

Virtual machines image protection in Cloud computing



By
Muhammad Kazim
2011-NUST-MS-CCS-23

Supervisor
Dr. Muhammad Awais Shibli
Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree
of Masters in Computer and Communication Security (MS CCS)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(February, 2014)

Approval

It is certified that the contents and form of the thesis entitled “**Virtual machines image protection in Cloud computing**” submitted by **Muhammad Kazim** have been found satisfactory for the requirement of the degree.

Advisor: Dr. Muhammad Awais Shibli

Signature: _____

Date: _____

Committee Member 1: Dr. Abdul Ghafoor

Signature: _____

Date: _____

Committee Member 2: Dr. Hamid Mukhtar

Signature: _____

Date: _____

Committee Member 3: Miss Rahat Masood

Signature: _____

Date: _____

Dedication

Dedicated
to
my loving Family
and Friends!

Certificate of Originality

I hereby declare that this submission titled **Virtual machines image protection in Cloud computing** is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or any other education institute, except where due acknowledgment, is made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic is acknowledged. I also verified the originality of contents through plagiarism software.

Author Name: Muhammad Kazim

Signature:_____

Acknowledgment

I am thankful to Almighty Allah for giving us the intellect and strength to carry out this project. I will also like to thank my supervisor Dr. Awais Shibli and my evaluation committee members Dr. Abdul Ghafoor, Dr. Hamid Mukhtar and Ms. Rahat Masood. They have been inspiring teachers and great motivators. They have provided me continuous assistance, inspiration and patience. I am thankful to all members of KTH-AIS lab especially Ayesha Kanwal and Ume Habiba for their help.

Muhammad Kazim

Table of Contents

1	Introduction and Motivation	2
1.1	Cloud Computing	2
1.2	Virtualization	3
1.3	Research Motivation	4
1.4	Problem Statement	5
1.5	Research Contributions	5
1.6	Thesis Organization	6
2	Related Work	8
2.1	Bare Metal Virtualization	8
2.2	Security requirements of Virtualization	9
2.2.1	Service Provider Requirements	9
2.2.2	Hypervisor Requirements	9
2.2.3	Virtual Machine Requirements	10
2.2.4	Disk Image Requirements	10
2.3	Attacks on Virtualization	11
2.3.1	Service Provider Attacks	11
2.3.2	Hypervisor Attacks	12
2.3.3	Virtual Machine Attacks	13
2.3.4	Disk Image Attacks	14
2.4	Security solutions for Virtualization	14
2.4.1	Service Provider Security	14
2.4.2	Hypervisor Security	15
2.4.3	Virtual Machine Security	16
2.4.4	Disk Image Security	18
2.5	Recommendations for secure usage of Virtual Machines	19
2.5.1	Secure Network	19
2.5.2	Disabling the Non-Required Features	19
2.5.3	Disconnect Unused Hardware Devices	20
2.5.4	Backup of Virtual Machine Images	20
2.5.5	Hardening of Virtual Machines	20

2.5.6	Auditing	20
2.6	Industrial Survey	20
2.6.1	Storage Made Easy	21
2.6.2	Piston Cloud	21
2.6.3	Metacloud	22
3	Research Methodology	24
3.1	Introduction	24
3.2	Types of Research Methods	24
3.2.1	Descriptive and Analytical Research	25
3.2.2	Applied and Fundamental Research	25
3.2.3	Quantitative and Qualitative Research	25
3.2.4	Conceptual and Empirical Research	25
3.2.5	Deductive and Inductive Approaches	26
3.3	Thesis Research Methodology	26
3.3.1	Research Objectives	26
3.3.2	Research Methodologies	27
3.3.3	Hypothesis	27
3.3.4	Observations	27
3.3.5	Implementation and Verification of Prototype	28
4	Design and Implementation	29
4.1	Design of Framework	29
4.1.1	Image Encrypt Module (IEM)	29
4.1.2	Image Decrypt Module (IDM)	30
4.1.3	Key Management Server (KMS)	30
4.1.4	Image Encryption through EVDIC	30
4.1.5	Image Decryption through EVDIC	31
4.2	EVDIC in OpenStack	32
4.2.1	EVDIC Image Encryption in OpenStack	33
4.2.2	EVDIC Image Decryption in OpenStack	33
4.3	Implementation	34
4.3.1	Object Upload in Swift	35
4.3.2	Object Download in Swift	35
4.3.3	Swift Source Code	35
4.3.4	Object Encryption in Swift	36
4.3.5	Object Decryption in Swift	36
5	Deployment and Validation through OpenStack	38
5.1	Deployment of OpenStack	38
5.2	Adding objects to OpenStack Swift	39

5.3	Debugging Swift	39
5.4	Code Validation through OpenStack	40
6	Evaluation	44
6.1	Evaluation Methodology	44
6.1.1	Evaluation through NIST Guidelines	45
6.1.2	Threat Model	45
6.2	Verification through Image Exploitation Tools	45
6.3	Performance Evaluation	46
7	Conclusion and Future Directions	50
7.1	Conclusion	50
7.2	Future Research Directions	51
7.2.1	Adding security of accounts in OpenStack Swift	51
7.2.2	Encrypted execution of Virtual Machines in Cloud Computing	52
Appendix A Documentation on Adding Encryption to OpenStack Swift		56
A.1	Deploying OpenStack	56
A.1.1	Adding objects to OpenStack Swift	57
A.2	Debugging OpenStack	60
A.3	OpenStack Object Server Code Breakdown and Object Encryption	62
A.4	Deployment	65
A.5	Testing	66
A.6	Configuring Glance to store virtual machine images	69
A.7	Enable and Disable any service in OpenStack	69
A.8	Using Devstack with proxy server	69
A.9	Maintaining Logs for Devstack	70
A.10	Common Errors	70
A.10.1	E: Could not get lock /var/lib/dpkg/lock - open (11 Resource temporarily unavailable)	70
A.10.2	Invalid Nova Credentials (Unauthorized HTTP 401)	70

List of Figures

1.1	Cloud service models	3
1.2	Top down research approach followed in thesis	6
2.1	Bare metal virtualization architecture	9
2.2	VM Escape attack	12
2.3	VM security by firewall, anti-virus and anti-spyware	16
2.4	Secure VM boot protocol	18
2.5	Securing the saved VM state	18
3.1	Thesis Research Methodology	28
4.1	Disk image encryption through EVDIC	31
4.2	Disk image decryption through EVDIC	32
4.3	Image encryption through EVDIC in OpenStack	33
4.4	Image decryption through EVDIC in OpenStack	34
5.1	Output of the running DevStack	39
5.2	Status of Container named TestContainer	40
5.3	Horizon URL and User Credentials	41
5.4	Image creation process in OpenStack	42
5.5	Successful creation of image on OpenStack	42
5.6	Custom instance successfully running	43
6.1	Exploring the VM image using Archive Manager	46
6.2	Time taken by different images during upload to Swift	47
6.3	Time taken by images during download from Swift	48
A.1	Output of the running DevStack	57
A.2	Status of Container named TestContainer	59
A.3	Horizon URL and User Credentials	66
A.4	Image creation process in OpenStack	67
A.5	Successful creation of image on OpenStack	67

A.6	Successful creation of image on OpenStack	68
A.7	Custom instance successfully running	68

List of Tables

2.1	Summary of security aspects of virtualization	23
6.1	Attacks mitigated with the description of security mechanism	48
6.2	Image upload time in Swift before and after encryption	49
6.3	Image download time in Swift before and after encryption . .	49

Abstract

Virtualization is a primary feature of Cloud computing that enables a single system to concurrently run multiple isolated virtual machines. A virtual machine uses a single file called disk image to represent the hard drive of its operating system. Although, extensive research has been carried out to increase the security of Cloud virtualization, there are still open challenges related to the security of disk images used by virtual machines. Virtual machine images can be compromised in many ways, for instance by unauthorized access, zero day attacks or installation of malicious software. With the increased adoption of Cloud infrastructure in information technology industry, there is an urgent need to safeguard disk images against prospective malicious attacks both for protecting the sensitive customer data and maintaining the integrity of virtual machines.

The contribution of this thesis is two folds. First, we have analyzed the security of Cloud virtualization components including service provider, hypervisor, virtual machines and disk images from three different aspects. These aspects include the security requirements for virtualization, possible attacks on different components and the existing security solutions for the protection of virtualization environment in the Cloud. Therefore, an holistic picture of virtualization security in the Cloud is provided through structured analysis in which security requirements, attacks and solutions correspond to each other.

Secondly, to protect virtual machines images from: infrastructure, hypervisor and storage attacks, we have proposed a security mechanism that encrypts virtual machines images in the Cloud storage. In particular, we have built an encryption system for disk images by using Advanced Encryption Standard AES-256. Our proposed methodology not only preserves the integrity and confidentiality of data in stored disk images but also protects images against attacks. The image is decrypted only when it is required by the virtual machine. Our system is implemented and validated on OpenStack (an open source Cloud computing platform). The performance evaluation of our solution shows that it incurs only a minor overhead of 15 percent.

Chapter 1

Introduction and Motivation

In this chapter we provide the introduction of research work carried during thesis. Cloud computing is introduced along with its benefits. Importance of different service models and deployment models of Cloud is also described. Virtualization is the basis of delivering infrastructure services in Cloud. We discuss its detailed architecture and present our research motivation, research contributions and problem statement in this chapter. This chapter ends with the discussion on thesis organization.

1.1 Cloud Computing

Cloud computing is becoming popular among IT businesses due to its agile, flexible and cost effective services. It has transformed the way the typical IT infrastructure was deployed by combining the technologies such as virtualization, web Services, service oriented architecture and grid computing. Unlike the traditional IT service model, businesses on Cloud can grow rapidly without need of large capital investments for purchasing new PC, servers and other dedicated hardware [1]. Therefore, the major advantages that Cloud computing offers are reduced operational costs, scalability, increased efficiency and better utilization of hardware resources.

Cloud has three service models to define the type of services it provides to users. These models are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Software as a Service (SaaS) allows users to access applications hosted by different vendors on Cloud via internet. Platform as a Service (PaaS) enables developers to code, test and deploy their applications on IaaS. In Infrastructure as a Service (IaaS) model, Cloud providers offer services such as computing, network, storage and databases via internet. IaaS is the base of all Cloud services

with PaaS and SaaS both built upon it [2]. The organization of different service models is shown in figure 1.1 [3]. The main features of IaaS are elasticity and virtualization.

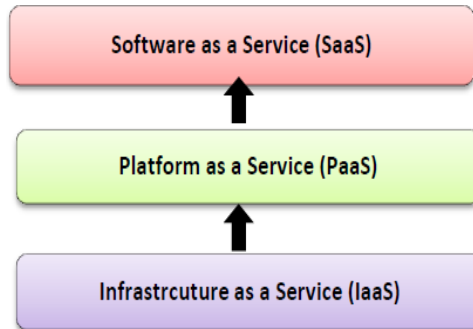


Figure 1.1: Cloud service models

The deployment model of Cloud can also vary depending upon the requirements of provider and users. The major deployment models of Cloud computing are Public Cloud, Private Cloud, Hybrid Cloud and Community Cloud. In Public Cloud model the services are available to everyone on the internet via web services. Vendors of Private Clouds manage infrastructure for one specific company to meet their requirements. Major benefits that Private Clouds offer are security and ease of management. Hybrid Cloud model is the combination of two or more Cloud models by using standardized technology that enables data and application sharing [4].

1.2 Virtualization

Virtualization is the primary component that makes Cloud computing special. Virtualization enables a single system to run multiple isolated virtual machines (VMs), operating systems or multiple instances of a single operating system (OS). It abstracts the underlying hardware to virtually provide interfaces similar to the hardware's physical platform. Virtualization was reintroduced in 1997 by Bugnion E. et al. who used virtual machines to run multiple commodity operating systems on a single PC [5]. It utilizes hardware in the best possible way by maximizing the jobs, which a single CPU can do. Organizations are using virtualization in private, public and hybrid Clouds to gain efficiency in platform and application hosting.

Different virtualization approaches can be applied to various system layers including hardware, desktop, operating system, software, memory, storage, data and network [6]. Hardware virtualization is a form of virtualization that

abstracts the underlying hardware [7]. It can be categorized into full virtualization, partial virtualization and para virtualization. Compared to the two other forms of virtualizations, full virtualization involves complete abstraction of underlying hardware to provide interfaces similar to the hardware's physical platform. Thus, it provides better operational efficiency by putting more work load on each physical system and hence more popularly used for servers virtualization. Full virtualization can be categorized into two forms: i) bare metal virtualization and ii) hosted virtualization.

Hosted and bare metal virtualization are used in software based partitioning approaches in latest UNIX/RISC and industry-standard x86 systems. In the hosted architecture, hypervisor lies on top of the standard operating system. However, running hypervisor on top of the host OS increases security risks and complexity of the system [8]. While in the bare metal approach, hypervisor comes directly on top of the hardware which provides direct access to the hardware resources. Bare metal approach is mostly used for server virtualization in large computing systems like Cloud computing as it provides better performance, more robustness and agility.

1.3 Research Motivation

IaaS model enables customers get the virtual machines on lease from the Cloud providers instead of purchasing physical machines. Virtual machine (VM) is a software container that behaves like a physical machine with its own operating system and virtual resources including CPU, RAM and hard disk. VMs use disk images as their hard drives, which are virtual representation of a physical drive. Virtual disk image is a single file or directory representing the hard drive of a guest operating system (operating system running within a virtual machine). It encapsulates all components of a guest OS, including the applications and virtual resources used by guest OS. The major advantage of using disk images in a Cloud virtualization environment is that VMs can be quickly launched and deployed across various hosts [8].

Despite all the Cloud benefits, there are still open challenges in achieving security for Cloud virtualization. Since multiple VMs run on a single physical machine, impact of attacks on virtualization environment can be much more as compared to that on a single physical system. With the increase in usage of Cloud virtualization in IT industry, there is a need to analyze its security in a more detailed form.

Research has been done to explore major security issues related to virtualization in Cloud. The standard bodies in computing security have issued guidelines on virtualization technologies. These guidelines cover different as-

pects of virtualization security. National Institute of Standard Technologies (NIST) guide [8] mentions security issues and recommendations for securing virtualization environment, whereas the Centre for Internet Security (CIS) guide [9] focuses on virtual machines security and their secure configuration. SANS guide sponsored by VMware [10] provides key configuration and security controls for VMware ESX and vSphere. Cloud Security Alliance (CSA) guide [11] and Payment Card Industry Data Security Standard (PCI DSS) [12] discuss security issues related to virtualization in Cloud and provide recommendations for secure virtualization environments. However, the holistic view of virtualization security has not been presented in a composed form. The impact of Cloud service provider on Cloud virtualization security has also not been discussed. Furthermore, there is need to investigate existing virtualization security solutions proposed in literature to mitigate different attacks.

Along with other virtualization components, disk images are vulnerable to many attacks from outsiders as well as from malicious users and administrators in Cloud. These attacks include unauthorized access to images and installation of malicious software on them. Hence, the integrity, confidentiality and availability of sensitive customer data in Cloud can be compromised. Even though security of disk images in Cloud is a major issue, only few solutions have been proposed in literature for their protection against attacks. In this thesis, we provide a scheme to protect disk images in Cloud storage from various attacks.

1.4 Problem Statement

Virtual machine images are vulnerable to different attacks in Cloud storage. For instance, they can be compromised by infrastructure, hypervisor and storage attacks, resulting in the loss of data integrity and confidentiality. Therefore, VM images must be secured in Cloud storage through best security practices, both for protecting the sensitive customer data and maintaining the integrity of disk images.

1.5 Research Contributions

This thesis has contributed towards Cloud security both from research as well as practical aspects. From research point of view, two research publications have been made. In the first publication, the security analysis of all virtualization components is presented in detail. This analysis will pave a way for

researchers to investigate, develop and improve security solutions for existing Cloud attacks. In the second publication, a security scheme is proposed for the encryption of disk images in Cloud environment. The design can be used to implement disk image encryption in any Cloud platform.

From the practical aspect, we have implemented the scheme of disk image encryption in an open source Cloud computing platform named OpenStack. OpenStack is composed of various independent modules with different functionalities. The image storage component of OpenStack was explored in detail to understand the code flow and to add encryption functionality in it. The encrypted disk images has increased the security of disk images in Cloud. This scheme ensure that the confidentiality of customer data in images as well as The design of this encryption scheme in OpenStack along with possible performance concerns are also explained in a published research paper. The top down approach followed in thesis is described in figure 1.2.

