its own instances, another approach to informing stakeholders about an incident is by means of traditional reporting mechanisms. These reports should not be limited to an incident only, customers may also use this mechanism to announce a suspicious behavior for more analysis.
- **Cloud consumer's responsibilities:** Roles and responsibilities of a cloud consumer in case of an incident should be defined previously, facilitating immediate reaction in a crisis.

B. Containment of the compromised component

Cloud consumers' allocated resources are not under their direct/physical control. Consumers control their resources using several access methods which may get compromised as well. Specifically in the IaaS service model, the issue is more challenging for responsible organizations (i.e. providers). One of the main reasons is the increased control of a cloud consumer over its allocated resources and virtual instances [5]. The cloud consumer may develop some procedures for containing its service in case of an incident, but applying these procedures is challenging as well. The cloud provider has to ensure that recent changes in the normal operation of a specific service is due to an incident and not a false positive.

We have identified several aspects that should be considered in this phase:

- 1) We should address the greatest risks and strive for sufficient risk mitigation at the lowest cost, with minimal impact on other mission capabilities [6].
- 2) The containment, eradication, and recovery should be done in a cost effective fashion. Thus, a cost-benefit analysis of each approach should be performed before application.
- 3) In a highly distributed system such as a cloud environment, we cannot apply stateful measures, they won't scale.

¹Avoiding false positive alarms

- 4) It is not feasible to stop all attacks or secure all components to avoid exploiting any existing vulnerabilities.
- 5) In addition to the previous item, existing security mechanisms are not completely applicable to the new computing model and they cannot protect the system from all attacks and cannot provide a fast reactive response to an incident.
- 6) As we cannot harden a cloud environment against all possible attacks, containment strategies and tolerating a successful attack are required approaches.

Our study approach is a case-based one, because:

- Several components, with different functionalities, may require a variety of containment realization mechanisms.
- Providing a single mechanism to cover all incidents, is not possible.
- A combination of mechanisms is possible, and also recommended for covering an attack which exploits several vulnerabilities.
- In each case, we will study different ways of an incident occurrence (e.g. malicious code can be injected in to either a cloud platform service (nova-compute) or OS modules/services.)

C. Case studies

1) *Case One: A Compromised Compute Worker:* The first case which we will discuss, has only one compromised component. In this case the nova-compute service in the compute worker is compromised, Figure 2.

Two incidents have happened simultaneously in this scenario, malicious code and unauthorized access. The malicious code is injected to the nova-compute service and introduces some misbehavior in it, such as malfunctions in the hosting service of virtual instances, nefarious usage of granted privileges to request for more IP addresses and cause IP address exhaustion in a specific consumer's project.

The malicious code is injected by means of another incident, unauthorized access. The attacker gains access to resources on the OpenStack-4 host, that he/she was not intended to have. Using those escalated privileges, the attacker changed the python code of the nova-compute and restarted the service. Thus, nova-compute started to behave maliciously.

2) *Case Two: A bogus component:* A bogus service is a threat for the cloud environment security. As the OpenStack is an open source software, an attacker can access the source code or its binaries and deploy a cloud platform service. When the attacker is managing a service, he/she can manipulate the service in a way that threaten the integrity and confidentiality of the environment. This section will discuss such an incident that a bogus cloud platform component is added to the environment. We will focus on a nova-compute service as the bogus cloud platform component.

A bogus nova-compute service or in general any cloud platform component can run on a physical machine or a virtual instance. Adding a physical node to the cloud infrastructure by an attacker, is unlikely; however, for the sake of completeness we study both the case that the bogus service is running on

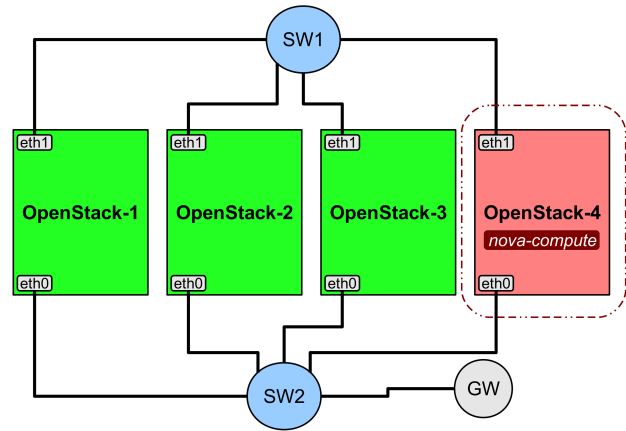


Fig. 2. Case One - The nova-compute service in the OpenStack-4 host is compromised.

