



UNIVERSITETET I AGDER

Analysis of Security in Cloud Platforms using OpenStack as Case Study

by

John David Cooper

A Thesis Submitted In

Partial Fulfillment of the

Requirements for the of Degree of

Masters of Science
in ICT

The University Of Agder
Faculty of Engineering and Science

Supervised by:

Prof. Vladimir . A . Oleshchuk — University of Agder

Gunnar Gudlaugsson —DevoTeam AS

June, 2013.

Grimstad Norway.

An intelligent mind acquires knowledge, and the ear of the wise seeks knowledge.

—**Proverbs 18:15**

Dedication

This thesis is dedicated to my parents, the late Mrs Cecilia Cooper and Mr David John Cooper. It is also dedicated to my unborn children.

Acknowledgment

First and foremost, I offer my sincerest gratitude to God for His love, provision and wisdom throughout my life.

It is with immense gratitude that I acknowledge the support of my principal supervisor, Prof Vladimir A. Oleschuk. This thesis would not have been possible without his help and guidance, not to mention his advice and unsurpassed knowledge in Information Security. Also, the good advice and friendship of my external supervisor, Gunnar Gudlaugsson has been priceless for both on academics and on personal level, for which I am grateful.

I would like to acknowledge the entire management of DevoTeam AS (Norway), for providing me with necessary tools, devices and platform to undertake this project.

I am most grateful to my dad, Mr David John Cooper, my sister, Mrs Sarah. M. Kamara, and also Mr and Mrs Emmanuel G Bowah, and Mrs Sarah Deen for their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

To my special friends Mr and Mrs Tomm Laurendz, and Mr and Mrs Ghislain Maurice Norbert Isabwe, I say many thanks for supporting me in many ways.

Last, but by no means least, I would like to thank my fiancée and best friend, Haja Saramba Kandeh for her great patience. For without her love, encouragement and editing assistance, I would not have finished this thesis.

[For any errors or inadequacies that will remain in this work, of course, the responsibility is entirely mine].

Abstract

In the last few years, cloud computing has grown from being a promising business concept to one of the fastest growing segments of the IT industry. Big companies like Amazon, Google, Microsoft etc., expand their market by adopting Cloud Computing systems which enhance their services provided to a large number of users. However, security and privacy issues present a strong barrier for users to adapt into the Cloud. This research investigates the security features and issues of cloud platforms using OpenStack as a case study. The goal was to identify security weakness in terms of Authentication and Identity Management(IAM), and Data Management. Base on the findings, specific recommendations on security standards and management models have been proffered in order to address these problems. These Recommendations if implemented, will lead to trust in cloud computing systems, which in turn would encourage more companies to adopt cloud computing, as a means of providing better IT services.

Keywords: Cloud Computing, Security & OpenStack.

Table of Contents

List of Tables.....	ix
List of Figures.....	x
Abbreviations.....	xi
CHAPTER ONE.....	1
1.1 Introduction.....	1
1.2 Prior Research on the Topic.....	3
1.3 Overview of OpenStack.....	5
1.4 OpenStack Projects.....	6
CHAPTER TWO.....	8
2.1 Motivations.....	8
2.2 Objectives and Limitations	8
2.3 Methodology.....	9
2.4 Deployment of OpenStack.....	10
CHAPTER THREE.....	14
3.1 Security Issues From Public and Private Cloud Service Providers.....	14
3.1.1 Authentication and Identity Management.....	14
3.1.2 Access Control	16
3.1.3 Privacy and Data Protection.....	17
3.2 Recommended Security and Privacy Approaches.....	17
3.2.1 Authentication and Identity Management solution.....	18
3.2.1.1 Details of Proposed Scheme	19
3.2.1.2 Conclusion.....	24
3.2.2 Access Control Approach.....	25
3.2.2.1 Scheme Description.....	25
3.2.2.2 Conclusion of proposed scheme.....	31
3.2.3 Privacy and Data Protection solution.....	31

3.2.3.1 Architecture of the Scheme	31
3.2.3.2 Privacy Requirement and Analysis	32
3.2.3.3 Conclusion.....	35
CHAPTER FOUR.....	36
Identity and Access management.....	36
4.1 User Identity Provisioning/De-provisioning.....	36
4.1.1 Overview.....	36
4.1.2 Elevation of Privileges in OpenStack Object Storage.....	38
4.2 Identity Federation.....	40
4.3 Authentication In OpenStack Object Storage.....	40
4.3.1 Overview.....	40
4.3. 2 Password Strength.....	41
4.3.3 Password storage.....	41
4.3.4 Analysis of Authentication Tokens.....	43
4.4 Authorization and Access Control.....	44
4.4.1 Overview.....	44
4.4.2 Too Much Access Rights for Reseller Admins.....	46
4.4.3 Elevation of Privileges Protection.....	48
4.5 Recommendations for IAM Issues in OpenStack Swift.....	49
CHAPTER FIVE.....	51
Data Management In OpenStack Object Storage	51
5.1 Locating Data In OpenStack.....	51
5.2 OpenStack Isolation and Possible Attacks.....	52
5.2.1How Isolation is Done In OpenStack Object Storage.....	52
5.2.2 Attacks on OpenStack Isolation.....	53
5.3 Backup and Recovery.....	53
5.4 Data Deletion.....	54
5.5 Encryption and Key Management.....	55