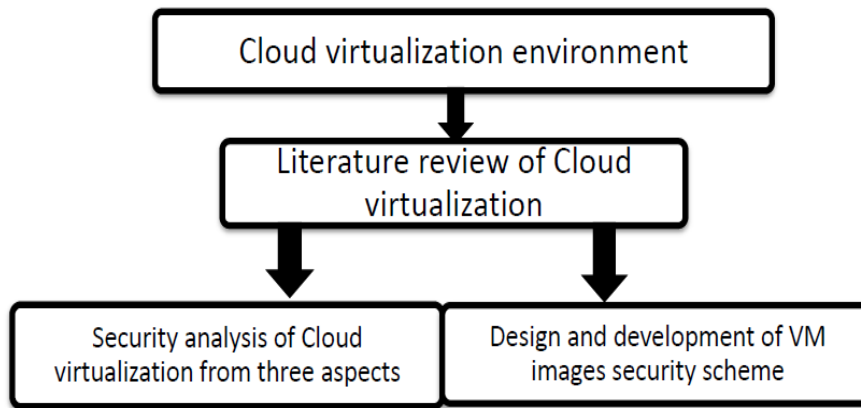


Figure 1.2: Top down research approach followed in thesis

1.6 Thesis Organization

This thesis is organized in a systematic way to clearly state the research approach and contributions made in the thesis. Chapter 2 describes the literature survey regarding Cloud virtualization security and specifically the survey of virtualization components including hypervisor, virtual machines and disk images. Chapter 3 describes the research methodology followed during the thesis to achieve the solution. Different research methodologies were combined to achieve the required results including deductive research and conceptual research methods. In chapter 4 we describe the methodology, design and implementation of the framework proposed during thesis.

We also discuss the components of OpenStack and the detailed architectural extension of OpenStack storage module called Swift. In chapter 5 the results and verification of implementation are provided. We describe the integration of our framework with OpenStack along with the complete testing and verification of source code changes. Chapter 6 provides the evaluation of our research work using a threat model and standard guidelines. We also discuss the performance evaluation of our work. Chapter 7 provides the research conclusion and possible future directions and extensions of our research.

Chapter 2

Related Work

This chapter presents the related work done in the area of Cloud virtualization security. Cloud computing has been a focused area of research from last decade and lot of research has been done in the area of Cloud virtualization. In our research, we have done both literature review of related work as well as industrial survey of related commercial solutions. We followed layered approach to review the literature of Cloud virtualization literature. The first part of the chapter presents general requirements to protect Cloud virtualization environment. The second part discusses different attack scenarios at virtualization. To cater the attacks on virtualization environment different security solutions proposed in literature have been presented and analyzed in this chapter. Moreover, some general recommendations for the security of virtualization in literature are also presented. This chapter ends with the analysis of various commercial solutions for virtualization security.

2.1 Bare Metal Virtualization

Virtualization has different types as discussed in chapter 1. Bare metal approach is used for server virtualization in Cloud. In the bare-metal virtualization architecture shown in figure 2.1, hardware refers to the physical resources such as CPU, RAM, storage disks, physical switches, I/O ports, and BUS systems (PCI, LPC) etc. Hypervisor is composed of resources necessary to run virtual machines including virtual machine manager (VMM), kernel layer and the driver layer [13]. Hypervisor hides hardware resources of the system from the operating system running on it, manages the execution of the guest operating systems and partitions hardware platform into multiple logical units called virtual machines (VMs). Each virtual machine has its own operating system (called guest OS) and applications running on it.

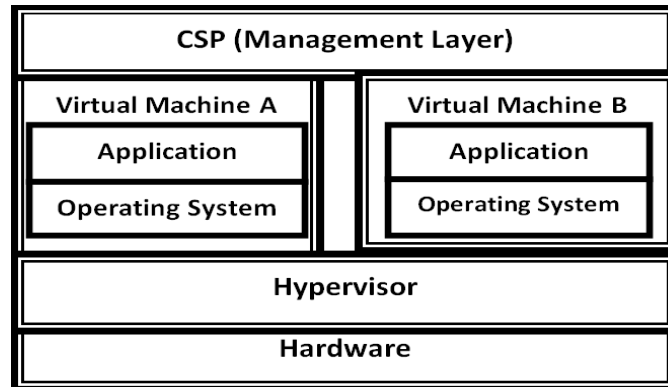


Figure 2.1: Bare metal virtualization architecture

2.2 Security requirements of Virtualization

In general, before planning and implementing security of any system it is important to understand the security requirements of that environment. This section details the security requirements for complete virtualization environment.

2.2.1 Service Provider Requirements

A report by Alert Logic [14] shows that 50 percent of Cloud users consider service provider security as a major threat. However, the impact of Cloud service provider on Cloud virtualization security has also not been discussed comprehensively in literature. To secure the virtualization hardware, (Cloud) service provider must limit access of hardware resources to authorized person. Similarly, proper access control should be implemented in the management layer, so that each administrator has access only to its concerned data and software. The service provider also needs to provide strong authentication mechanisms to users. Furthermore, security principles for the development of trusted computing system such as economy of mechanism, complete mediation, open design, principle of least privilege, psychological acceptability must also be followed by the service provider.

2.2.2 Hypervisor Requirements

Hypervisor provides the necessary resource management functions that enable sharing of hardware resources between the VMs. Hypervisor must maintain the isolation between VMs and support multiplexing of multiple VMs on single hardware platform [15]. It must ensure that no application from

any VM can directly take control of it as a host to modify the source code of hypervisor and other VMs in the network. Hypervisor should also monitor the guest OS and applications in VMs to detect any suspicious behavior [16].

Programs that control the hypervisor must be secured using similar practices used for security of programs running on servers. Similarly access to the hypervisor must be restricted. Other security measures to secure hypervisor include installing updates to the hypervisor, restricting administrator access to the hypervisors management interfaces and analyzing hypervisors logs to see if it is functioning properly [8].

2.2.3 Virtual Machine Requirements

Limit on VM resource usage has to be assigned so that malicious VMs can be restricted from consuming extra resources of the system [12]. Moreover, isolation between virtual machines should be provided to ensure that they run independently from each other. Operating system of the VM (guest OS) must be secured by using the best practices followed for securing the OS of physical machines. These practices include updating the OS regularly for patches and updates, using anti-virus software, securing internet and email, keeping back-up of files, protecting the bootable media by access control, removing unnecessary programs and restricting remote access [17]. Using the configuration standards and tools for securing the OS of physical machines cant provide all security to OS of VMs, so special policies and steps must be taken by the Cloud providers to ensure OS security [8]. Monitoring of guest OS must be done to look for its potential compromise.

Privileged VM (Dom0) is the first domain started by XEN hypervisor after boot. It is responsible for monitoring the communication between the remote users and guest VMs. Dom0 is also responsible for creating and destroying all guest VMs and providing device drivers to the guest VMs. Dom0 should boot the guest VMs without tampering them. The state of the VM saved as a disk file in Dom0 must remain confidential, and it must not be tampered [18].

2.2.4 Disk Image Requirements

Hypervisors use disk images (host les used as disk drive for guest OSs) to present guest OSs with virtual hard drives. Guest OS images can be moved and distributed easily, so they must be protected from unauthorized access, tampering and storage. To securely manage the guest OS images they must be examined and updated regularly according to the requirements. Unnecessary images must not be created and if any image is useless it must be

removed from system [8]. Whenever VM is migrated from one physical machine to another, images on previous disks should be completely removed. Similarly, data on old broken disks should also be removed before they are discarded. Furthermore, backup of the virtual machines images must be maintained.

VM checkpoint is a feature that allows the users to take snapshot of VM image in the persistent storage. Snapshot records the state of the running image that contains all components of the guest OS. Snapshot is generally captured as a difference between the image and the running state. The major function of checkpoint is to restore VM to its previous state if the VM enters any undesired state. However, the snapshot access should be given to authorized users and checkpoint must be used only to return VM to a stable and non-malicious state [19].

2.3 Attacks on Virtualization

Each component of virtualization layer can act as an attack vector to launch multiple attacks on the system. Attacks that target different components of virtualization environment may result in security issues such as compromise of complete Cloud infrastructure, stealing of customer data and system hacking. This section discusses different attack scenarios at virtualization environment in Cloud.

2.3.1 Service Provider Attacks

If the attacker has physical access to the Cloud hardware, he may run malicious application or code in the system to damage the VMs by modifying their source code and changing their functionality. With the help of physical access to system, attackers can also launch cross VM side channel attacks. These attacks include CPU cache leakage to measure the load of other virtual web server on the network [20]. Moreover, if access control is not implemented properly, different administrators such as network admin and virtualization admin might access the customer data that they are not authorized to access. These activities will result in security compromises such as loss of data confidentiality and unauthorized traffic monitoring.

Service provider has to ensure that software deployed on Cloud are built using proper coding practices. Flawed coding can result in web application attacks such as SQL Injection, Cross Site Scripting, Denial of Service and Code Execution etc. Alert Logic [14] report shows web application attacks to be the most common attacks on Cloud environment, impacting almost 52

percent customers.

2.3.2 Hypervisor Attacks

A Cloud customer can lease a guest VM to install a malicious guest OS, which attacks and compromises the hypervisor by changing its source code in order to gain access to the memory contents (data and code) of VMs present in the system [16]. With more features in hypervisor its increased code size has resulted in design and implementation vulnerabilities. To control the complete virtualization environment malicious hypervisors such as BLUEPILL rootkit, Vitriol and SubVir and are installed on the fly, which give attacker the host privileges to modify and control VMs [21]. This technique used by malicious software to take complete control of the underlying operating system by hiding itself from administrator and security software is called hyperjacking [22].

Another attack in which program running in one VM can get root access to the host machine is called VM Escape [8]. It is done by crashing the guest OS to get out of it and running an arbitrary code on the host OS. Therefore, such malicious VMs can take complete control of the host OS. Escaping the guest OS allows the VMs to interact with the hypervisor and provides them access to other guest OS on the system as well. Figure 2.2 shows that the attacker from his virtual machine (VM 2) is able to escape his VM. VM 2 is used to compromise the hypervisor which is further used to launch attacks on other VMs (VM 1) in the system.

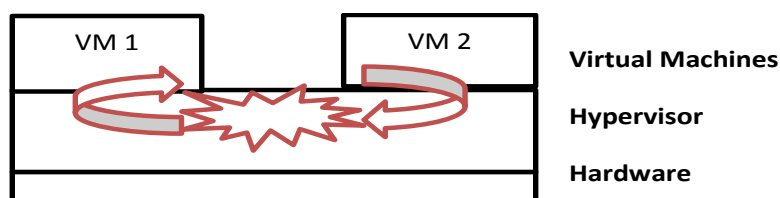


Figure 2.2: VM Escape attack

Hardware devices such as hard disk, network and graphic cards have the ability to access main memory contents without going through the CPU. This feature is provided to facilitate the CPU so that peripherals can directly transfer data to and from memory without the involvement of the CPU. Without Direct Memory Access CPU will remain fully occupied in any read/write operation. Allowing the device drivers to write at an arbitrary location in physical address space can result in many security threats. In virtualized environment if any device driver is malicious than it can write to Hypervisor and any other address space in order to compromise the system.

2.3.3 Virtual Machine Attacks

Malicious programs in different virtual machines can achieve required access permissions to log keystrokes and screen updates across virtual terminals [23] that can be exploited by attackers to gain sensitive information. If isolation is not properly implemented covert channels can be used for unauthorized communication with other VMs in the system. General attacks on OS of physical systems can also be targeted on guest OS of VMs to compromise them. Java applets can be installed through internet on virtual machines OS, which can install Trojans and malwares on VMs [24]. Attackers can use Trojans and malwares for traffic monitoring, stealing critical data, and tampering the functionality of VMs. Other security attacks from OS are possible through buggy software, viruses and worms that attacker can exploit to take control of VMs. Conficker, Zeus botnet, command and control botnet communication activity are the examples of such attacks that result in data destruction, information gathering and creation of backdoors for attackers. Attacks through buggy software, viruses and worms can exploit the guest OS in VMs. Furthermore, unpatched VM operating systems can be exploited by zero day attacks.

Network DOS attacks can be launched on a Cloud network and one such attack is the TCP SYN attack. TCP performs a three way handshake to establish a connection. The resources allocated for these three way handshake can be exploited to launch a TCP SYN flood attack. Attacker sends too many connection requests in form of TCP SYN packers to the victim so that the victim cannot respond to the legal requests. After allocating resources for the incoming requests victim replies to attacker with a SYN-ACK packet. The attacker doesnt respond to the SYN-ACK and the resources of the victim remain occupied [25].

Attacker can lease a VM in Cloud and through malicious software he can consume extra resources from the system. This may lead to Denial of Service (DOS) attack as the system resources will be unavailable to legitimate users. The privileged host virtual machine Dom0 can be compromised by attacker to either tamper boot process of guest VMs or access all guest VMs including their memory, disk space and network traffic. By controlling Dom0 attacker can create too many virtual machines to consume all resources of the system or destroy any virtual machine containing important data by launching DOS attack at Cloud. Furthermore, the saved state of guest virtual machine as a disk file appears in plaintext to Dom0. Attacker can compromise the integrity and confidentiality of the saved VM state and when restored VM may not function as desired.

2.3.4 Disk Image Attacks

Unnecessary guest OS images in Cloud can result in different security issues if the security of each image is not maintained [8]. If a malicious guest OS image is migrated to another host, it can compromise the other system as well. Furthermore, creating too many images and keeping unnecessary images can consume resources of the system which can be used as a potential attack vector by attacker to compromise the system [11]. When VMs are moved from one physical machine to other, data of VM images might still exist on previous storage disks that attacker can access. Similarly, attackers might also recover some data from old broken disks [8]. The security of image backup is also an issue. By gaining access to the backup images attacker can extract all information and data.

To create new VM image files existing VM images can be easily copied which results in VM image sprawl problem, in which a large number of VMs created by customers may be left unnoticed. VM image sprawl results in large management issues of VMs including the security patches. Investigation of VM images on Cloud (EC2, VCL) has shown that if patches are not applied VM images are more vulnerable to attacks, and they may also not fulfill organizations security policy. Secondly, some VM images are mostly offline and to patch these images they will have to be started. This will add to the computation cost of Cloud provider [26].

Attacker can access VM checkpoint present in the disk that contain VM physical memory contents and can expose sensitive information of VM state. A new checkpoint can be created by attacker and loaded in system to take VM to any state desired by attacker. If all the checkpoints in storage are accessed, information about previous VM states can be obtained [19].

2.4 Security solutions for Virtualization

To cater the attacks on virtualization environment different security solutions have been proposed in literature. This section discusses those security solutions for each component of virtualization architecture. By implementing these security solutions the attacks discussed in section 3 can be mitigated or at least the impact of those attacks on virtualization environment can be minimized.

2.4.1 Service Provider Security

Unauthorized person should not have physical access to the virtualization hardware of the system. In order to protect VMs from unauthorized access

by Cloud administrators, each VM can be assigned access control that can only be set through Hypervisor. The three core principles of access control namely identification, authentication and authorization will restrict admin access from unauthorized data and system components [27]. Moreover, if any administrator is involved in security compromise, access control implemented in Cloud can help identify that person. Web application attacks can be prevented by installing an application layer firewall in front of web facing applications and by having the customer code reviewed for common vulnerabilities [12].

An online identity management community OpenID has been integrated with an open source Cloud platform OpenStack to provide identity management in Cloud [28]. Sandra R. et al. [29] proposed an architecture using SELinux, XEN, IPsec as tools to enforce Mandatory Access Control (MAC) policies at VM, OS and network layers. These MAC policies control the communication between VMs based on application templates that can be configured by administrators dynamically. Furthermore, the security requirements of virtualized environment differ from that of physical system, Cloud service provider must make sure that the security tools for vulnerability assessment also include the virtualization tools used [8].

2.4.2 Hypervisor Security

Hypersafe is a system that maintains code integrity of the Hypervisor. It extends the hypervisor implementation and prevents its code modification by locking down the write-protected memory pages. It secures the Hypervisor against the control hijacking attacks by protecting its code from unauthorized access [30]. Also, in a system there are dedicated portions of memory to which only hardware has access. These dedicated portions of memory (that Hypervisor cant access) can be used to store information related to memory regions of VMs. So a compromised hypervisor cannot write at the memory locations of VMs.

Device drivers can be restricted from writing into memory resources by using input/output memory management units (IOMMU). It is used to connect a DMA capable memory bus to I/O units. When the machine starts, hypervisor checks if IOMMU is enabled and correctly initialized. If not, than hypervisor must stop the boot process for checking. Hypervisor may utilize DMA remapping hardware to limit DMA from input/output device to physical memory of Dom0 [16].

VM Escape attack can only be executed through a local physical environment. Therefore, the physical Cloud environment must be prevented from insider attacks. The interaction between guest machines and host OS must

also be properly configured [15]. In order to stop one VM from affecting or communicating with other VMs isolation must be properly implemented and maintained by hypervisor. Moreover, further possible attack vectors on hypervisors can be reduced by hardening the hypervisor [8]. These techniques include separating the duties of administrative functions, restricting the hypervisors administrator access to modify, create or delete hypervisor logs, and monitoring the hypervisor logs regularly.

2.4.3 Virtual Machine Security

Administrator must deploy a software or application that stops VMs from using extra resources unless authorized [31]. Moreover, a light weight process must run on a virtual machine that collects logs from the VMs and monitors them in real time to prevent any tampering of VMs. The guest OS and applications running on it must be hardened by using best security practices. These practices include installing security software such as anti-viruses, anti-spyware, firewall, Host Intrusion Prevention System (HIPS), web application protection, and log monitoring in guest OS as shown in figure 2.3. All the security measures for OS in physical machines must also be applied on the OS of VMs (guest OS). To prevent OS from java applet attacks, java installation must be avoided until it is essentially required by any program. Furthermore, guest OS and applications running on it must be hardened by using best security practices [12]. These practices include installing security software such as anti-viruses, anti-spyware to detect any suspicious activity and to notify the user or admin about such activity.