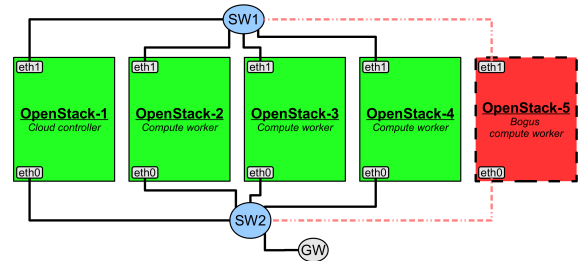


Fig. 3. Case Two - A physical bogus compute worker node is added to the infrastructure.

a new physical machine and the one when it is running on a virtual instance. Both cases are depicted in Figures 3, and 4.

III. APPROACHES

We have devised a set of approaches which will be explained in detail in the following.

A. Restricting infected components

A general technique for containing an incident is restricting the infected component. The restriction can be applied in different layers, with a variety of approaches, such as: filtering in the AMQP server, filtering in other components, disabling

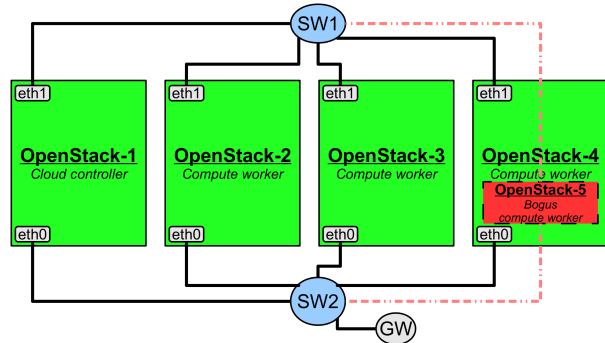


Fig. 4. Case Two - A virtual bogus compute worker is added as a consumer's instance.

the infected service or the communicator one. Additional measures can also be employed to support the restriction, like: removing infected instances from the project VLAN, disabling live migration, or quarantining infected instances.

We explain each of these approaches in the following sections.

1) *Filtering in the messaging server (cloud controller)*: We will propose several filtering mechanisms in the messaging server in order to contain and eradicate an incident in a cloud environment. The OpenStack platform has been used to build our experimental cloud environment. This approach is a responsibility of the cloud provider and the target layer in the cloud platform application layer.

a) *Advantages*:

- The filtering task at the messaging server level can be done without implementation of new functionalities. We can use existing management interfaces of the RabbitMQ (either CLI or web interface) to filter the compromised component.
- The filtering task can be done in a centralized fashion by means of the management plug-in, although we may have multiple instances of the messaging server.
- Implementing this approach is completely transparent for other stakeholders, such as cloud consumers.
- We can scale out² the messaging capability by running multiple instance of the RabbitMQ on different nodes. Scaling out the messaging server will also scale out the filtering mechanism³.
- This approach is at the application layer, and it is independent of network architecture and employed hardware.
- The implementation at the messaging server level helps in having a fine-grained filtering, based on the message content.

b) *Disadvantages*:

- A centralized approach has its own disadvantages as well, such as being a single point of failure or becoming the system bottleneck.
- Implementing the filtering mechanism at the messaging server and/or the cloud controller adds an extra complexity to these components.
- When messages are filtered at the application layer in the RabbitMQ server, the network bandwidth is already wasted for the message that has an infected source, destination, or even context. Thus, this approach is less efficient comparing to the one that may filter the message sooner (e.g. at its source host, or in the source cluster)
- Most of the time application layer approaches are not as fast as hardware layer one. In a large scale and distributed environment the operation speed plays a vital role in the system availability and QoS.