5.6 Data Integrity	55
5.7 Recommendations for Data Management.....	57
CHATER SIX.....	58
Summary of Contributions and Future Work.....	58
6.1 Contributions.....	58
6.2 Future Work.....	60
Glossary.....	61
Bibliography.....	63
Appendix A.....	69
Appendix B.....	71

List of Tables

3.1 Description of Notations used in the AIM Scheme.....	20
3.2 Notations used In Access Control Scheme.....	25

List of Figures

1.1 Cloud Computing logical Diagram.....	1
1.2 The relationship between the three components.....	7
2.1 VirtualBox Installed on a host PC.....	10
2.2 Virtual Machine creation.....	11
2.3 OpenStack Object Storage Installed in a virtualized environment.....	12
2.4 VirtualBox Host-Only Networking on Windows.....	13
3.1 Architecture of the proposed scheme.....	18
3.2 Login and authentication Phase.....	23
3.3 Format of a data file stored on the cloud.....	27
3.4 Cloud Data Protection System architecture.....	32
4.1 Swift Authentication (pluggable).....	37
4.2 SwAuth Authentication System Password File Contents.....	39
4.3 Analysis of Authentication Tokens from OpenStack using Burp.....	44
4.4 Dashboard View of OpenStack Access & Security Section.....	46
5.1 OpenStack Object Storage Directory Structure.....	52

Abbreviations

ACL	Access Control List
AHL	Attribute History List
APIs	Application Programming Interfaces
VPN	Virtual Private Network
CDPS	Cloud Data Protection System
CNF	Conjunctive Normal Form
CPU	Central Processing Unit
CSA	Cloud Security Alliance
CSP	Cloud Service Provider
DaaS	Data as a Service
DIFC	Decentralized Information Flow Control
DEK	Data Encryption Key
DoS	Denial of Service
ENISA	European Network and Information Security Agency
FIPS	Federal Information Processing Standards
HTTP	Hypertext Transfer Protocol
IBM	International Business Machine
IDC	International Data Cooperation
IaaS	Infrastructure as a Service
IAM	Authentication and Identity Management

IDM	Identity Management
I/O	Input-Output
JSON	JavaScript Object Notation
MK	Master Key
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform as a Service
PK	Public Key
PW	Password
ReST	Representational State Transfer
RSA	Rivest, Shamir, and Adelman
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SK	Secret Key
SLA	Service Level Agreement
SPML	Service Provisioning Markup Language
SQL	Structured Query Language
TLS	Transport Layer Security
UI	User Interface
UUID	Universally Unique Identifier
VM	Virtual Machine
XACML	eXtensible Access Control Markup Language

CHAPTER ONE

1.1 Introduction

The term Cloud computing refers to the delivery of computing as a service rather than a product. I.e providing resources, software, and information to computers and other devices as service over a network (typically the Internet) as shown in figure 1 below. Cloud computing provides services without requiring users to know the location and details of its operation[1]. Furthermore, it is a design concept which tries to separate the application from the operating system on which the hardware runs.