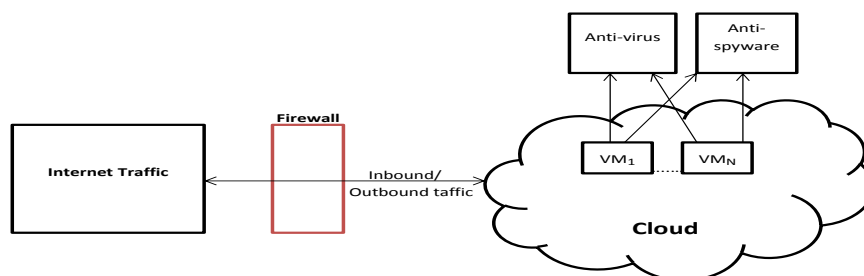


Figure 2.3: VM security by firewall, anti-virus and anti-spyware

To identify the faults in guest OS Dan P. et al. [32] proposed a system called "Vigilant". It utilizes virtualization and machine learning methods to monitor VMs through hypervisor without putting any monitoring agent in VMs (out-of-band detection). Flavio L. et al. [33] proposed Advanced Cloud Protection System (ACPS) that monitors and protects the integrity of OS in guest VMs. The periodic monitoring of executable system files is

done to check the behavior of Cloud components. It uses virtual introspection techniques to deploy guest monitoring machine in system without being noticed by attacker on guest VM. Hence any suspicious activity on the guest OS can be blocked. TVDc is developed by IBM to address strong isolation, integrity and security in virtualized system [34].

sHype system, a secure hypervisor architecture for XEN, offers isolation of VM systems with flexible security of Mandatory Access Control (MAC) [35]. MAC systems are designed to control information flow and communication between VMs across multiple machines. To authorize the communication between VMs on same machine, sHype adds hooks to XEN authorization mechanism and uses Type Enforcement (TE) model. Bell-La Padula policy restricts information flow at access class level, but implementing this model to have network per access class is impractical. So to minimize cost some risk must be taken by defense communities. Accepted risk may be expressed by using Chinese wall policy model. Chinese Wall policy model allows selection of any policy in data access, but once the policy has been selected future choice gets limited according to the selection [36].

To provide protection from networked DOS attacks firewall proxies can be used so that attacker request is forwarded only after client side ACK is received. HOP count filtering achieves 90 percent detection of DOS attacks, by inspecting packets TTL field and dropping suspected spoofed packets. However such solutions can result in excessive overhead on infrastructure as I/O devices such as network interfaces are being used. KVM is one of the standard hypervisor-based virtualization systems. KVM supports VirtIO drivers which are standard drivers in virtualization. KVM generates an interrupt when a message has to be sent from host to VM or from VM to host. VirtIO drivers can be modified to bundle together a series of packets so that when several packets are received only a single interrupt is generated. A timer can be used to send all packets to VM received during a specific time [25].

To protect the newly created virtual machines for users (guest VMs) from compromised privileged virtual machine Dom0, a protocol is designed by Jinzhu Kong [18]. Hypervisor generates a pair of secret keys, Kernel and the initrd image are kept encrypted all the time with the secret key King. First the user attests the Cloud server through Trusted Platform Module (TPM), if attestation succeeds then user sends a boot request to the Dom0 which then boots the guest domain. The guest VM executes the wrapping code and requests Hypervisor to decrypt kernel and initrd images. Hypervisor encrypts this request with its private key and asks user for key to decrypt kernel so that a VM can be created. The user sends private key King encrypted under the public key of Hypervisor. Hypervisor decrypts the user message, and the private key King is used to de- crypt the kernel,

initrd images and to launch the guest virtual machine. In this way the newly created VM is secured from compromised Dom0. The complete workflow is shown in Figure 2.4. To avoid the VM storage attacks, before saving the

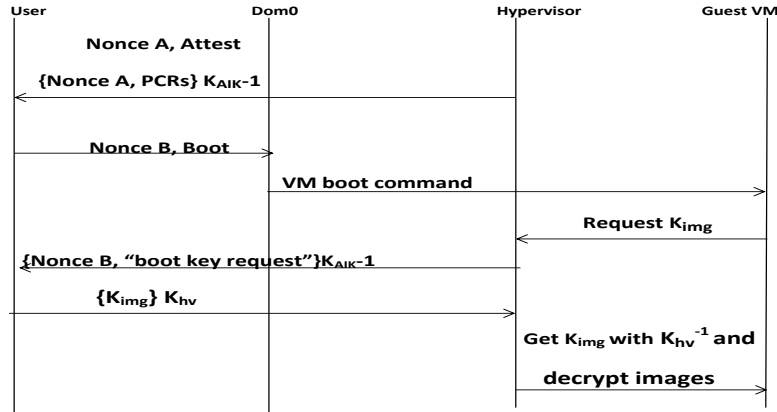


Figure 2.4: Secure VM boot protocol

state of the virtual machine in Dom0 its encryption can be done using AES-256, where key can be any random initialization vector. The hash of the encrypted state can be taken using MD5. When the virtual machines are to be restored, the new hash can be taken to verify the integrity of saved virtual machine. If the hash of the restored state and hash of the saved state match it means that the virtual machine state is not altered [18]. Figure 2.5 shows the secure storage of saved VM state.

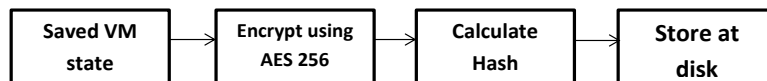


Figure 2.5: Securing the saved VM state

2.4.4 Disk Image Security

Organizations using virtualization must have a policy to manage the creation, usage, storage and deletion of images. Image files must be scanned for the detecting viruses, worms, spyware and rootkits that hide themselves from security software running in guest OS. J. Wei et al. [37] proposed an image management system to efficiently manage images in Cloud and detect security violations in images. It proposes the use of filters, virus scanners and rootkit detectors to provide protection against potentially compromised images. Nuwa [38] is a tool designed to apply efficient patching to VM images in Cloud. By analyzing patches, Nuwa rewrites the patching scripts so that

they can be applied offline. As a result, the installation scripts for online patching can be applied to images when they are offline.

When VMs are to be migrated from one physical machine to another, Cloud admin must recheck and ensure that all data is removed from previous or broken disks. To protect the backup VM images cryptographic techniques such as encryption may be employed to encrypt all backup images. If any VM is deleted then its backup must also be removed from system. Furthermore, to protect VM images from storage attacks, Cloud provider must encrypt the complete VM images when not in use [8]. Checkpoint attacks can be prevented by encrypting the checkpoint files. Another mechanism to provide security to Checkpoints is SPARC. SPARC is a mechanism designed to deal with security and privacy issues resulting from VM checkpoint. SPARC enables users to select applications that they want to checkpoint so sensitive applications and processes can be checkpointed [19].

2.5 Recommendations for secure usage of Virtual Machines

For the security of virtual machines following practices need to be adopted by the administrators of Cloud organization.

2.5.1 Secure Network

Network security must be provided to virtual machines to protect them from the network layer attacks. Virtual machines might have some open ports other than ports that normally remain open, which may allow others to connect remotely to virtual machine and change its configuration. A firewall must be there to allow access to these open virtual machine layer service ports [9].

2.5.2 Disabling the Non-Required Features

Using the features of screensavers, search tools, system update may create some issues if they are run in virtual machines. The processor intensive applications may consume the resources of system when these resources are needed by some other virtual machine. Similarly, the operating system of the virtual machines runs the low priority processes at same time which can affect normal priority tasks. So the low priority tasks should be disabled and if any task has to be run its start time must be staggered [9].

2.5.3 Disconnect Unused Hardware Devices

Virtual machines can control physical devices out on host such as CD drives. If any virtual machine has access to physical drive, than all virtual machines requesting access to that particular drive will remain blocked. Host must be configured to access these devices only when required [9].

2.5.4 Backup of Virtual Machine Images

Backup of the virtual machines images must be maintained. However, the security of backup data stream and virtual machine backup image on disk is an issue. Cryptographic techniques such as encryption may be employed to protect the data stream. Similarly, data in transit may be protected by securing network through techniques such as VLANs [12]. If any VM is deleted than its backup must also be removed from system.

2.5.5 Hardening of Virtual Machines

Virtual machines must be deployed in Cloud by following the industry standards. Some recommendations for hardening virtual machines are, to put limit on virtual machine resource usage, ensure OS of each virtual machine is hardened, harden the Hypervisor, and harden each of VM virtual hardware [15].

2.5.6 Auditing

Administrators can find offline VM guests by using a logging server to monitor logs. VM power status, change of hardware configuration and changes to virtual machines on host should be audited [12]. Some other recommendations to secure VMs on Cloud are that in VMs mixed mode deployment may occur, as a result VMs that have different classes of data may lie on same physical machine. Moreover, inter VM interactions must be monitored carefully to look for malicious behavior of VMs in case of any attack.

2.6 Industrial Survey

In this section we discuss about the industries currently working in the area of Cloud computing and its security. These products are mostly based on open source Cloud platforms such as OpenStack and provide additional features such as security and control.

2.6.1 Storage Made Easy

It is a commercial solution that provides many different security features in Cloud environment. It is designed to unify the information stores securely. Storage Made Easy (SME) can be used with different Cloud networks such as OpenStack, Amazon S3, Azure, RackSpace, HP Storage and other Clouds. It is available to be used as a SaaS application as well as a IaaS application. The major features of SME include secure image storage in Cloud through strong encryption of AES-256, Cloud data protection gateway, secure authorization and storage of company data, versioning of all files, file sharing and collaboration with other Clouds [39].

Storage Made Easy APIs are also available for developers. These API are based on REST interface and some libraries are written in .NET. SME can be integrated with the security standards such as OpenID and Kerberos. Other identity features that can be integrated with SME are LDAP/Active Directory. The user integration using SME with the OAuth server is provided in SME documentation.

2.6.2 Piston Cloud

Pistons Enterprise Operating System offers secure Cloud services to the enterprises. It is powered by OpenStack and contains all its features with the added security components. It makes use of the private infrastructure while maintaining security and control over infrastructure. Its major benefits are the ease of use as complete private Cloud can be built in less than an hour. For installation a USB drive is used that contains installation software to fully configure servers with secure Cloud stack. Pistons Enterprise Operating System is very efficient for handling big data as data is present closer to virtual machines. Scaling the Cloud is easy as each new server is automatically detected in the Null Tier Architecture. To protect the sensitive data from single point of failures data is usually replicated [40].

Pistons Enterprise Operating System runs on security hardened Linux Kernel. This Linux kernel is based on HLFS (Hardened Linux from Scratch). This distribution follows the basic security principle of having minimum packages required to run OpenStack. Updates are usually automatic to prevent Cloud from zero day attacks. Along with security patches, updates also involve installation of new packages issued by OpenStack. To provide access security, Piston Cloud offers Role-Based Access Control. Moreover, secure booting is also ensured to protect system from possible attacks. Additional features offered by Piston Cloud include an availability framework to reduce downtime and virtual machines access through dashboard.

2.6.3 Metacloud

Metacloud delivers public Cloud services based on OpenStack with the security and privacy features. It provides a fully functional, highly available and easy to use Cloud. It provides a complete support to monitor, troubleshoot, upgrade, fix bugs and provide other services. Users and admins access Metacloud through dashboard. Dashboard also provides information about Cloud visibility. Metacloud OpenStack is fully compatible with OpenStack APIs and OpenStack CLI [41].

Table 2.1: Summary of security aspects of virtualization

Category	Requirements	Attacks	Solutions
Service Provider	Limit access to hardware	Malicious code execution	Develop and implement policy to limit access to hardware
	Implement access control	Stealing of customer data through unapproved access	Implement MAC policies at VM, OS and network layers
	Provide strong authentication mechanisms to users	Unauthorized access to Cloud system and data	OpenID integration with OpenStack Cloud to provide secure authentication
Hypervisor	Maintain isolation between VMs	VM Escape attack	Properly configure the interaction between guest machines and host VM
	Hypervisor should monitor functionality of guest VMs	Customers can lease a guest VM to install a malicious guest OS	Encrypt the VMs to protect them from compromised hypervisor and VMs
	Programs controlling the hypervisor must be secured using best software security practices	Malicious hypervisors attacks including BLUEPILL, Vitriol and SubVir	Hypersafe is a system designed to maintain the integrity of Hypervisor Use techniques to harden the hypervisor security
Virtual Machines	There must be limit on VMs resource usage	Using a malicious VM to consume extra resources of the system, resulting in DOS attack	Administrator must deploy a software or application that limits VMs from using extra resources unless authorized
	Isolation between virtual machines should be implemented properly	Malicious programs use covert channels to communicate with other VMs in unauthorized way	Vigilant can monitor faults in guest OS of VM
	Update the OS regularly and use anti-virus software, secure internet and restrict remote access	Malicious programs can monitor traffic, steal critical data, and tampering the functionality of VMs	Security features such as firewall, HIPS, log monitoring must be provided in guest OS
	Guest OS must be monitored regularly for updates and errors	Attacks through worms, viruses, botnets can also be used to exploit the VMs	Use anti-viruses, anti-spyware programs in guest OS to detect any suspicious activity Advanced Cloud Protection System (ACPS) can monitor and protect the integrity of guest OS
	Securely boot the guest VMs	Attacker can tamper boot process of guest VMs	Security protocol by J. Kong can be to ensure secure boot of guest VMs
	Saved VM state must not be tampered by Dom0	Attacker can compromise the integrity and confidentiality of the saved state of guest virtual machine	Use encryption and hashing of VMs state before saving VM
Guest Images	Snapshot access must be prevented from authorized access	VM checkpoint attacks	Checkpoint attacks can be prevented by encrypting the checkpoints or using SPARC
	Make a policy to remove unnecessary images	Security issues from unnecessary images can compromise system	J. Wei et al. proposed an image management system to manage images in Cloud
	Apply updates and patches to maintain images secure	Old images are vulnerable to zero day attacks	Nuwa is a tool designed to apply efficient patching to VM images in Cloud
	There must be policy to remove images from old disks after VM migration	Attackers can access and recover data from old and broken disks	After VM migration, Cloud admin must ensure that data is removed from old disks
	Backup of the virtual machines images must be maintained	Unauthorized access to the backup data can result in leakage of sensitive information	Backup of VM images must be encrypted. If any VM is removed then its backup must also be removed

Chapter 3

Research Methodology

In this chapter, we discuss the research methodology adopted during our thesis to achieve the goals. Our thesis had two major components. First one was the detailed security analysis of the virtualization components to identify possible attack vectors and other was the design and implementation of framework for the protection of virtual machine images in Cloud computing. For the completion of our research goals we adopted different research methodologies during our research. These methodologies include deductive approach and conceptual research methods. We define the complete methodology for studying theory, developing hypothesis, performing the observations and the development of prototype based on hypothesis.

3.1 Introduction

Research is the methodological and structured search of new knowledge in order to establish facts and solving existing problems. It involves finding the most efficient answers to worthwhile questions. Research can be carried out to find solutions of unsolved problems, develop theoretical knowledge on a matter using specific facts or to improve the existing solutions to problems. Research is done in different phases with certain objectives. Research objective is the major motivation behind the research work. During research, different ways to reach the goals can be considered and eventually the most efficient solution is adopted. Therefore, considering the effectiveness of solutions with respect to existing solutions is an important part of research.

3.2 Types of Research Methods

The different methods adopted during research are explained below.

3.2.1 Descriptive and Analytical Research

Descriptive research is done to describe the state of the art research related to a specific area. It does not involve any novel contribution rather it consist of various surveys of literature and questionnaires to find facts. Analytical research involves the usage of critical thinking to identify facts of a particular solution and develops ways to improve it. The whole process is done by the evaluation of data already available to researcher [42].

3.2.2 Applied and Fundamental Research

Applied research involves study of the problems faced by business, industry, state or clients and developing a solution for them. It aims at real world problems to develop new products and technologies. While fundamental or pure research involves the systematic study carried out to understand the core concepts of a phenomena [43]. It involves observation of facts to develop theory without the idea of future practical applications. Fundamental research adds to the broad base of organized knowledge that can be used for applied research.

3.2.3 Quantitative and Qualitative Research

As the name of quantitative research suggest, it relates to measuring the quantity or number of data for mathematical analysis. Generally it is used to develop theories for long term research. It generally works by finding the relationship between observations of quantity and its connection with mathematical expressions. Large amount of data collected by quantitative research can also be used to make conclusions for research. Qualitative research refers to measuring the quality to understand the motivation and reasons behind phenomenon. To collect data for quantitative research, generally unstructured methods are used. It is usually done to identify the initial knowledge about any area of carry out further research in it. Combination of quantitative and qualitative research is used by researchers and companies to get better understanding of results.