It is possible to use the zFilter technique as a more efficient implementation of the message delivery technique.

²Scaling out or horizontal scaling is referred to the application deployment on multiple servers [7].

³But it may require a correlation entity to handle the filtering tasks among all messaging servers.

It can be implemented on either software or hardware. The zFilter is based on the bloom-filter data structure. Each message contains its state; thus this technique is stateless [8]. It also utilizes source routing. zFilter implementations are available for the BSD family operating systems and the NetFPGA boards in the following address, <http://www.psirp.org>.

- Filtering a message without notifying upper layers, may lead to timeout trigger and resend requests from waiting entities. It can also cause more wasted bandwidth.

c) *Realization*: A variety of filtering mechanisms can be utilized in the messaging server; each of these mechanisms focuses on a specific component/concept in the RabbitMQ messaging server. We can enforce the filtering in messaging server *connection*, *exchange*, and *queue* that will be discussed next.

- **Connection**: A connection is created to connect a client to an AMQP broker [9]. A connection is a long-lasting communication capability and may contain multiple channels [10]. By closing the connection all of its channels will be closed as well.
- **Exchange**: An exchange is a message routing agent which can be durable, temporary, or auto-deleted. Messages are routed to qualified queues by the exchange. A Binding is a link between an exchange and a queue. An exchange type can be one of *direct*, *topic*, *headers*, or *fanout*. [11]

An exchange can be manipulated in different ways in order to provide a filter mechanisms for our cloud environment:

- **Unbinding a queue from the exchange**: The compromised component queue won't receive messages from the unbinded exchange.
- **Publishing a warning message**: Publishing an alert message to that exchange, so all clients using that exchange will be informed about the compromised component. Thus, by specifying the compromised component, other clients can avoid communicating with it. The main obstacle in this technique is the requirement for implementing new functionalities in clients.
- **Deleting the exchange**: Deleting an exchange will stop routing of messages related to it. It may have multiple side effects, such as memory overflow and queue exhaustion.
- **Queue**: Queue is called as a "weak FIFO" buffer, that each message in it can be delivered only to a single client unless re-queuing the message [11].
 - **Unbinding** a queue from an exchange avoids further routing of messages from that exchange to the unbind-ed queue. We can unbind the queue which is connected to the compromised component and stop receiving messages by the infected client.
 - **Deleting** a queue not only removes the queue itself, but also remove all messages in the queue and cancel

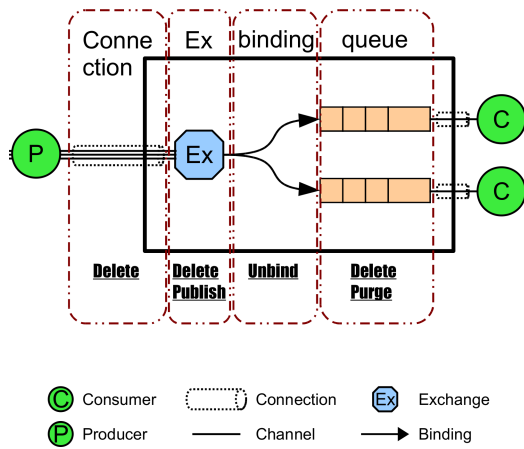


Fig. 5. Overview of RabbitMQ messaging server and applicable containment approaches.

all consumers on that queue.

- **Purging** a queue removes all messages in the queue that do not need acknowledgment. Although it may be useful in some cases, it may not be as effective as required in occurrence of an incident.

Figure 5 depicts a simplified overview of messaging server internal entities and the application points of our approaches.

2) *Filtering in each component:* Applicable filtering mechanisms in the messaging server have been studied in the previous section. This section discusses mechanisms that are appropriate for other components. These components are not essentially aware of messaging technique details and specifications.

a) *Advantages:*

- The implementation of the filtering mechanism in each component avoids added complexity to the messaging server and cloud controller.
- This approach is a distributed solution without a single point of failure in contrast to the previous one with a centralized filtering mechanism.
- Assuming locality principle in the cloud, wasted bandwidth is limited into a cluster/rack which host the infected components. Network connections have much higher speed in a rack or cluster.
- This approach does not require a correlation/coordination entity for filtering messages. Each component behaves independently and autonomously upon receiving an alarm message, that announces a compromised node.