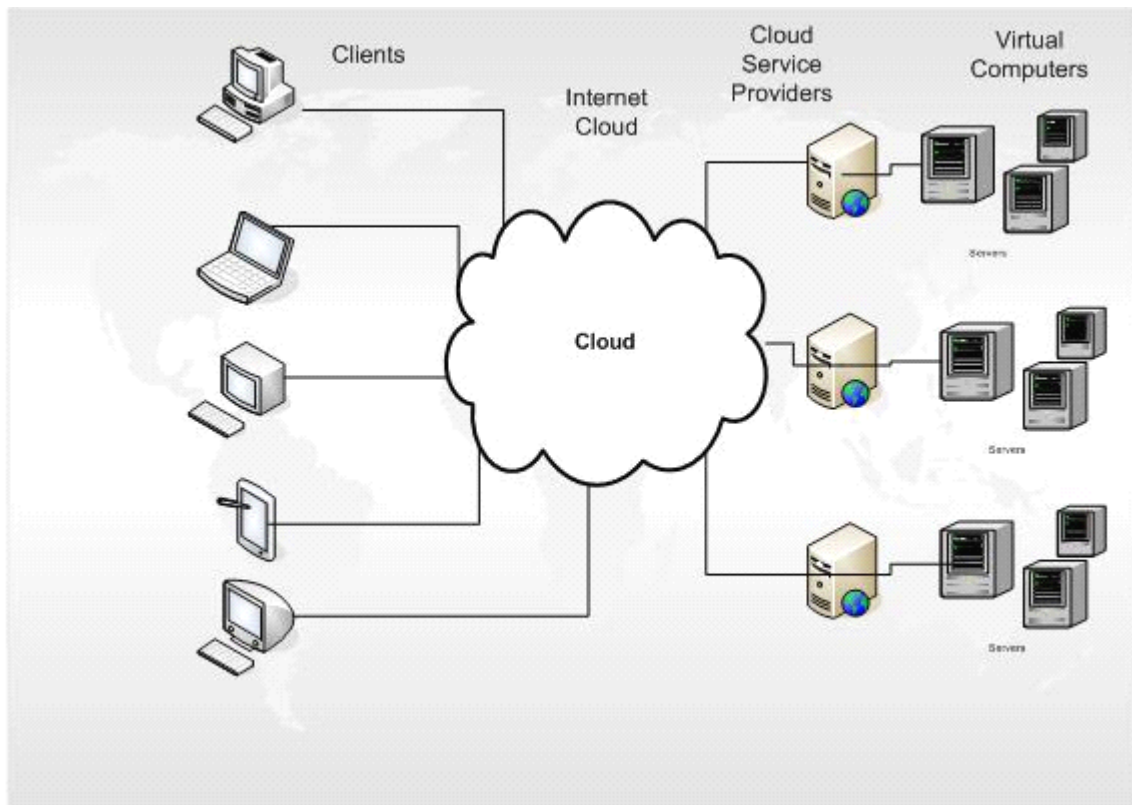


Fig: 1.1 Cloud Computing logical Diagram [2]

Due to its high scalable nature, it can provide infinite computing resources on demand, which remove lots of burden on cloud service providers when it comes to hardware provisioning. Since an up-front commitment can be eliminated, smaller

cloud providers are able to provide services to companies and can also increase their hardware resources only when there is high demand. During usage of cloud services, users are charged based on short-term basis. For instance a user can be charged for storage service on a daily base. Because of the scalability of Cloud in providing services, users can benefit from different services (e.g. Data as a Service (DaaS), Software as a Service (SaaS) or Platform as a Service (PaaS). Therefore, one can say that Cloud Computing has evolved at an incredible pace.

In the last few years, cloud computing has grown from being a promising business concept to one of the fastest growing segments of the Information Technology (IT) industry. Big companies like Amazon, Google, Windows Azure, and OpenStack has expanded their market by adopting Cloud Computing which has enabled them to provide for large number of users.

However, security and privacy issues present a strong barrier for users to adapt to this new form of IT. According to an International Data Cooperation (IDC) survey in August 2008, security was regarded as the top challenge in cloud computing [3]. Security is one of the top concerns, says users of cloud computing who fear that their business information and critical IT resources in the Cloud are vulnerable to attacks. Furthermore, cloud computing became a hot topic at the RSA security conference in San Francisco in April 2009, where Cisco CEO Chambers said that Cloud computing was inevitable, but that it would shake up the way that networks are secured[4].

Most cloud security problems arise because of lack of control, lack of trust mechanisms, multitenancy etc. These problems exist mainly in third party management models and also self-managed cloud platforms. Security is very difficult to implement in cloud computing because of the different forms of attacks in both the application side and in the hardware components. In fact, some attacks with catastrophic effects only need one security flaw.

When it comes to privacy, its concept varies widely among countries, cultures, and jurisdictions. Privacy rights or obligations are related to the collection, use, disclosure, storage, and destruction of personal data. But in the end the general idea about privacy lies in how organizations account for user's data, as well as the transparency to an organization's practice around personal information.

In this research, an investigation has been made on the security features of cloud platforms using OpenStack as a case study. The goal of this thesis was to identify security weakness related to Authentication and Identity Management (IAM), and Data Management. Base on the findings, different recommendations have been provided on how to address these problems. These Recommendations if implemented, will lead to trust in cloud computing systems which in turn would encourage companies to adopt cloud computing.

1.2 Prior Research on the Topic

The evolution of this new form of computing has become a very popular research area, because it poses some risk in its operation. One important risk factor is that of Security and privacy. And this needs to be corrected as early as possible. The following enhances the variety of researches conducted in the area which have been considered in this thesis.

Since 2009, there has been a number of articles and papers on cloud computing with high focused on its security implementations. Cloud computing is a very complex system which needs to be understood properly. The NIST defines Cloud Computing as:

“A model for enabling convenient, on-demand network access to a shared pool of configured computing resources (e.g., networks, servers, storage, applications, and services as shown in fig1.1) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential-characteristics, three

service models, and four deployment models.” [5].