3.2.4 Conceptual and Empirical Research

Conceptual research is related to find the basic or generalized ideas behind a phenomenon. It focuses only on theory to develop new concepts and research directions. The conceptual research starts with theory development and study on historical research on that phenomenon. It also involves doing

state of the art review of the literature on specific behaviors and performing the critical analysis on research. Critical analysis includes identifying, understanding and evaluating the research on any area to give ones own propositions. Empirical research is completely related to the practical work on the basis of observations and data-analysis. It can involve extensive experimentation in which a researcher must know or guess the probable results. The purpose of experimentation is to verify the research claims [43].

3.2.5 Deductive and Inductive Approaches

Deductive method is a way of logical reasoning in research that involves a top-down tier approach going from general to specific things. It includes theory, hypothesis, experimental observations and validation. Therefore, it is the more trustworthy research approach. Inductive research involves study of experimental observations to develop general theory. It is called the bottom-down approach. Researchers begin by studying specific observations and measures, than specific patterns from observations are formulated, hypothesis is developed and at the end more generic theories are made.

3.3 Thesis Research Methodology

Our research was conducted in two major phases. One was the analysis of Cloud virtualization to gain in-depth understanding of possible security issues related to virtualization. This led us to identify the general requirements, possible attacks and proposed security solutions for Cloud virtualization. The second objective was to design a protocol for the security of virtual machine images in Cloud. We designed our protocol by proposing the encrypted storage of virtual machine images in Cloud and implemented it. To achieve our research goals we had to adopt more than one research methodology described above. For instance, we used conceptual research to study the security architecture of Cloud virtualization. Similarly, we used empirical research to implement our solution for security in Cloud.

3.3.1 Research Objectives

The objectives with which the research is carried out are called research objectives. We had two major objectives. These objectives were:

- Analyzing the security of Cloud virtualization.

- Design, implement and validate a framework for the protection of virtual machine images in Cloud.

3.3.2 Research Methodologies

To achieve our research objectives, deductive research method was followed. To study and critically review the literature we followed conceptual research approach. We started by studying the theories related to IaaS security in Cloud, Cloud virtualization and virtual machines security. This literature review consisted of studying the various survey papers, conference papers, journal papers and technical papers. This research led us to the development of hypothesis for our research. A hypothesis is a tentative assumption of future research goals. The hypothesis is set as the basis of future research for a phenomenon that has to be tested and verified.

3.3.3 Hypothesis

The hypothesis of our research was,

- (a) Does the security solutions for Cloud virtualization satisfy all the requirements to prevent it from all possible attacks?
- (b) Are the current solutions for virtual machine images enough for their security? Does the virtual machines images life cycle need modifications or extensions?

3.3.4 Observations

After defining the hypothesis, next step was of observations. The major observations made related to the Cloud virtualization security to support the hypothesis were,

- (a) Virtualization is a basic feature of IaaS that enables a single system to concurrently runs multiple virtual machines. Hypervisor is a component that is largely responsible for managing virtual machines and their interaction. Guest images function as the hard disks of virtual machines. Each of these components is vulnerable to some attacks in Cloud and security solutions to prevent them from all attacks do not exist.
- (b) Virtual machine images contain all the operating system, applications and data in a virtual machine. During storage in Cloud they can be exploited by malicious nodes in Cloud, and external attacks that might lead to data leakage and image tampering.

- (c) NIST guidelines on virtualization technologies have proposed usage of cryptographic techniques to protect virtual machine images in Cloud.
- (d) CSA guideline mentions attacks on the stored VM images in Cloud and emphasizes the need of their security be encryption images all the time.

3.3.5 Implementation and Verification of Prototype

The next step was developing and verifying the prototype developed for our research, which is the last step of deductive approach. Our design focused on the complete life cycle and management of images in Cloud. After the termination of VM, the image has to be stored in Cloud. Our prototype includes the image encryption protocol that encrypts images before storage on Cloud. The choice of cryptographic algorithms is based on the NIST guidelines on storage technologies. When the VM is to be instantiated, our prototype involves decryption of image. The evaluation of the system was done by design a threat model to verify that all storage attacks on VM images are mitigated by our prototype. The performance evaluation also reveals a minor overhead caused due to usage of cryptographic techniques for the security of images. The detailed research methodology adopted during thesis is shown in figure 3.1.

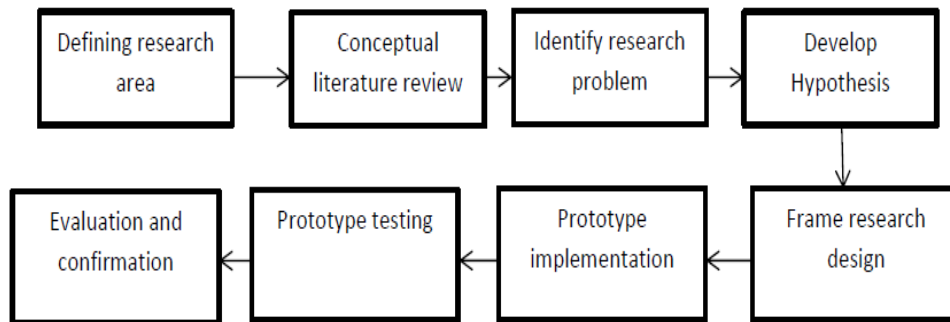


Figure 3.1: Thesis Research Methodology

Chapter 4

Design and Implementation

This chapter discusses the design and methodology followed during the thesis to achieve the solution. It first describes the details of different components of proposed model for image security called Encrypted Virtual Disk Images in Cloud (EVDIC) and then elaborates their functionalities. The architectural concatenation of EVDIC is described with OpenStack. It includes the complete workflow for image encryption, storage and decryption in the storage component of OpenStack (Swift).

4.1 Design of Framework

Encrypt Disk Images in Cloud (EVDIC) is the proposed framework for addressing the security issues of stored disk images in Cloud. EVDIC workflow is divided into two parts. One part is the image encryption through EVDIC, before an image is stored on the disk. Other is the image decryption through EVDIC, while retrieving a stored image from the disk to be used by a virtual machine. The major components involved during the image encryption and decryption are Image Encryption Module, Key Management Server and Image Decryption Module. They interact with the other components of Cloud. The major functionalities of these components are described below.

4.1.1 Image Encrypt Module (IEM)

The Image Encrypt module intercepts a request to store an image in the disk. It interacts with the key management server to obtain an encryption key for a user that it uses to encrypt the image. The encryption of images is done by using AES-192 or AES-256. After doing the encryption, IEM sends the encrypted image to the disk for storage.

4.1.2 Image Decrypt Module (IDM)

The major function of Image Decrypt module is to decrypt the images. It intercepts a request to retrieve a stored disk image from the disk and interacts with the key management server to obtain user key for decryption. After getting the key from KMS, IDM locates the stored image on disk using the metadata stored with image. Once the image is located, it is retrieved and decrypted by the IDM. IDM sends the decrypted image to the related component so that it can be used by the virtual machine.

4.1.3 Key Management Server (KMS)

KMS is responsible for management of keys used for encryption. Once the encryption keys are derived for users, they are stored in KMS. The unique identification of each user is maintained by a field called KeyID and it is used to retrieve and store encryption keys for each user when required. Due to security purposes, the KMS is placed at a separate location form the Cloud. All communication between KMS and EVDIC components takes place through Secure Socket Layer version 3.0 (SSLv3). SSL 3.0 provides the security during key exchange and ensures message confidentiality and integrity [44]. Keeping the KMS outside the Cloud will keep KMS secure even if the complete Cloud infrastructure is compromised. On the other hand, if keys are stored in the Cloud and Cloud infrastructure is compromised, attacker can access the encryption keys in KMS. These encryption keys could then be used to decrypt the encrypted images and access critical data.

4.1.4 Image Encryption through EVDIC

The workflow for image encryption through EVDIC is described below.

- When a running virtual machine is terminated by user, the virtual disk images have to be stored somewhere in the Cloud.
- The Image Encrypt module intercepts the request to store image in disk.
- To do the encryption, IEM interacts with the key management server (KMS) using the key identification (KeyID) value of the user. KeyID is a unique value to identify each user.
- The encryption key is retrieved from the KMS and send back to the IEM using SSL.

- Using the key of user, IEM encrypts the disk image through a secure encryption algorithm such as AES-192 or AES-256.
- The encrypted image is stored on the disk. KeyID of the user will be stored as metadata along with the stored image that will be used to retrieve the encrypted image when required. The design of EVDIC encryption scheme is shown in figure 4.1.

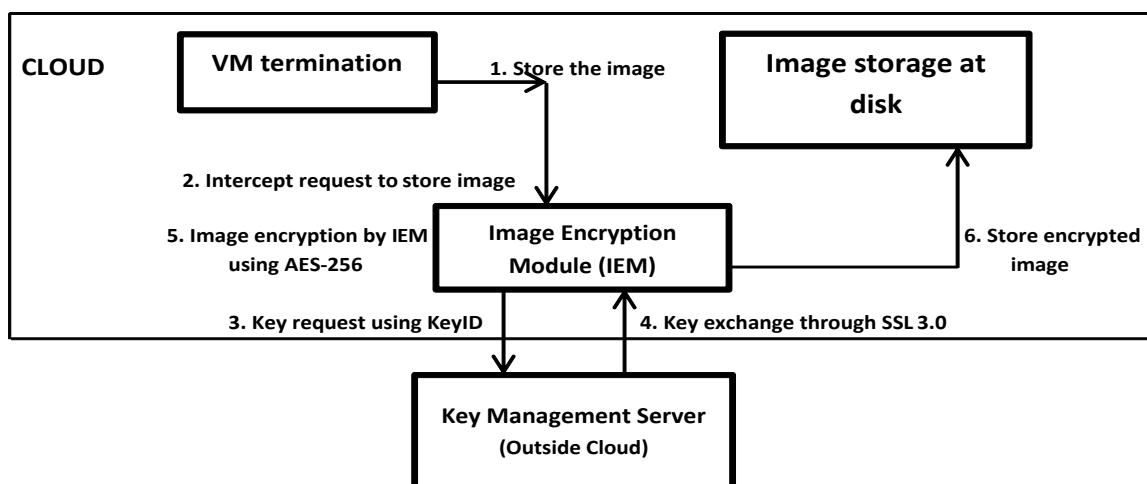


Figure 4.1: Disk image encryption through EVDIC

4.1.5 Image Decryption through EVDIC

The workflow for image encryption through EVDIC consists of following steps.

- All the stored disk images are in encrypted form. Whenever a virtual machine is to be launched in Cloud, it requests for the virtual disk image.
- IDM intercepts the request to retrieve the disk image.
- IDM identifies the KeyID of the user and communicates with the KMS for key exchange.
- The encryption key is retrieved from the KMS and send back to the IDM using SSL.
- After getting the encryption key, IDM locates the encrypted disk image in disk with the help of KeyID stored as metadata with the image.

- The located image is then decrypted by the IDM.
- The decrypted image is returned to disk so it can be used by the virtual machine. The design of image decryption scheme is shown in figure 4.2.

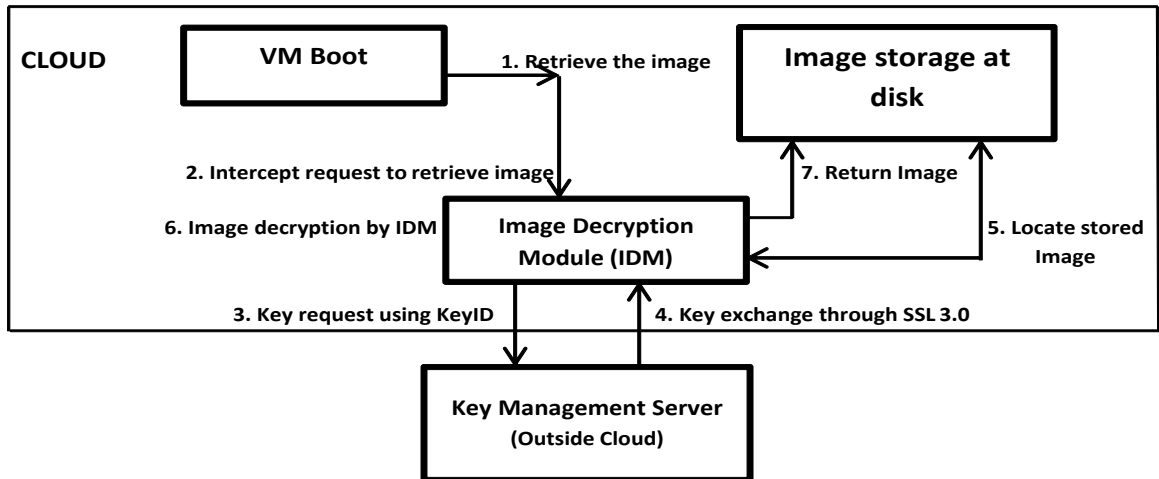


Figure 4.2: Disk image decryption through EVDIC

4.2 EVDIC in OpenStack

OpenStack provides a large scale open source Cloud computing platform [45]. The major components of OpenStack are Compute, Image Service, Identity, Dashboard, Network, Object Storage and Block Storage [46]. The ComputeWorker (nova-compute) is the component that manages virtual machines life cycle through hypervisor API's (XenAPI for XenServer, libvirt for QEMU or KVM, VMwareAPI for VMware). When a virtual machine is to be launched in OpenStack, nova-compute interacts with the Image Service (Glance) to request a VM image through Image Application Programming Interface (Glance API).

Glance deals with the retrieval, registering and storage of virtual machine images. Glance API stores the images in Object Storage, Swift (if specified by the user). Swift is a highly scalable object storage, that can store a large amount of data through a RESTful HTTP API. Each image stored in Swift has an image ID and a URL associated with it that is used to locate and upload image from storage. Furthermore, all objects in Swift have their own metadata [47].

4.2.1 EVDIC Image Encryption in OpenStack

As the user terminates its virtual machine, Glance API stores the image. Glance interacts with Swift to store the image on it. Swift first determines the partition to store image using account, container and object names. EVDIC can be integrated with OpenStack such that after Swift Proxy server sends a request to the Object Server to upload image through RESTful HTTP API, control is transferred to the EVDIC Image Encryption Module (IEM). IEM performs the image encryption in Object server, as all the write requests pass through it. All the swift servers write data in chunks, therefore each chunk must pass through the IEM before storage.

EVDIC uses the same procedure for encryption as mentioned above in EVDIC image encryption scheme. The encryption key for each user is generated through PBKDF-2 and retrieved from KMS using SSL 3.0. This encryption key is used to encrypt each chunk of image through AES-256. The encrypted chunks are written in a temporary file by Swift, before storing them in the partition on disk. Hash of the images is calculated using MD5 before their encryption and stored along with metadata. The Glance API stores the location of stored image in glance-registry including the image ID and URL of image. Disk image encryption through EVDIC in OpenStack is shown in Figure 4.3.

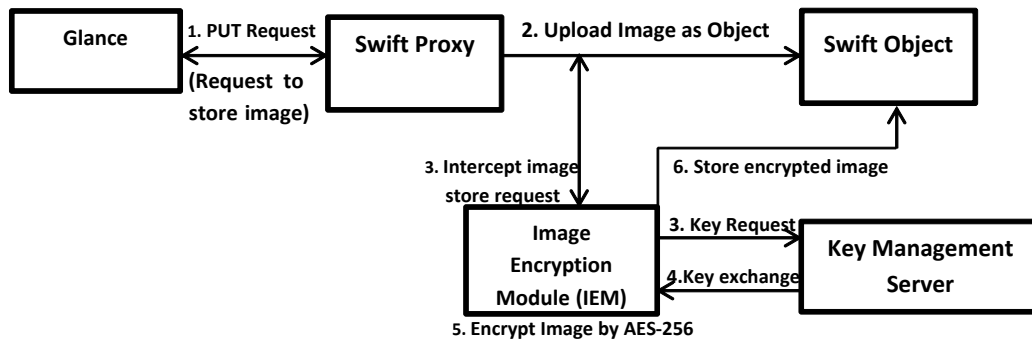


Figure 4.3: Image encryption through EVDIC in OpenStack

4.2.2 EVDIC Image Decryption in OpenStack

When the virtual machine of a user is to be launched, nova- compute passes request for accessing disk image to Glance using Image API. Glance sends a message to Swift to download the desired image. To locate the image the partition name is produced by Swift in a similar way to one generated during object upload. During the image download from Object server, image is

passed through the image decryption module. IDM uses KeyID in metadata of stored image to retrieve the encryption key from KMS using SSL 3.0. IDM decrypts the image and hash of the decrypted image is taken by MD5 to compare it with the hash stored in metadata. In this way the integrity of image is ensured. IDM sends the decrypted image to Proxy server, and Proxy server returns image to Glance. Glance returns the image to nova-compute using Glance API. The decrypted image can be used by the virtual machine. The OpenStack image decryption through EVDIC is shown in the figure 4.4.