As there is no boundary in the cloud, performing security enforcement at each component is a more reliable approach. Traditionally, most security mechanisms have been employed at the organization/system boundaries. However, as the realization of boundaries is becoming weaker in a cloud environment, this approach is a reasonable one to fulfill the new requirements.

b) *Disadvantages:*

- When the filtering must be performed in each component, all interacting components must be modified to support the filtering mechanism. However, this issue can be relaxed by using a unified version of messaging client (e.g. pika python client) and modifying the client in case of new requirements.
- The message which should be discarded traverses all the way down to the destination, and wastes the link bandwidth on its route.
- Dropping a message without notifying upper layers, may lead to timeout trigger and resend requests from waiting entities. It can also cause more wasted bandwidth.

c) *Realization:* This approach can be implemented at two different levels: blocking at either the messaging client level (e.g. AMQP messaging client) or the OpenStack component/service level.

First, the responsible client can be modified to drop messages with specific properties (e.g. infected source/destination). As an example, the responsible client for AMQP messaging in the OpenStack is amqplib/pika; we must implement the mechanism in this AMQP client (or its wrapper in the OpenStack) to filter malicious AMQP messages. Using this method, more interaction between the OpenStack and clients may be required to avoid resend requests. Because of using the same AMQP client in all components, the implementation is easier and its modification process needs less effort.

The second method is to develop the filtering in each of the OpenStack components, such as nova-compute, nova-network, nova-scheduler, etc. This method adds more complexity to those components and it may not be part of their responsibilities.

We propose a combination of these methods. Implementing the filtering mechanism in the carrot/amqplib wrapper of the OpenStack has advantages of both methods and avoids unnecessary complexity. The OpenStack wrapper for managing AMQP messaging is implemented in `src/nova/rpc.py`. In order to identify the malicious message, we use the message address which is part of its context. Then, the actual dropping happens in the `AdapterConsumer` method. Assuming that the source address is set in the context variable, filtering is straight forward. By checking the message address and avoiding the method call, most of the task is done. The only remaining part is to inform the sender about the problem, that can be implemented by means of the existing message reply functionality.

3) *Disabling services:* Disabling services is a strategy for containing the incident. The disabled service can be either the infected or the communicator one. The communicator service handles tasks distribution and delegation. This method can be used only by the cloud provider, and is at the application layer.

a) *Disabling an infected service:* An incident can be contained by disabling the infected service. It has several advantages, including:

- After stopping the nova-compute service, running instances will continue to work. Thus, as a result con-

- sumers' instances will not be terminated nor disrupted.
- All communications to and from the compromised node will be stopped. So, the wasted bandwidth will reduce massively.
- Shutting down a service gracefully, avoids an extra set of failures. When the service is stopped by Nova interfaces, all other components will be notified and the compromised node will be removed from the list of available compute workers.

Like any other solution, it has multiple drawback as well, including:

- Keeping instances in the running status can threaten cloud consumers. The attacker may gain an access to running instances on the compromised node.
- The live migration feature will not work anymore. Thus, the threatened consumers cannot migrate running instances to a safe or quarantine compute worker node.
- Neither the cloud provider nor consumers can manage running instances through the OpenStack platform.

This approach requires no further implementation, although we may like to add a mechanisms to turn services on and off remotely.

b) Disabling a communicator service: An incident can be contained by disabling or modifying its corresponding communicator service. An example of a communicator service in an OpenStack deployment nova-scheduler service. The nova-scheduler decides that which worker should handle the newly arrived request, such as running an instance.

By adding new features to the scheduler service, the platform can avoid forwarding request to the compromised node. Advantages of this approach are:

- No more requests will be forwarded to the compromised node.
- Consumers' instances remain in the running status on the compromised node. So, consumers will have enough time to migrate their instances to a quarantine worker node or dispose their critical data. Even estimate impacts of the incident.
- This approach can be used to identify the attackers, hidden system vulnerabilities, and the set of employed exploits. In other words, it can be used for forensic purposes.

And its disadvantages are:

- New features should be implemented. These new features are more focused on the decision algorithm of the scheduler service.
- This approach will not secure the rest of our cloud environment, but it avoids forwarding new requests to the compromised node. However, this drawback can be seen as an opportunity. We can apply this approach and also move the compromised node to a **HoneyCloud**. In the HoneyCloud we don't restrict the compromised node, instead analyze the attack and attacker's behavior. But even by moving the compromised node to a HoneyCloud, hosted instances on that node are still in danger.

It is possible that consumers' instances are all interconnected. Thus, those running instances, on the compromised node in the HoneyCloud, threaten the rest of consumers' instances. The rest of instances may even be hosted on a secure worker node. The next proposed approach is a solution for this issue.

4) Removing instances from the project VLAN: This approach does not contain the compromised node, instead focuses on containing instances hosted by the compromised worker node. This is important because those instances may have been compromised as well. The first step toward securing the consumer's service is to disconnect potentially infected instances.

The main usecase of this approach is when the attacker disrupts other solutions (i.e. disabling nova-compute management functionalities, escalated privileges at the OS layer), or when instances and the consumer's service security is very important (e.g. eGovernment services).

It has several advantages specifically for cloud consumers, including:

- Disconnect potentially infected instances from the rest of consumer's instance.
- It does not require new features implementation.
- The attacker cannot disrupt this method.

And its disadvantages are as follows:

- This method only works in a specific OpenStack networking mode (i.e. VLANManager networking mode).
- The consumer completely loses control over isolated instances, that may lead to data loss or disclosure, service unavailability, etc.

5) Disabling live migration: Live migration can cause wide-spread infection, or can be a mechanism for further intrusion to a cloud environment. It may take place intentionally or unintentionally (e.g. an affected consumer may migrate instances to resolve the attack side effects, or the attacker that has the consumer privileges migrates instances to use a hypervisor vulnerability and gain control over more nodes). Disabling this feature helps the cloud provider to contain the incident more easily, and keep the rest of the environment safer.

6) Quarantining instances: When we migrate instances from a compromised node, we cannot accept the risk of spreading infection along instance migration. Thus, we should move them to a quarantine worker node first. The quarantine worker node has specific functionalities and tasks, including:

- This worker node limits instances connectivity with the rest of cloud environment. As an example, only cloud management requests/responses are delivered by the quarantine host.
- It has a set of mechanisms to check instances' integrity and healthiness. These mechanisms can be provided by the underlying hypervisor, cloud platform, or third parties' services.

B. Replicating services

An approach to overcome the implications of an incident is replicating services. A service in this section is a service which is delivered and maintained by the cloud provider. It can be a cloud platform service (e.g nova-compute) or any other services that concerns other stakeholders. The replication can be done passively or actively, and that is due to new characteristics of the cloud model. The replication of a cloud service can be done either at the physical or virtual machine layer.

1) *Replicate services on physical machines:* Replicating service on physical machines is already done in a platform such as the OpenStack. The provider can replicate cloud services either passively or actively when facing an issue in the environment.

2) *Replicate services on virtual machines:* Replication of service on virtual machines has multiple benefits, including:

- Virtual machines can be migrated while running (i.e. live migration), this is a practical mechanism for stateful services that use memory.
- Replication at the instance layer is helpful for forensics purposes. It is also possible to move the compromised service in conjunction with the underlying instance to a HoneyCloud. This is done instead of moving the physical node, ceasing all services on it, and changing the network configuration in order to restrict the compromised node communication.
- Using virtual machines in a cloud environment we can also benefit from the cloud model elasticity and on demand access to computing resources.

This approach is also the main idea behind the CC-VIT [17]. By applying the CC-VIT to our environment, the preferred hybrid fault model will be REMH, and the group communication is handle using the AMQP messaging.