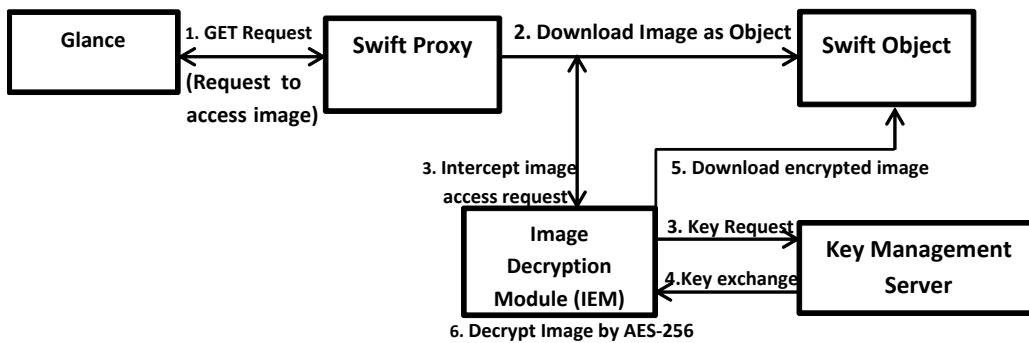


Figure 4.4: Image decryption through EVDIC in OpenStack

4.3 Implementation

Our framework has been implemented in Swift. Swift is the storage component of OpenStack. It presents a scalable system without as single point of failure, capable of storing large amounts of unstructured data. It uses RESTful HTTP API. Swift usage can vary according to the needs. For example, it can be just used to store virtual machine images or small objects as well as a storage service for Private Clouds. Swift has large development community and is designed to operate in failures without any downtime. Client libraries for development in Python are available in all programming languages including Python, Java, Ruby and C++ [47].

Objects are stored as clusters in Swift. To extend the storage capacity of cluster, new nodes can be added without any downtime. As all the communication between Swift components happens using HTTP and all Swift objects have a URL. A typical Swift URL is composed of base address, account, object and container. The seven major components of Swift are, i) Proxy Servers ii) Accounts iii) Containers iv) Objects v) Rings vi) Zones vii) Partitions.

Proxy servers are responsible for handling all the incoming HTTP requests. After determining the storage node of object, they also handle responses from other components. Rings are responsible for mapping the physical location on disks. To locate the objects in Swift cluster, all the components need to interact with Rings. Rings contain partitions and replicas. The collection of all stored data is called a partition. Both the accounts and containers are SQLite Databases. Accounts contain the lists of containers; containers have the list of objects using the database referencing lists. Figure shows the mapping followed by accounts and containers. Object is the actual data to be stored such as an image.

4.3.1 Object Upload in Swift

Swift uses RESTful HTTP API, therefore the major methods for managing object are PUT, GET, POST and DELETE. HTTP PUT method is responsible for updating the resource on the server. To upload an object to Swift server, client sends a PUT request to proxy server which connects with Keystone (authorization component of OpenStack) to determine if the client is authorized using clients token. After client is verified, cluster receives the upload request and all the accounts, containers and objects servers are used to determine the partition to store object. The object is forwarded to storage node where the required partition lies. After the object write is successful, HTTP 200 message is sent to the client. Object upload also requires the containers database to be updated. This update reveals that a new object is present in the container.

4.3.2 Object Download in Swift

To download an object, HTTP uses its GET method. This method is responsible for retrieving an object from server. Once a client send a GET request to proxy server, the user authorization is verified by the Keystone component. After user verification, partition for the object is determined using the accounts, containers and objects server. Ring is searched to find the required partition and the object is retrieved from that storage node. If the object retrieval request fails through one storage node, requests are made to the other nodes.

4.3.3 Swift Source Code

Swift is maintained and developed by one of the largest open-source teams in the world, and is in the top 2 percent of all project teams on Ohloh. It

has 53,605 lines of code and is written in Python. The source code of Swift is available at the online GitHub repositories [48]. These repositories can be modified to download and replace custom code instead of the default ones. To keep track of source code modifications they need to be added in version control. This custom repository can be shared with anyone who can then download its working copy. This working copy of code can be used with OpenStack deployment instead of the default one.

4.3.4 Object Encryption in Swift

The code for object server of Swift is located `server.py` file in `obj` directory. There are only two classes in `server.py`. `DiskFile` class is for file Input and Output. The `ObjectController` class handles all the requests made by the user to the Swift object storage.

The major class is the `ObjectController` class that has the methods of RESTful HTTP API including PUT, POST and GET. To upload a file to the object storage, a put request is issued and handled by PUT function of the `ObjectController` class. The PUT contains the code for writing all the object files to disk. Swift writes objects as chunks to disk instead of a single file. Each object file is read into a number of small chunks and stored in a temporary file which acts as a buffer. Contents from the temporary file are then written to a permanent file when buffer size reaches a certain threshold value (default is 512 MB). It is the responsibility of PUT method to read all chunks of a single object and write them to disk.

Each of the chunks is written to a buffer, it is better to encrypt them before they are written to it. In this way, contents are copied from the buffer to the actual file will be encrypted such that when all chunks are written to the actual file, the file will be encrypted. To handle encryption, we have created a separate `Encryptor.py` file that handles the encryption and returns encrypted chunks in the object server. Encryption classes use the popular M2Crypto package of Python to perform encryption [49]. M2Crypto is present as a as a fully supported package in the Ubuntu 10.04 and later repositories. Writing the encrypted chunks in Swift is similar to writing the normal chunks. The current implementation uses a constant key that is saved in a configuration file.

4.3.5 Object Decryption in Swift

To download a file from the object storage, a RESTful HTTP method of GET is used. GET is implemented inside the `ObjectController` class of `server.py` (object server). When a request is made to retrieve a file, the object server

returns an iterator to retrieve that file. The file location is determined using the hashing of accounts, containers and objects. The GET method downloads the encrypted file chunk by chunk and decrypts them using the M2Crypto Library. The decrypted object (file) is returned to the proxy server of Swift.

Chapter 5

Deployment and Validation through OpenStack

This chapter discusses details about the deployment of OpenStack for development. DevStack is composed of shell scripts to make complete development environment of OpenStack. This chapter also describes the use of DevStack for debugging the source code of OpenStack. For debugging the source code the Python debugger (pdb) is used. The results of adding encryption to OpenStack are shown along the techniques to verify the encryption of disk images from the OpenStack dashboard.

5.1 Deployment of OpenStack

In order to perform development in OpenStack, a developers version of OpenStack exists that is known as DevStack. DevStack is easy to deploy and provides a development platform for OpenStack. DevStack consists of documented shell scripts to configure complete OpenStack development environment [50]. The recommended way to install DevStack is on Linux running on a virtual machine. Typical configuration used for installing Devstack is Ubuntu 12.04 LTS as host OS For deploying DevStack on your virtual machine, git has to be installed inside the VM. After installing fit, DevStack repo can be cloned by using the following command, "git clone https://github.com/openstack-dev/devstack.git". After browsing to devstack directory, create a localrc file in devstack directory. The localrc file in DevStack contains some of the configuration primitives which can be modified as-per-need basis. Now OpenStack can be started. In devstack directory, execute the stack.sh script and after the execution is complete, output will be similar to one shown in Figure 5.1. Figure shows that the working deploy-

```
Keystone is serving at http://172.16.25.136:5000/v2.0/  
Examples on using novaclient command line is in exercise.sh  
The default users are: admin and demo  
The password: malik  
This is your host ip: 172.16.25.136  
stack.sh completed in 750 seconds.  
kazim@ubuntu:/opt/stack/devstack$ █
```

Figure 5.1: Output of the running DevStack

ment of OpenStack is configured to run only Keystone and Swift. Keystone is an OpenStack project that provides Identity, Token, Catalog and Policy services for use projects in OpenStack. The Keystone Identity Service allows clients to obtain tokens that can be used to access OpenStack cloud services [51]. As OpenStack GUI (Horizon) it is not enabled, it cant be accessed.

5.2 Adding objects to OpenStack Swift

To add objects to Swift, go into the DevStack directory. The absolute path of DevStack Swift is `/opt/stack/devstack`. The command `source openrc` authorizes users for using OpenStack Swift and generates a token for authorization. Without this token Swift cant be accessed. Before adding objects in Swift, containers need to be added. Status of the new container can be seen by giving the command `Swift stat`. Containers are essentially databases that are used to store information about objects. These are directories where files (objects) can be stored. Figure 5.2 shows the status of new container created.

The command `swift stat` will now show a single container. The newly created container can also be viewed by opening the root directory and navigating to `/opt/stack/data/swift/drives/sdbX/Y/sdbZ/containers/`. Inside the nested container directory, a file named as `xxx.db` will be present. This is the database that holds the information about objects stored by Swift.

5.3 Debugging Swift

All of the code for OpenStack components is located in their respective directories. Repository locations are stored in `stackrc` file in the `devstack` directory. The default location of the `swift` repo can be replaced with user modified version of Swift. To do that navigate to Swift on github and fork the repo. After that, copy the forked repo's clone URL.

```
kazim@ubuntu:/opt/stack/devstack$ swift stat TestContainer
  Account: AUTH_71900f62ab4441cab52b8d550a26b0f5
Container: TestContainer
  Objects: 0
   Bytes: 0
  Read ACL:
Write ACL:
  Sync To:
  Sync Key:
Accept-Ranges: bytes
X-Timestamp: 1384843605.14920
X-Trans-Id: tx2eac3c10376048498e821-00528b095f
Content-Type: text/plain; charset=utf-8
kazim@ubuntu:/opt/stack/devstack$
```

Figure 5.2: Status of Container named TestContainer

The debugging OpenStack Swift code can be started at this stage. For debugging OpenStack code, the Python Debugger (pdb) is used. Major features of pdb include source code listing, line stepping at source code level, setting breakpoints and providing quick response. Pdb can be used to execute the statement, evaluate an expression, call a function and enter post-mortem debugging all under the debugger control. To add pdb debugger to Swift, navigate to the obj directory, import pdb and use its debugging options.

This sets a breakpoint in the PUT function of the ObjectController class and will hit each time a command is given to upload a file to the server. Ensure that two terminals are running side by side. Enter screen -r in one of the terminal (call it Terminal2). Hit Ctrl + A + X where X corresponds to the number against s-object. Now on the other terminal (Terminal1), Swift upload commands can be issued for any container.

A breakpoint will be hit on Terminal2 where code can be debugged using normal pdb commands. In case if something goes wrong in Terminal2, Ctrl + C can be pressed. This will restart the object server and debugging can be started again.

5.4 Code Validation through OpenStack

Complete OpenStack deployment through DevStack can be used accessed through dashboard. To verify the implementation we will add new images to verify their encrypted storage. We start by accessing OpenStack to add new images and launch VMs. The URL on which Horizon is running after the successful execution of DevStack can be entered to browser. Login using the

credentials (username and password) obtained from Keystone after successful execution of DevStack as shown in figure 5.3.

```
Horizon is now available at http://172.16.25.136/
Keystone is serving at http://172.16.25.136:5000/v2.0/
Examples on using novaclient command line is in exercise.sh
The default users are: admin and demo
The password: malik
This is your host ip: 172.16.25.136
stack.sh completed in 750 seconds.
kazim@ubuntu:/opt/stack/devstack$
```

Figure 5.3: Horizon URL and User Credentials

After the login, new and custom images can be added to OpenStack. The custom bootable image for OpenStack named cirros can be downloaded from launchpad.net. For test case, download a Qcow2 bootable image named cirros-0.3.0-i386-disk.img [52].

After downloading the image, go to Images tab on left side of Dashboard. Click on create image, you can browse to the qcow image and select it to upload. The image and format can be selected. All properties during creation of custom image named Newimage are shown in figure 5.4.

This image is present in encrypted state on the disk. The location of image can be seen in the directory /opt/stack/data/swift/drives/sdbX/Y/sdbZ/Objects/. To verify that the image is encrypted, view the size of image uploaded and the original image that was downloaded. Moreover, Md5sum can be used to take hashes of the two files and compare them.

To launch an instance using the encrypted image, click on Instance tab on left and then on launch instance. Select the custom image named Newimage to be used by the instance. The instance created using the Newimage is shown in figure 5.6.

The image on disk will be automatically decrypted when you click on launch. The instance can be used by the user and after the instance termination; the image will be saved in encrypted form on disk. Since the image is present in encrypted state on disk, the data on image cannot be read and accessed unless it is decrypted. Hence, the confidentiality of image in Cloud storage is preserved.

Create An Image

Name
Newimage

Description
Additional information here...

Image Source
Image File

Image File
/home/kazim/Desktop/cirros-0.3.1 [Browse...](#)

Format
VDI

Minimum Disk (GB)
1

Minimum Ram (MB)
64

Description:
Specify an image to upload to the Image Service.
Currently only images available via an HTTP URL are supported. The image location must be accessible to the Image Service. Compressed image binaries are supported (.zip and .tar.gz.)
Please note: The Image Location field MUST be a valid and direct URL to the image binary. URLs that redirect or serve error pages will result in unusable images.

Figure 5.4: Image creation process in OpenStack

OpenStack Dashboard - Mozilla Firefox

Images - OpenStack Dashbo... Google

172.16.25.136/admin/images/images/

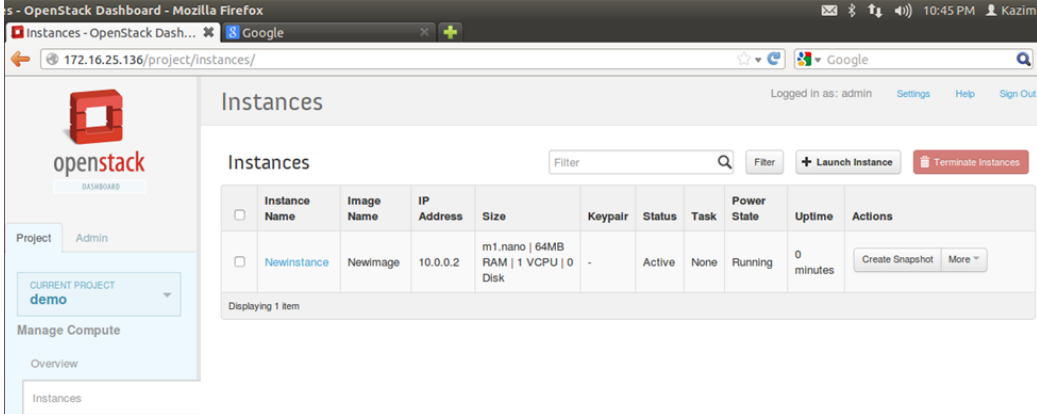
Logged in as: admin Settings Help Sign Out

Images [+ Create Image](#) [Delete Images](#)

<input type="checkbox"/>	Image Name	Type	Status	Public	Protected	Format	Actions
<input type="checkbox"/>	Newimage	Image	Active	Yes	No	RAW	Edit More

Displaying 1 item

Figure 5.5: Successful creation of image on OpenStack



The screenshot displays the OpenStack Dashboard interface in a Mozilla Firefox browser. The page title is "Instances - OpenStack Dash..." and the URL is "172.16.25.136/project/instances/". The user is logged in as "admin". The dashboard shows a sidebar with the OpenStack logo and navigation options for "Project" and "Admin". The main content area is titled "Instances" and features a table with the following data:

Instance Name	Image Name	IP Address	Size	Keypair	Status	Task	Power State	Uptime	Actions
<input type="checkbox"/> Newinstance	Newimage	10.0.0.2	m1.nano 64MB RAM 1 VCPU 0 Disk	-	Active	None	Running	0 minutes	Create Snapshot More

Below the table, it indicates "Displaying 1 item". The interface also includes a search filter, a "Launch Instance" button, and a "Terminate Instances" button.

Figure 5.6: Custom instance successfully running

Chapter 6

Evaluation

This chapter provides the evaluation results of the proposed system for virtual machine images security in Cloud storage. We describe evaluation of our solution against the possible threats on virtual machine images. The security of various system components and the core features of data security are described. The threat model provides the attacks mitigated by various system features. The performance analysis presents the overhead caused by adding image encryption during upload and download in Cloud.

6.1 Evaluation Methodology

The proposed solution in thesis provides security to virtual machine images in Cloud storage. The evaluation scheme is based on validating the effectiveness of security features against various attacks on virtual machine images. To prove the security of proposed system various formal verification techniques will be used. The verification of various algorithms is shown by following the standard guidelines for National Institute of Standards and Technology (NIST) guidelines.

Generally, before verifying the security properties of any system it is important to identify the security attributes. Since our scheme is based on security of images that contain operating system, applications and data, security features such as confidentiality and integrity of images must be ensured. Confidentiality and integrity are the two most areas of computer security. Confidentiality refers to keeping the information secret from anyone except the legitimate user; while integrity refers to ensuring the originality of images such that they are not altered by anyone except the authorized user. Along with measuring the ability of a system to protect data originality these security attributes also provide source originality. These attributes are critical

for the security of our system and guarantee that the images (or files) are trustworthy and accurate.

6.1.1 Evaluation through NIST Guidelines

NIST has published a guideline for storage encryption technologies [53]. This guideline presents the overview of storage security. In a distributed storage system such as Cloud that contains a large amount of data, it is imperative to follow guidelines for data storage security. For the security of data files NIST has provided detailed phases for storage encryption technology. These phases include identifying needs, designing the solution, implementing and testing a prototype, deploying the solution and managing the solution. For designing and implementing the solution it is recommended to use the most secure cryptographic standards for encryption i.e. Advanced Encryption Standard (AES) with key size of 256 bits (AES-256) [54].