We can use physical-to-virtual converters to have the advantages of both approaches. These tools convert a physical machine to a virtual machine image/instance that can be run on top of a hypervisor.

Moreover, each of these replicas can be either active or passive. This will have a great impact on the system availability.

C. Disinfecting infected components

Disinfecting an infected component is a crucial task in handling an incident and securing the system. It can be accomplished with multiple methods having a variety of specifications.

None of the following approaches will be used for cleaning the infected binary files, instead less complex techniques are employed that can be applied in a highly distributed environment. Cleaning a binary file can be offered by a third party security service provider, that has focused on large scale antivirus software.

1) **Updating the code**

The service code can be updated to the latest, patched version. This process should be done in a smooth way so

all components will be either updated or remain compatible with each other after partial components update.

Several tools has been developed with this purpose. One of the best examples is the Puppet project [18].

2) **Purging the infected service**

Assuming that the attacker has stopped at the cloud platform layer, by removing the service completely we can assure containment of the incident.

3) **Replacing the service**

Another method which is not as strong as others, is achieved by replacing the infected service with another one that uses a different set of application layer resources, such as configuration files, binaries, etc. Thus, we can be sure that the infected resources have no effect on the new service.

D. Migrating instances

The affected consumer can migrate an specific instance or a set of instances to another compute worker or even another cloud environment. The migration among different provider is an open challenge nowadays, because of the weak interoperability of cloud systems and lack of standard interfaces for cloud services.

In our deployment, both Amazon EC2 APIs and RackSpace APIs are supported. Thus, in theory a consumer can move between any cloud environment provided by the Amazon EC2, RackSpace, and any open deployment of OpenStack without any problem.

E. Node authentication

In this method each worker must have a certificate signed by a trusted authority. This authority can be either an external one or the cloud controller/authentication manager itself. Having a signed certificate, the worker can communicate with other components securely. The secure communication can bring us any of the following: confidentiality, integrity, authentication, and non-reputation.

In this case, worker's communication and authenticity is important for us. For this purpose we can use two different schemes: message encryption or a signature scheme. Each of these schemes can be used for the whole communication or the handshake phase only.

When any of those schemes are applied only to the handshake phase, any disconnection or timeout in the communication is a threat to the trust relation. As an authenticated worker is disconnected and reconnected, we cannot only rely on the worker's ID or host-name to presume it as the trusted one. Thus, the handshake phase should be repeated to ensure the authenticity of the worker.

Although applying each scheme to all messages among cloud components is tolerant against disruption and disconnection, its overhead for the system and the demand for it should be studied case by case.

By applying each of those schemes to all messages, we can tolerate disconnection and disruption. However, using cryptographic techniques for all messages introduce an overhead for the system which may not be efficient or acceptable.

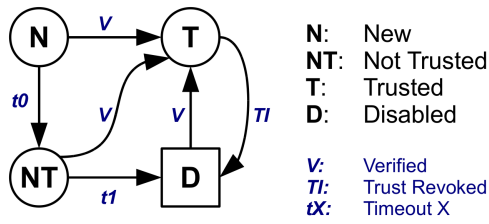


Fig. 6. A sample markov model for trust states of a component.

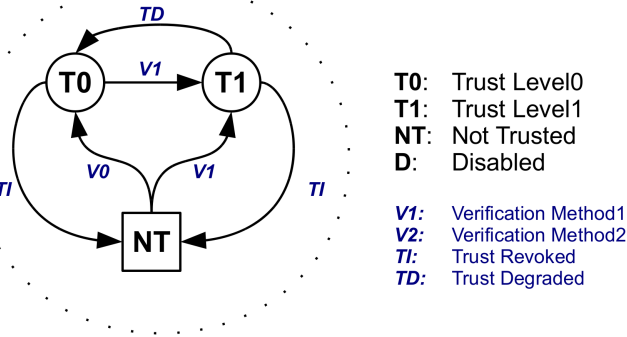


Fig. 7. A sample markov model for transitions between different trust levels of a component.

F. Policies

1) *No new worker policy:* In addition to all those technical approaches, a set of management policies can also relax the issue. As an example, no new worker should be added unless there is a demand for it. The demand for a new worker can be determined when the resource utilization for each zone is above a given threshold.

2) *Trust levels and timeouts:* Introducing a set of trust levels, a new worker can be labeled as a not trusted worker. Workers which are not trusted yet, can be used for hosting non-critical instances, or can offer a cheaper service to consumers.

In order to ensure the system trustworthiness in a long run, a not-trusted worker will be disabled after a timeout. A simple Markov model of those transitions are depicted in Figure 6.

Assuming we have only two trust levels, Figure 7 depicts transitions between them. As an example, T_0 can be achieved by the human intervention; and the second level of trust T_1 is gained by cryptographic techniques or trusted computing mechanisms.

3) *Manual confirmation:* In this method, recently added workers are not used for serving consumers' requests until their authenticity is confirmed by the cloud provider. This method requires human intervention; thus, it can become a bottleneck in the cloud infrastructure. Techniques, explained in the next part, can relax the bottleneck issue.

IV. CONCLUSION

We have presented an approach to handling compromised components in an OpenStack IaaS configuration. Cloud Computing present some unique challenges to incident handling,

but our experience shows that with proper adaptation, traditional incident management approaches can also be employed in a Cloud Computing environment.

REFERENCES

- [1] Y. Chen, V. Paxson, and R. H. Katz, "What's New About Cloud Computing Security?" EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-5, Jan 2010. [Online]. Available: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.html>
- [2] T. G. Karen Scarfone and K. Masone, "Computer Security Incident Handling Guide," NIST, Special Publications SP 800-61 Rev. 1, March 2008, <http://csrc.nist.gov/publications/nistpubs/800-61-rev1/SP800-61rev1.pdf>.
- [3] A. TaheriMonfared, "Monitoring Intrusions and Security Breaches in Highly Distributed Cloud Environments," March 2011.
- [4] AWS Security Team, "Vulnerability Reporting," <http://aws.amazon.com/security/vulnerability-reporting/>, March 2011.
- [5] J. Reed, "Following Incidents into the Cloud," SANS Institute, Security Reading Room, 2011, http://www.sans.org/reading_room/whitepapers/incident/incidents-cloud_33619.
- [6] G. Stoneburner, A. Goguen, and A. Feringa, "Risk Management Guide for Information Technology Systems," National Institute of Standards and Technology, Special Publications, July 2002.
- [7] M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, march 2007, pp. 1–8.
- [8] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "Lipsin: line speed publish/subscribe inter-networking," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 195–206. [Online]. Available: <http://doi.acm.org/10.1145/1592568.1592592>
- [9] "Rabbitmq core api guide," <http://www.rabbitmq.com/api-guide.html>, May 2011.
- [10] C. Trieloff, C. McHale, G. Sim, H. Piskiel, J. O'Hara, J. Brome, K. van der Riet, M. Atwell, M. Lucina, P. Hintjens, R. Greig, S. Joyce, and S. Shrivastava, "Advanced message queuing protocol specification," AMQP.org, amq-spec, July 2006, version 0.8.
- [11] D. Samovskiy, "Introduction to amqp messaging with rabbitmq," July 2008.
- [12] "libvirt Wiki," http://wiki.libvirt.org/page/Main_Page#libvirt_Wiki, March 2011.
- [13] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proc. Network and Distributed Systems Security Symposium*, February 2003.
- [14] S. Berger, R. Cceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vtpm: Virtualizing the trusted platform module," IBM Research Division, Research Report RC23879, February 2006.
- [15] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *HOTCLOUD*. USENIX, 2009.
- [16] S. Berger, R. Cceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan, "Tvdc: Managing security in the trusted virtual datacenter," IBM Research Division, Research Report RC24441, November 2007.
- [17] Y. Tan, D. Luo, and J. Wang, "Cc-vit: Virtualization intrusion tolerance based on cloud computing," in *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*, December 2010, pp. 1–6.
- [18] "Puppet labs," <http://www.puppetlabs.com/>, May 2011.