After identifying the security needs for our system that include preserving the confidentiality and integrity of virtual machine images, we have used the AES-256 encryption standard for encrypting the images in Cloud storage. To ensure the integrity of images in Cloud storage, hashing technique of MD5 is used. Before encryption of image, its MD5 is taken and stored with the metadata of image file. After decryption of image, its MD5 is calculated again and compared with the MD5 of image in metadata. This ensures that the integrity of images in Cloud storage is maintained. After deploying our solution on image storage component (Swift) of OpenStack, we test the prototype against various attacks on VM images.

6.1.2 Threat Model

Virtual machine images are vulnerable to many attacks in Cloud. These attacks include alteration of images, source code modification, data leakage, backup data exposure and tampering of saved images by hypervisor. Our system provides security to images from all these attacks. The mitigated attacks on disk images and the mechanism for security against these attacks is shown in the table 6.1.

6.2 Verification through Image Exploitation Tools

Various tools exist that can be used to explore the image files and modify them according to the needs. These tools include ISObuster, CDmage, Dae-

man Tools and Archive Manager [55]. These tools provide the utility to open any image file. If someone has unauthorized access to images, he can use tools to explore data inside images. As a result the confidentiality of images can be compromised. These tools also allow modifying image files which can be used by unauthorized users to violate the integrity of images. The figure shows the complete contents of image getting displayed with the tool of Archive Manager without launching image. However, the content of encrypted images cannot be displayed by these tools. As a result, the encrypted images are prevented from data leakage and image modification.

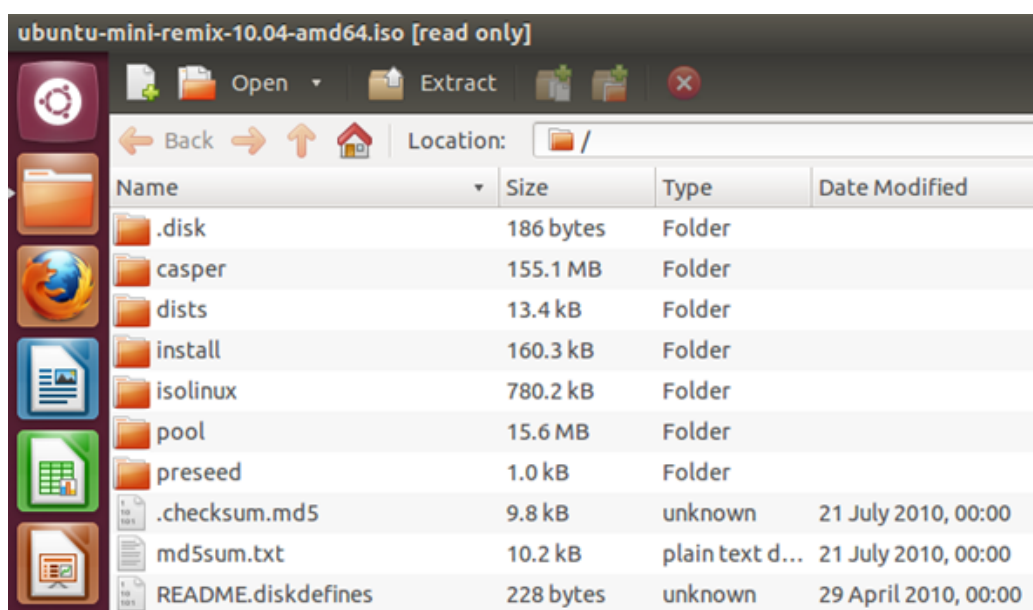


Figure 6.1: Exploring the VM image using Archive Manager

6.3 Performance Evaluation

In Cloud a large number of virtual machines may be present. The number of virtual machines can range from thousands to millions. Each virtual machine has its own image. Image contains the operating system, all the data of virtual machines and is required by VMs to successfully boot. Similarly, when a virtual machine is terminated the image has to be stored on Cloud. Accessing the image to load a virtual machine and storing it while terminating a virtual machine takes some processing time depending on resources and infrastructure. Applying the cryptographic operations such as encryption can increase the processing time for managing images to be used by virtual

machines. The processing time can be increased due to many factors such as increase of image size after encryption, retrieval of encryption keys during encryption and decryption and due to applying the encryption algorithm of AES-256 during VM loading and termination.

We have done analysis using the standard feature of Linux used to measure time interval for execution of a successful command. The analysis involves the performance overhead caused during upload of encrypted images to Swift (OpenStack) and their download from Swift for usage by VMs. Images of different sizes will be taken into consideration. System specifications for testing are core i3 processor with 4 GB RAM.

The table 6.2 shows the time taken by different images during upload with encryption and without encryption. Overall analysis shows that encryption results in 15.8 percent increase in time taken during image upload. Figure 6.2 shows the graphical analysis of time taken by different images during upload in Swift. Table 6.3 shows the download time taken by encrypted images and non-encrypted images. Overall analysis shows that the time taken by encrypted images during download is 14.8 percent more than the time taken by normal images during download. Figure 6.3 shows the graphical analysis of time taken by different images during download from Swift. Overall, encryption causes the overhead of 15.3 percent during image upload and download.

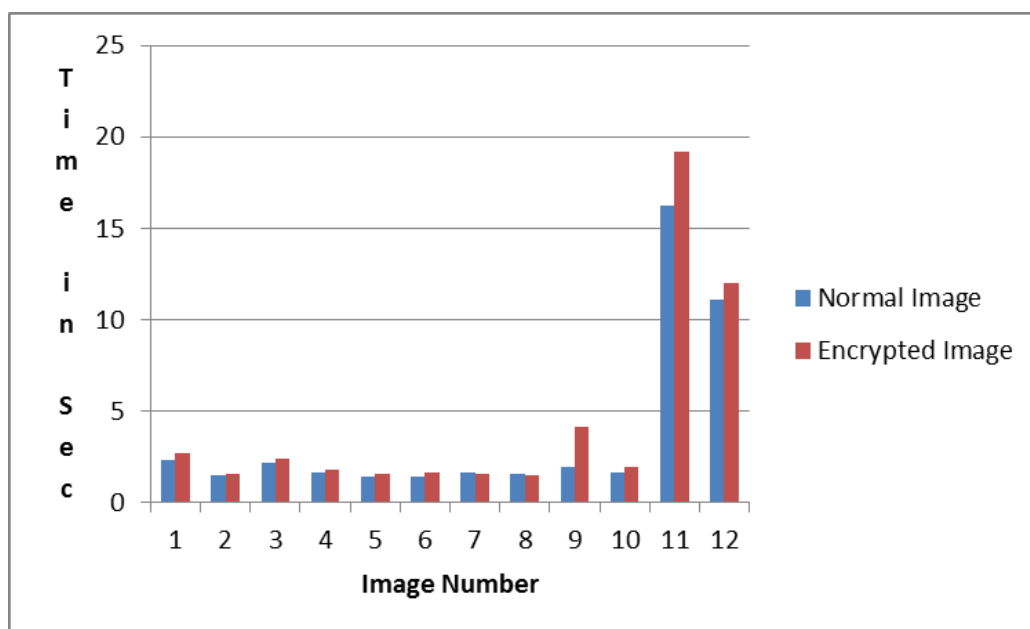


Figure 6.2: Time taken by different images during upload to Swift

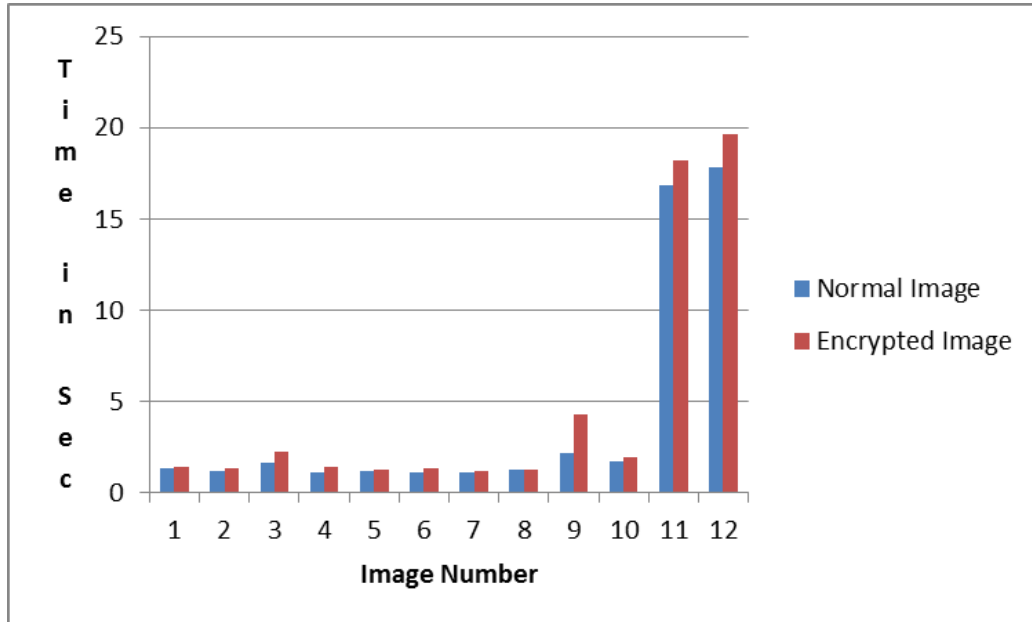


Figure 6.3: Time taken by images during download from Swift

Attack	Status	Entity	Mechanism
Alteration	Mitigated	Disk image	Encryption is used to protect the data from alteration and Forgery. Only authorized users have the privilege to get the keys for decryption.
Data Leakage	Mitigated	Disk image	Encrypted data is unreadable to anyone except the authorized user.
Image modification	Mitigated	Disk image	Hashing is used to prevent any modification of image and ensure its integrity.
Backup data exposure	Mitigated	Disk image	Encrypted backup data is prevented from exposure to unauthorized users.
Hypervisor tampering the saved images	Mitigated	Disk image	Hypervisor is unable to read and modify the encrypted saved images.

Table 6.1: Attacks mitigated with the description of security mechanism

Image	Normal Upload Time (sec)	Upload time after encryption (sec)
cirros-0.3.0-x86_64-disk.img	2.324	2.674
cirros-0.3.0-x86_64-disk.img	1.486	1.567
cirros-0.3.0-i386-rootfs.img	2.191	2.387
cirros-0.3.0-i386-rootfs.img	1.612	1.754
cirros-0.3.0-i386-disk.img	1.393	1.574
cirros-0.3.0-i386-disk.img	1.402	1.665
cirros-0.3.0-arm-rootfs.img	1.64	1.591
cirros-0.3.0-arm-rootfs.img	1.54	1.487
cirros-0.3.0-x86_64-uec.tar.gz	1.934	4.11
cirros-0.3.0-x86_64-uec.tar.gz	1.627	1.942
Ubuntu-mini-remix-10.04-amd64.iso	16.265	19.149
Ubuntu-mini-remix-10.04-amd64.iso	11.057	12.037
Total Time	44.833	51.937

Table 6.2: Image upload time in Swift before and after encryption

Image	Normal download time (sec)	Download time after encryption (sec)
cirros-0.3.0-x86_64-disk.img	1.305	1.422
cirros-0.3.0-x86_64-disk.img	1.184	1.36
cirros-0.3.0-i386-rootfs.img	2.173	2.252
cirros-0.3.0-i386-rootfs.img	1.629	1.378
cirros-0.3.0-i386-disk.img	1.16	1.224
cirros-0.3.0-i386-disk.img	1.101	1.319
cirros-0.3.0-arm-rootfs.img	1.105	1.215
cirros-0.3.0-arm-rootfs.img	1.232	1.283
cirros-0.3.0-x86_64-uec.tar.gz	1.103	4.283
cirros-0.3.0-x86_64-uec.tar.gz	1.726	1.942
Ubuntu-mini-remix-10.04-amd64.iso	16.827	18.185
Ubuntu-mini-remix-10.04-amd64.iso	17.793	19.657
Total Time	48.338	55.52

Table 6.3: Image download time in Swift before and after encryption

Chapter 7

Conclusion and Future Directions

This chapter comprises of two parts. In the first part, we present the conclusion derived from our research. We describe the research goals of our system, the results achieved during research and our major contributions. In the next part of chapter, the future directions of our research work are described. The future research work includes adding security framework to protect virtual machines during execution in Cloud. We also propose security of user accounts in Swift along with image protection as future work.

7.1 Conclusion

The research was carried out in two different phases. The first phase comprised of the analysis of various components of virtualization in Cloud computing. This analysis resulted in identification of security issues related to the virtual machine images. The security of Cloud cannot be maintained unless its virtualization environment is secured. Although different virtualization approaches exist, bare metal virtualization approach is commonly used in large computing systems such as Cloud for server virtualization. Our research analyses general architecture of bare metal virtualization and covers security aspects of its different components. Cloud virtualization environment can be compromised by different attacks at service provider, hypervisor, virtual machines, guest operating system and disk images. We have identified attack scenarios at these components, general requirements for virtualization security and different existing security schemes that provide security to virtualization environment have also been discussed. We present the holistic picture of virtualization security in Cloud through structured analysis in

which security requirements, attacks and solutions correspond to each other.

In the next phase of research, the framework for protection of virtual machine images was designed in Cloud environment. This framework consisted of virtual machine images encrypted storage in Cloud. Disk images are vulnerable to different attacks during storage such as data leakage, malware installation on images and snapshot access in storage. Using the proposed architecture EVDIC, integrity of disk images and confidentiality of data stored on them can be guaranteed during storage. We describe the integration of EVDIC with OpenStack which is open source Cloud platform. EVDIC performs the encryption of images in Object sever of Swift (storage component). The evaluation of the system was done using the NIST guideline on storage technologies. The effectiveness of the system to ensure virtual machine images integrity and confidentiality was verified using different image exploration tools. Furthermore, the performance evaluation shows that our system results in 15.3 percent overhead on the system. This overhead results from accessing images and applying cryptographic operations on them.

7.2 Future Research Directions

Future directions include the problems that can be solved by extending our research work. They provide research directions to solve the related research problems. We describe two important future research directions as further extensions of research which are.

1. Adding security of accounts in OpenStack Swift
2. Encrypted execution of virtual machines in Cloud computing

7.2.1 Adding security of accounts in OpenStack Swift

Swift architecture as discussed in chapter 4 contains four major components. These components are Proxy, Container, Accounts and Objects. Objects are the real files such as VM images that are to be stored in Swift, while Accounts and Containers are databases [56]. Containers contain the lists of Objects in Swift and Accounts store the list of Containers. Anyone with access to Containers or Accounts databases can access the complete list and the IDs of objects stored in Swift even if all the objects are encrypted. Therefore, to secure the list of objects we propose a security framework for Swift Account and Container databases. The framework consists of encrypting the Account and Container databases so that only legitimate users can access them. During the object storage on Swift, first the Object server will encrypt the objects.

The next step will be to encrypt the Container and Accounts databases respectively. Similarly, during object download from Swift, first the Account database and then the Container database will be decrypted. After locating the object from these databases the Object server will decrypt the object.

7.2.2 Encrypted execution of Virtual Machines in Cloud Computing

7.2.2.1 Introduction

Among all the security concerns related to Cloud, Infrastructure as a Service (IaaS) layer security issues are the most critical. IaaS layer contains virtualization components such as virtual machines, hypervisor and virtual network. Virtual machines are vulnerable to many attacks, such as attacker accessing host disk files through his virtual machine, creating rogue virtual machines to occupy system resources and launch a DOS attack at Cloud, and using backdoor virtual machines to leak sensitive data. Providing security to virtual machines is the core of secure IaaS services in Cloud. However, virtual machines security has not been addressed properly due to which virtual machines and thus the complete Cloud system can be compromised by attackers.

Virtual machine images require high integrity, and their security is foundation of overall Cloud computing system security. A large number of virtual machines may be present in a Cloud server in active or offline state. Traditional security solutions for virtual machines have significant limitations, and cannot deal with most threats. So a pre-emptive solution is required that can provide security to VMs and prevent them from all possible attacks. Along with these security issues other issues related to business, economy and research also remain. Different commercial and research solutions have been provided to secure the virtual machines at IaaS layer but the scope of these solutions is quite limited and they are designed to deal with some specific threats to virtual machines. E.g. Seongwook J. et al proposed real time log monitoring to verify the integrity of virtual machines, Mikhail I. et al developed a system called SPARC to prevent VM Checkpoint attacks (in which malicious nodes taking snapshots of virtual machine image states), Trent J. et al used sHype and added hooks to XEN authorization mechanism to prevent covert channel communication between virtual machines in same network and so on.

The purpose of this research will to develop a system that uses cryptographic techniques to encrypt Virtual Machines during execution in Cloud. This proposed system will mitigate a large number of possible threats that

can be performed by compromising the Virtual Machines. Successful execution of this system will protect the source code of virtual machines from tampering and protect data leakage from virtual machines. Moreover, all the active and offline virtual machines will be prevented from attacks through malicious virtual machines of attacker in Cloud, network DOS attacks, from covert channels that are used as a backdoor to transfer data between virtual machines, compromised Hypervisor attacks, VM Checkpoint attacks, attacks on virtual machines during VM boot and storage. This research will provide secure IaaS layer services to Cloud users and encourage Telecom and IT industries to shift their company infrastructure to Cloud, to meet the ever increasing needs of technology and infrastructure. The open source architecture of the proposed system could be used to further extend this system.

7.2.2.2 Objectives of Research

The primary purpose of this project is to provide infrastructure level security to Cloud environment by developing a system that provides secure execution environment for virtual machines and protects them from tampering and leakage of data. Each virtual machine is encrypted in the Cloud, so that no attack can be launched on virtual machines even if the underlying hypervisor or any virtual machine in the system is malicious.

The proposed system will provide a secure platform to virtual machines in cloud environment. Major objectives of the proposed system are as follows.

1. Providing integrity to virtual machines in the cloud so that the source code of VMs can be protected from tampering. Any malicious node in Cloud system such as malicious hypervisor, virtual machine or any attack through network may try to infect virtual machine code to change its functionality, and compromise the integrity of virtual machine. However, the source code of encrypted virtual machine cant be altered.
2. This system will provide confidentiality of data contained in the virtual machines. Virtual machines may contain sensitive customer data and files which the attacker may try to access and read. In the proposed system, even if attacker has access to virtual machine he will not be able to read whats stored inside the VM.
3. The execution of virtual machines will be in a secure environment where they will be safe from all attacks. For example, there will be no covert channels for inter VM communication, one compromised VM will not be able to modify other VMs in system, a compromised hypervisor will

not be able to modify any VM, and storage of VMs in unused state will also be secure.

4. Management of keys used for encryption will be secure and effective. A centralized key management server will be there to deal with all key management issues in an efficient way. This server will follow the security standards for key management given by NIST.

7.2.2.3 Possible Issues

One of the two major components of system will be the key management server that manages keys and policies related to encryption, other will be a virtual machine on top of hypervisor that will export virtual machines and encrypt them according to policy. The storage model will be flexible, that will not only include encryption but also key management through policy. Key policy will remove complexities of key management. The proposed system will be deployed on open source Cloud computing platforms such as OpenStack, which is getting increasingly adopted around the world. The principle technical issues that must be addressed are following:

1. Virtual machines have their own operating system and behave just like a physical machine with their own CPU, RAM and hard drive. A single virtual machine image requires 2GB of memory to run (in OpenStack), and in a complete Cloud system a number of virtual machines are running that requires a lot of physical resources such as RAM, hard disk, and CPU cycles. By encrypting the virtual machines, size of each virtual machine will increase which will consume more resources of the system. These all activities will result in increased performance overhead on system. We will simulate the results of the system, and analyze its performance.
2. In encryption one of the most important issue is the secure storage and retrieval of keys used in encryption. Weak key management can compromise strong encryption algorithm. Eventually the security of data and information protected by encryption depends on the security of keys. So the mechanism for the protection of keys must be effective to protect all private and public keys from modification, and private keys from disclosure. In the proposed system, there will be a dedicated server which will ensure secure storage of keys and all issues related to key management. This server will follow the best security practices for key management mentioned by NIST.

7.2.2.4 Final Results

The following results will be obtained from this research.

1. A functional system that can be deployed on existing Cloud computing systems, to provide IaaS layer security by encryption of virtual machines. Virtual machines in Cloud are provided an environment in which they can execute securely without their integrity and confidentiality getting compromised.
2. Cloud vendors can use the proposed system to provide secure infrastructure layer security to the customers.
3. The proposed system will provide security services to the organizations using Cloud by mitigating major threats to their infrastructure.
4. Virtual machines security in Cloud is a new topic of research. The open source architecture of proposed system can be studied and extended by researchers to add further security features for IaaS layer.

Appendix A

Documentation on Adding Encryption to OpenStack Swift

A.1 Deploying OpenStack

In order to perform development in OpenStack, they have provided a developers version of OpenStack called as DevStack. It is easy to deploy and provides a development platform for OpenStack. DevStack is a documented shell script to build complete OpenStack development environments and is located at <http://devstack.org/>. The recommended way to install DevStack is on Linux running on a virtual machine. Typical configuration used for installing Devstack is as follows.

- Ubuntu 12.04 LTS as host OS
- Ubuntu 12.04 LTS as guest OS running on VMware Player

For deploying DevStack on your virtual machine, you have to make sure that git is installed inside your VM. Install it using the following commands.

```
apt-get update
apt-get install git
```

Clone the DevStack repo using the following and navigate to devstack directory.

```
git clone https://github.com/openstack-dev/devstack.git
cd devstack
```

We are not going to execute the `stack.sh` script right now because we have to modify our `localrc` to disable all other services except `swift` and `keystone`. The `localrc` file (located in `devstack`) contains some of the configuration primitives which can be modified as-per-need basis. At this point, create a `localrc` file in `devstack` directory and add following lines in it.

```
ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
RABBIT_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=token
```

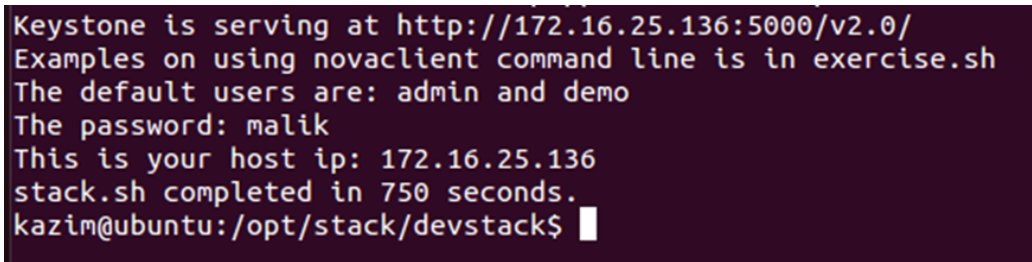
Now add the following lines.

```
disable_all_services
enable_service key mysql s-proxy s-object s-container
                s-account
```

We are now ready to start OpenStack. `cd` into the `devstack` directory and execute the `stack.sh` script using the following command.

```
./stack.sh
```

After the execution is complete, you will see output similar to one shown in figure A.1.



```
Keystone is serving at http://172.16.25.136:5000/v2.0/
Examples on using novaclient command line is in exercise.sh
The default users are: admin and demo
The password: malik
This is your host ip: 172.16.25.136
stack.sh completed in 750 seconds.
kazim@ubuntu:/opt/stack/devstack$
```

Figure A.1: Output of the running DevStack

You have a working deployment of OpenStack, configured to run only `Keystone` and `Swift`. Hence don't expect the OpenStack GUI (`Horizon`) to work right now as it is not enabled.

A.1.1 Adding objects to OpenStack Swift

At this point, you should already be familiar with `Keystone`, its relation with every other component of OpenStack and its necessity. `Keystone` is an OpenStack project that provides `Identity`, `Token`, `Catalog` and `Policy` services for

use specifically by projects in the OpenStack family. The Keystone Identity Service allows clients to obtain tokens that can be used to access OpenStack cloud services.

To add objects to Swift, go into the DevStack directory. The absolute path of DevStack Swift is `/opt/stack /devstack`. Enter the following command in your terminal.

```
source openrc
```

This command authorizes you for using OpenStack Swift and generates a token for you. Without it, you cannot access Swift. Now that we are authorized, enter the following command.

```
swift stat
```

You should see something like this.

```
Account: AUTH_54805eb6837e40818998e59ee6ea40c7
Containers: 0
Objects: 0
Bytes: 0
Accept-Ranges: bytes
X-Timestamp: 1379268007.20819
X-Trans-Id: tx93f58db540574b228fd87effd76c5221
Content-Type: text/plain; charset=utf-8
```

Now that we have got complete access to Swift, lets create containers and objects. Enter the following command to create a container in OpenStack Swift.

```
swift post TestContainer
```

This will create a new container called `TestContainer`. Status of the new container can be seen by giving the command `Swift stat TestContainer`. Containers are essentially databases that are used to store information about objects. Consider them as directories where you can store files (objects). Figure A.2 shows the status of new container created.

Now enter `swift stat`. You will see that your account now contains a single container. You can also physically see a container. Open your root directory and navigate to `/opt/stack/data/swift/drives/sdbX/Y/sdbZ/containers`.

Go deep into the nested directories inside the container directory and you will eventually find a file named as `xxx.db`. This is the database that will hold information about objects stored by Swift.

If you want to know more about the container, enter `swift stat TestContainer`. Now lets upload a test file. Navigate to any directory containing a desired file to be uploaded and enter the following command.

```
kazim@ubuntu:/opt/stack/devstack$ swift stat TestContainer
  Account: AUTH_71900f62ab4441cab52b8d550a26b0f5
Container: TestContainer
  Objects: 0
   Bytes: 0
  Read ACL:
Write ACL:
  Sync To:
  Sync Key:
Accept-Ranges: bytes
X-Timestamp: 1384843605.14920
X-Trans-Id: tx2eac3c10376048498e821-00528b095f
Content-Type: text/plain; charset=utf-8
kazim@ubuntu:/opt/stack/devstack$ █
```

Figure A.2: Status of Container named TestContainer

```
swift upload TestContainer Test_File
```

That was it for some of the basic operations on Swift. You can view CLI help by issuing.

```
swift h
```

A.2 Debugging OpenStack

After DevStack is configured, navigate to `opt/stack` directory on your machine. All of the code for OpenStack components is located in their respective directories. Now navigate to `swift/swift` directory. These directories contain code for their respective components. For instance, `account` will contain code for the account server, `obj` will contain the code for object server and so on. Following are the most important directories:

- `account`
- `common`
- `container`
- `obj`
- `proxy`

We will debug the code in a bit but lets, first make some important things clear. All the code in these directories are downloaded from online Swift GitHub repos. You can modify these repos to download and replace your code instead of the default ones. This is very important. As soon as you start working on your own code, you will have to add it to some version control to keep track of your changes. You can then share that repository with anyone who can then download a working copy such that his OpenStack deployment will use your code instead of the default one.

Repository locations are stored in `stackrc` file in the `devstack` directory. Lets now replace the default location of the swift repo with your own. Go to `https://github.com/openstack/swift`

and fork the repo. After that, copy the forked repo's clone URL and paste it in `stackrc`.

Deploy OpenStack by executing `stack.sh` script. This deployment will now run your code (in this case Swift code. At this point, you should be comfortable with the Python Debugger (`pdb`) because this is the tool we would rely on for debugging Swift. We have to do some minor tweaks so lets do them first.

Locate `utils.py` in `common` directory of `swift` and add following lines in `utils.py`.

```
stdio_files = [sys.stdin, sys.stdout, sys.stderr]
stdio_files = []
```

Now lets set our first breakpoint. Navigate to the obj directory and open server.py. Navigate to the class ObjectController's PUT function and add the following lines:

```
import pdb
pdb.set_trace()
```

This sets a breakpoint in the PUT function of the ObjectController class and will hit each time you try uploading a file to the server. Save the file and exit. The next step is the actual debugging.

Ensure that you have two terminals opened side by side. Enter screen -r in one of the terminal (call it Terminal2). Hit Ctrl + A + X where X corresponds to the number against s-object.

Now on the other terminal (Terminal1), issue upload commands to a swift container of your choice.

You will be able to hit breakpoint onTerminal2 where you can debug it using normal pdb commands. In case if something goes wrong in Terminal2, hit Ctrl + C, press the up arrow key and hit enter. This will restart the object server and you are good to go again with your debugging.

A.3 OpenStack Object Server Code Break-down and Object Encryption

OpenStack Swift API is implemented as a set of ReSTful web services. Navigate to the `opt/stack` directory on your virtual machine. Now, navigate to `swift/swift/obj` directory and open `server.py` in your favorite Python editor. `server.py` contains most of the Swift Object Server code.

- `DiskFile` class for file I/O
- `ObjectController` class which handles all the requests made by the user to the Swift object storage

You will find the conventional requests methods such as PUT, POST and GET inside the `ObjectController` class. The point of our focus here is file PUT so we are going to skip the breakdown of POST and GET functions. To upload a file to the object storage, a put request is issued and handled by PUT function of the `ObjectController` class.

At this point, it would be a good idea to setup breakpoints in the `ObjectController` PUT function and upload a file a file to start debugging. When you upload a file to the object storage, it is not written in a single go. Each file is read into a number of small chunks and are stored in a temporary file which acts as a buffer. Contents from the temporary file are then written to a permanent file when buffer size reaches a certain threshold value (default is 512 MB). Following are some of the important lines/loops from the PUT function.

```
file = DiskFile(self.devices, device, partition, account,
                container, obj, self.logger, disk_chunk_size=
                self.disk_chunk_size, origin_disk_chunk_size=
                self.origin_disk_chunk_size)
with file.mkstemp() as fd:
    for chunk in iter(lambda:
        reader(self.network_chunk_size), ''):
        while chunk:
            written = os.write(fd, chunk)
            chunk = chunk[written:]
    %# For large files sync every 512MB (by default) written
    if upload_size-last_sync>=self.bytes_per_sync:
        tpool.execute(fdatasync, fd)
        drop_buffer_cache(fd, last_sync,
```

```

                                upload_size-last_sync)
last_sync = upload_size

```

With this basic understanding of how a call to PUT works, we will now talk about object encryption. Since each of the chunks is written to a buffer, it would be wise to encrypt them before they are written to it. In this way, contents are copied from the buffer to the actual file will be encrypted such that when all chunks are written to the actual file, the file will be encrypted. To handle encryption, we have created a separate `Encryptor.py` file that will handle the encryption and return encrypted chunks. Our Encryption classes use the popular `M2Crypto` package to perform encryption. `M2Crypto` is present as a fully supported package in the Ubuntu 10.04 and later repositories. It can be installed using the command.

```
sudo apt-get install python-m2crypto
```

The following line in the PUT function of `server.py` will perform and return an encrypted chunk.

```

enc = encrypted();
enc.EncryptFile(fname)

```

Parameter `fname` is the path of the file that is to be written to disk. Writing the encrypted chunks would be similar to our previous descriptions. Note that the current implementation uses a constant key that is saved in a configuration file. Our implementation lacks a functional keystore. See the following code used for encryption in the `encrypted` class in `Encrypt.py` file.

```

class encrypted:
    def __init__(self):
        self.key=128
        self.alg='aes_128_cbc'
        self.key="123452345"
        self.iv = '3141527182810345'
    def set_state(self, op=ENC):
        self.cipher =M2Crypto.EVP.Cipher
        (alg=self.alg,key=self.key,
        iv=self.iv, op=op)
    def encrypt(self,msg):
        v = self.cipher.update(msg)
        v = v + self.cipher.final()
        return v
    def EncryptFile(self,dname):

```

```
os.chdir(dname)
for filename in os.listdir('.'):
    with open(filename) as f:
        content = f.read()
        f.close()
    self.set_state(1)
    content = self.encrypt(content)
    print 'Successful Encryption'
    faname = os.path.join(dname, filename)
    f = open(faname, 'w')
    f.write(content)
    f.close();
```

Decryption is analogous to our description above but will occur in the ObjectController class. When a request is made to retrieve a file, the object server returns an iterator to retrieve that file. It is the responsibility of that GET function of server.py to decrypt file before returning it. Following code in Encrypt.py performs the decryption.

```
def DecryptFile(self, dname, obj):
    objectname = obj
    os.chdir(dname)
    for filename in os.listdir('.'):
        with open(filename) as f:
            content = f.read()
            f.close()
        self.set_state(0)
        content = self.encrypt(content)
        print 'Successful Decryption'
        completeName = os.path.join(dname, filename)
        f = open(completeName, 'w')
        f.write(content)
        f.close();
```

A.4 Deployment

For deploying the Swift encryption code in OpenStack, following steps need to be followed.

- It is recommend a fresh VM of Ubuntu desktop, as previously saved changes in Devstack can get lost.
- Create a folder in /opt named stack.
- Go to /opt/stack and copy the devstack.tar folder provided with code.
- From command line, extract the devstack.tar.gz folder in /opt/stack using the following command: `tar xpvzf devstack.tar.gz devstack/`
- Now extract the swift.tar.gz with encryption code, using same command in the /opt/stack folder.
- Run the Devstack script and it will download nova, keystone, glance, horizon and others services from internet.
- Change glance/etc/glance-api.conf file to use Swift store.
- Run the script `./stack.sh`.
- After successful completion of script, goto `/opt/stack/data/swift/1/sdb1/objects`. Delete the existing objects here so that you can check for the objects you add by yourself.
- From dashboard, add an .img Qcow2 bootable image. Check the image you just uploaded from dashboard in `/opt/stack/data/swift/1/sdb1/objects`. This image will be stored encrypted. This can be verified by using image size and md5sum as mentioned in documentation.
- Launch an instance from this image, the image will be decrypted before use by VM and VM should be launched in 'active' state. You should be able to access its console.

A.5 Testing

Complete OpenStack deployment through DevStack can be used accessed through dashboard. The URL on which Horizon is running after the successful execution of DevStack can be entered to browser. Login using the credentials (username and password) obtained from Keystone after successful execution of DevStack as shown in figure A.3.

```
Horizon is now available at http://172.16.25.136/
Keystone is serving at http://172.16.25.136:5000/v2.0/
Examples on using novaclient command line is in exercise.sh
The default users are: admin and demo
The password: malik
This is your host ip: 172.16.25.136
stack.sh completed in 750 seconds.
kazim@ubuntu:/opt/stack/devstack$
```

Figure A.3: Horizon URL and User Credentials

After the login, new and custom images can be added to OpenStack. The custom bootable image for OpenStack named cirros can be downloaded from <https://launchpad.net/cirros/+download>.

For test case, we can download a Qcow2 bootable image from above link. After downloading the image, go to Images tab on left side of Dashboard. Click on create image, you can browse to the qcow image and select it to upload. You can name the image and select its format. All properties during creation of custom image named Newimage are shown in figure A.4.

This image is present in encrypted state on the disk. The location of image can be seen in the directory `/opt/stack/data/swift/drives/sdbX/Y/sdbZ/Objects/`. To verify that the image is encrypted, we can see the size of image uploaded and the original image that was downloaded. Moreover, we can use Md5sum to take hashes of the two files and compare them.

Figure A.6 shows the md5sum and size of images added in Swift without encryption. Without encryption, the md5sum and size of original and uploaded images are same.

To launch an instance using the encrypted image, click on Instance tab on left and then on launch instance. Select the custom image named Newimage to be used by the instance. The instance created using the Newimage is shown in figure A.7.

Create An Image

Name
Newimage

Description
Additional information here...

Image Source
Image File

Image File
/home/kazim/Desktop/cirros-0.3.1 [Browse...](#)

Format
VDI

Minimum Disk (GB)
1

Minimum Ram (MB)
64

Description:
Specify an image to upload to the Image Service.
Currently only images available via an HTTP URL are supported. The image location must be accessible to the Image Service. Compressed image binaries are supported (.zip and .tar.gz.)
Please note: The Image Location field MUST be a valid and direct URL to the image binary. URLs that redirect or serve error pages will result in unusable images.

Figure A.4: Image creation process in OpenStack

OpenStack Dashboard - Mozilla Firefox

Images - OpenStack Dashbo... Google

172.16.25.136/admin/images/images/

Logged in as: admin Settings Help Sign Out

Images [+ Create Image](#) [Delete Images](#)

<input type="checkbox"/>	Image Name	Type	Status	Public	Protected	Format	Actions
<input type="checkbox"/>	Newimage	Image	Active	Yes	No	RAW	Edit More

Displaying 1 item

Figure A.5: Successful creation of image on OpenStack

The image on disk will be automatically decrypted when you click on launch. The instance can be used by the user and after the instance termination; the image will be saved in encrypted form on disk. Since the image

```

kazim@ubuntu:/opt/stack/devstack$ swift post test
kazim@ubuntu:/opt/stack/devstack$ swift upload test cirros-0.3.0-i386-disk.img
cirros-0.3.0-i386-disk.img
kazim@ubuntu:/opt/stack/devstack$ md5sum /opt/stack/devstack/cirros-0.3.0-i386-disk.img
a780d603d326da412999b6c5094d31b5 /opt/stack/devstack/cirros-0.3.0-i386-disk.img
kazim@ubuntu:/opt/stack/devstack$ md5sum /opt/stack/data/swift/1/sdb1/objects/161/ea5/50d39323b4da5637f076d544974d8ea5/1385725959.63670.
data
a780d603d326da412999b6c5094d31b5 /opt/stack/data/swift/1/sdb1/objects/161/ea5/50d39323b4da5637f076d544974d8ea5/1385725959.63670.data
kazim@ubuntu:/opt/stack/devstack$ ls -l /opt/stack/devstack/cirros-0.3.0-i386-disk.img
-rw-rw-r-- 1 kazim kazim 9159200 Nov  8 02:00 /opt/stack/devstack/cirros-0.3.0-i386-disk.img
kazim@ubuntu:/opt/stack/devstack$ ls -l /opt/stack/data/swift/1/sdb1/objects/161/ea5/50d39323b4da5637f076d544974d8ea5/1385725959.63670.d
ata
-rw----- 1 kazim kazim 9159200 Nov 29 03:52 /opt/stack/data/swift/1/sdb1/objects/161/ea5/50d39323b4da5637f076d544974d8ea5/1385725959.6
3670.data
kazim@ubuntu:/opt/stack/devstack$ █

```

Figure A.6: Successful creation of image on OpenStack

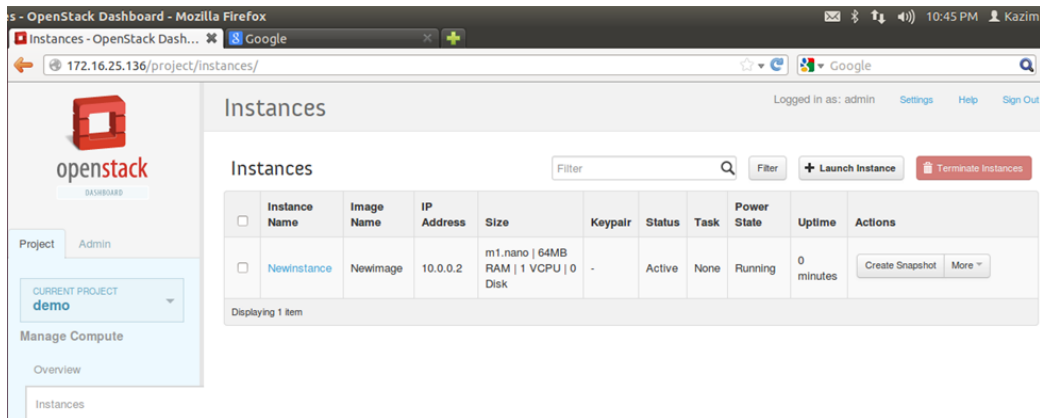


Figure A.7: Custom instance successfully running

is present is encrypted state on disk, the data on image cannot be read and accessed unless it is decrypted. Hence, the confidentiality of image in Cloud storage is preserved.

A.6 Configuring Glance to store virtual machine images

The configuration file for the Image API is `/etc/glance/glance-api.conf`. To configure glance for using Swift, modify the `/etc/glance/glance-api.conf` file.

```
default_store = swift
```

A.7 Enable and Disable any service in OpenStack

The `devstack` directory contains a file named `localrc`. `localrc` file can be used to enable or disable or any service of OpenStack. To use Swift for development, only Keystone and Mysql can be enabled with Swift after disabling all other services. It can be done using the following commands.

```
disable_all_services
enable_service key mysql s-proxy s-object s-account
                s-container
```

Similarly, any other OpenStack service can be enabled by modifying `localrc`.

A.8 Using Devstack with proxy server

For running Devstack behind the proxy server, following lines need to added to the `localrc` file in `/opt/stack/devstack` folder. It is assumed that proxy address is `10.1.11.11` with port `8080`.

```
http_proxy = http://10.1.11.11:8080/
https_proxy = http://10.1.11.11:8080/
export no_proxy = "localhost,127.0.0.1"
HOST_IP=localhost
SERVICE_HOST=$HOST_IP
IMAGE_HOST=$HOST_IP
IDENTITY_HOST=$HOST_IP
```


A.9 Maintaining Logs for Devstack

To maintain logs for Devstack, following line needs to be added to the localrc file in /opt/stack/devstack directory. The logs will be maintained in /opt/stack/logs directory.

```
LOGFILE=$DEST/logs/stack.sh.log
```

A.10 Common Errors

A.10.1 E: Could not get lock /var/lib/dpkg/lock - open (11 Resource temporarily unavailable)

A common error encountered while running a devstack script is, E: Could not get lock /var/lib/dpkg/lock - open (11 Resource temporarily unavailable). To solve this issue, run the following commands in devstack directory before rerunning script.

```
sudo rm /var/lib/apt/lists/lock
sudo rm /var/cache/apt/archives/lock
```

A.10.2 Invalid Nova Credentials (Unauthorized HTTP 401)

The admin user credentials also need to be put into /etc/nova/api-paste.ini. The values that need to be added in .ini file are

```
admin_tenant_name, admin_user, and admin_password.
```

More details can be found here, <https://ask.openstack.org/en/question/3571/nova-client-error-unauthorized-http-401>.

Bibliography

- [1] “Cloud computing service models,” <http://www.ibm.com/developerworks/cloud/library/cl-cloudservicemodels/?cmp=dw&cpb=dweld&ct=dwnew&cr=dwnen&ccy=zz&csr=021011>, last Accessed: 2012-10-26.
- [2] “Cloud computing architectural framework,” https://wiki.cloudsecurityalliance.org/guidance/index.php/Cloud_Computing_Architectural_Framework, last Accessed: 2012-10-27.
- [3] D. Orlando, “Cloud computing service models,” <http://www.ibm.com/developerworks/cloud/library/cl-cloudservices1iaas/cl-cloudservices1iaas-pdf.pdf>, last Accessed: 2012-10-27.
- [4] “What is cloud?” <http://www.ibm.com/cloud-computing/in/en/what-is-cloud-computing.html>, last Accessed: 2012-10-29.
- [5] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum, “Disco: Running commodity operating systems on scalable multiprocessors,” *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 4, pp. 412–447, 1997.
- [6] “Virtualization overview,” <http://www.vmware.com/pdf/virtualization.pdf>, last Accessed: 2013-11-14.
- [7] “Virtual machines security guidelines,” http://www.lasr.cs.ucla.edu/classes/239_1.fall10/papers/CIS_VM_Benchmark_v1.0.pdf, last Accessed: 2013-09-26.
- [8] P. Hoffman, K. Scarfone, and M. Souppaya, “Guide to security for full virtualization technologies,” *National Institute of Standards and Technology (NIST)*, pp. 800–125, 2011.
- [9] J. Kirch. (2007) Virtual machine security guidelines version 1.0. The Center for Internet Security. [Online]. Available: http://www.lasr.cs.ucla.edu/classes/239_1.fall10/papers/CIS_VM_Benchmark_v1.0.pdf

-
- [10] D. Shackelford. (2010) A guide to virtualization hardening guides. SANS. [Online]. Available: http://www.sans.org/reading_room/analysts_program/vmware-guide-may-2010.pdf
- [11] G. Brunette and R. Mogull, “Security guidance for critical areas of focus in cloud computing v2.1,” *Cloud Security Alliance*, pp. 1–76, 2009.
- [12] V. S. I. G. P. S. S. Council, “Pci dss virtualization guidelines v2.0,” pp. 1–39, 2011.
- [13] “The virtualization practice,” <http://www.virtualizationpractice.com/threat-analysis-layers-upon-layers-8942>, last Accessed: 2012-12-04.
- [14] “State of cloud security report: Targeted attacks and real world hacks,” <http://www.alertlogic.com/resources/cloud-security-report/>, last Accessed: 2013-04-14.
- [15] J. Szefer, E. Keller, R. Lee, and J. Rexford, “Eliminating the hypervisor attack surface for a more secure cloud,” in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 401–412.
- [16] J. Szefer and R. Lee, “A case for hardware protection of guest vms from compromised hypervisors in cloud computing,” in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*. IEEE, 2011, pp. 248–252.
- [17] “Best practices for computer security,” <http://kb.iu.edu/data/akln.html>, last Accessed: 2012-12-13.
- [18] J. Kong, “Protecting the confidentiality of virtual machines against untrusted host,” in *Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium on*. IEEE, 2010, pp. 364–368.
- [19] M. Gofman, R. Luo, P. Yang, and K. Gopalan, “Sparc: a security and privacy aware virtual machine checkpointing mechanism,” in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. ACM, 2011, pp. 115–124.
- [20] S. Jin, J. Ahn, S. Cha, and J. Huh, “Architectural support for secure virtualization under a vulnerable hypervisor,” in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 272–283.

-
- [21] A. Ibrahim, J. Hamlyn-harris, and J. Grundy, “Emerging security challenges of cloud virtual infrastructure,” 2010.
- [22] E. Ray and E. Schultz, “Virtualization security,” in *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. ACM, 2009, p. 42.
- [23] J. Reuben, “A survey on virtual machine security,” *Helsinki University of Technology*, 2007.
- [24] “Crisis malware targets virtual machines,” <http://www.zdnet.com/crisis-malware-targets-virtual-machines-7000002986>, last Accessed: 2012-12-24.
- [25] R. Shea and J. Liu, “Understanding the impact of denial of service attacks on virtual machines,” in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*. IEEE Press, 2012, p. 27.
- [26] W. Zhou, P. Ning, X. Zhang, G. Ammons, R. Wang, and V. Bala, “Always up-to-date: scalable offline patching of vm images in a compute cloud,” in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 377–386.
- [27] “Authentication and access control the corner stone of information security,” <http://www.tutorialspoint.com/white-papers/40.pdf>, last Accessed: 2012-12-26.
- [28] R. Khan, J. Ylitalo, and A. Ahmed, “Openid authentication as a service in openstack,” in *Information Assurance and Security (IAS), 2011 7th International Conference on*. IEEE, 2011, pp. 372–377.
- [29] S. Rueda, Y. Sreenivasan, and T. Jaeger, “Flexible security configuration for virtual machines,” in *Proceedings of the 2nd ACM workshop on Computer security architectures*. ACM, 2008, pp. 35–44.
- [30] Z. Wang and X. Jiang, “Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity,” in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 380–395.
- [31] D. Shackelford, *Virtualization Security: Protecting Virtualized Environments*. John Wiley & Sons, 2012.

-
- [32] D. Pelleg, M. Ben-Yehuda, R. Harper, L. Spainhower, and T. Adeshiyan, “Vigilant—out-of-band detection of failures in virtual machines,” *Operating systems review*, vol. 42, no. 1, p. 26, 2008.
- [33] F. Lombardi and R. Pietro, “Secure virtualization for cloud computing,” *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1113–1122, 2011.
- [34] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan, “Tvdc: managing security in the trusted virtual datacenter,” *ACM SIGOPS Operating Systems Review*, vol. 42, no. 1, pp. 40–47, 2008.
- [35] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. Doorn, J. L. Griffin, and S. Berger, “shype: Secure hypervisor approach to trusted virtualized systems,” *Techn. Rep. RC23511*, 2005.
- [36] T. Jaeger, R. Sailer, and Y. Sreenivasan, “Managing the risk of covert information flows in virtual machine systems,” in *Proceedings of the 12th ACM symposium on Access control models and technologies*. ACM, 2007, pp. 81–90.
- [37] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, “Managing security of virtual machine images in a cloud environment,” in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 91–96.
- [38] W. Zhou, P. Ning, X. Zhang, G. Ammons, R. Wang, and V. Bala, “Always up-to-date: scalable offline patching of vm images in a compute cloud,” in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 377–386.
- [39] “Storage made easy,” <https://storagemadeeasy.com/files/07bbead85c6467c40f69f20156d23524.pdf>, last Accessed: 2014-02-06.
- [40] “Piston cloud computing,” <http://www.pistoncloud.com>, last Accessed: 2013-02-21.
- [41] “Metacloud, inc.” www.metacloud.com, last Accessed: 2013-02-21.
- [42] W. Goddard and S. Melville, *Research methodology: An introduction*. Juta and Company Ltd, 2004.
- [43] W. Booth, G. Colomb, and J. Williams, *The craft of research*. University of Chicago Press, 2003.

-
- [44] D. Wagner and B. Schneier, “Analysis of the ssl 3.0 protocol,” in *The Second USENIX Workshop on Electronic Commerce Proceedings*, 1996, pp. 29–40.
- [45] “Openstack,” <http://www.openstack.org>, last Accessed: 2013-06-28.
- [46] “Openstack compute: Administration guide,” <http://docs.openstack.org/trunk/openstack-compute/admin/bk-compute-adminguide-trunk.pdf>, last Accessed: 2013-06-22.
- [47] “Openstack swift architecture,” <http://swiftstack.com/openstack-swift/>, last Accessed: 2013-08-12.
- [48] “Openstack/swift github,” <https://github.com/openstack/swift/>, last Accessed: 2014-01-28.
- [49] “M2crypto 0.22.3 : Python package index,” <https://pypi.python.org/pypi/M2Crypto>, last Accessed: 2014-01-16.
- [50] “Devstack,” <http://devstack.org>, last Accessed: 2014-01-30.
- [51] K. Jackson, *OpenStack Cloud Computing Cookbook*. Packt Publishing Ltd, 2012.
- [52] “Cirros project files,” <https://launchpad.net/cirros/+download>, last Accessed: 2014-01-30.
- [53] K. Scarfone, M. Souppaya, and M. Sexton, “Guide to storage encryption technologies for end user devices,” *NIST Special Publication*, vol. 800, p. 111, 2007.
- [54] P. FIPS, “197: Advanced encryption standard (aes),” *National Institute of Standards and Technology*, 2001.
- [55] “Extraction of contents from image (iso) file,” <http://blogs.technet.com/b/odsupport/archive/2011/04/19/how-to-extract-the-contents-from-an-iso-file-without-burning-the-iso-to-disc.aspx>, last Accessed: 2014-01-16.
- [56] “Swift architectural overview,” http://docs.openstack.org/developer/swift/overview_architecture.html, last Accessed: 2014-01-26.