Base on the above definition, since the aim of cloud computing is sharing of computing resources (networks, servers, storage, application, and services), this raises lot of concerns for the security and privacy risks involved in such computing. In a final report on the topic “Understanding the Security, Privacy and Trust Challenges” [6], the Security, Privacy and Trust challenges in cloud computing have been identified accordingly. Other research papers have also highlighted these problems in the areas of Authentication and Identity Management, Access Control, Trust Management and Policy Integration, Secure-Service Management, Privacy and Data Protection [7]. With respect to this research area, special interest have been placed on Authentication and Identity management, and that of Privacy and Data protection. In a Thesis Presented to the Faculty of the Computer Science Department of Middlebury College by Bevan Barton, the issue of Cloud Infrastructure Vulnerabilities which presents security issues at the network, host, and application levels was also mentioned [8]. His thesis points out the difficulty of applying encryption methods in the cloud. Of course this is another important factor to look at, since most cloud platforms are yet to provide better encryption facilities for user data.

In a specialization project report presented to the University of Agder, analyzes and comparison of security features and solution provided by different cloud platforms; OpenStack, Amazon and Windows Azure, was presented. Based on the theoretical findings in that research, It was realized that out of the three, OpenStack has more weaknesses than the remaining two especially in the aspect of data protection[21]. This observation presented more reasons to make an in-depth security scrutiny on OpenStack Platform. Also, in Primož Cigoj’s masters seminar [9], security issues in OpenStack where analyzed. His argument was that there are some underlying security issues in the cloud and he pointed this out using OpenStack as case study in his work.

Over the years, solutions to these problems have been suggested at conferences and also written in research papers and documents. The notion of public auditability has recently been proposed in order to ensure that the integrity of remotely stored data under different security models is maintained [10], [11], [12], [13]. This will allow external party, in addition to the user himself, to verify the correctness of remotely stored data. However, most of these schemes [10], [11], [12], do not support the privacy protection of users' data against external auditors, i.e., they may potentially reveal users' information to the auditors, as it will be discussed in this paper. This severe drawback greatly affects the security of these protocols in Cloud Computing. In as much as providing auditing mechanisms is good, it is also important that vulnerabilities leading to unauthorized information leakage is not introduced into cloud systems[14].

However, there are legal regulations, such as the US Health Insurance Portability and Accountability Act (HIPAA) [15], which demands that outsourced data must not to be leaked to external parties. Applying data encryption before outsourcing data is one way to mitigate privacy problems [13].

1.3 Overview of OpenStack

OpenStack is an open source software which anyone can use to build both private and public cloud[16]. The aim of OpenStack is "to produce the ubiquitous open source cloud computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable"[17].

The Platform was founded by Rackspace and that of NASA. It all started when NASA was trying to build a private cloud on top of Eucalyptus, but encountered a problem since Eucalyptus could not be scaled and open as NASA hoped it should be [19][18]. As a result, NASA decided to build a cloud controller from scratch and

started a project call “NOVA” [19]. While NASA was making its project to be open sourced, Rackspace was about to do the same with both its compute engine and Storage controller. This made the two to collaborate and ended up creating an open source cloud software [18]. OpenStack was first launched in July 2010. Later in October of the same year, the first project(Austin) was released and made available to the public [20]. The platform has made several releases since then. The following are the different releases and dates of availability [20]:

- | | |
|--|--|
| I. Austin—21 st October 2010. | II. Bexar—03 rd February 2011 |
| III. Cactus—15 th April 2011 | IV. Diablo—22 nd September 2011 |
| V. Essex—5 th April 2012 | VI. Folsom—27 th September 2012 |
| VII. Grizzly—4 th April 2012 | |

1.4 OpenStack Projects

I) OpenStack Compute: It is also known for its code name as “Nova”. It’s from NASA’s compute files and it provides and manages large networks, users and projects [21]. The aim of Nova is to be hardware and hypervisor agnostic. Currently, it supports different virtualization technologies such as; Citrix XenServer, UML, Microsoft Hyper-V, Xen, KVM, QEMU, VMWare and LXC Containers [22].

II) OpenStack Object Storage: This component is also known for its code name “Swift”. It’s also from Rackspace’s “Cloud Files” and provides storage system for large amounts of static data by the use of clusters of standardized servers [21]. It provides a ReSTful API for uploading and downloading files into the cluster. It support Language-specific libraries such as Java, C#, PHP, Python and Ruby.

III) OpenStack Image Repository: Another name for this project is Glance. It provides REST interface for querying information about virtual disk images. With glance, clients can register and retrieve new virtual disk images from the

system[21]. The Image registry allows multiple formats, including the following: Raw, VHD (Hyper-V), VDI (VirtualBox), qcow2 (Qemu/KVM), VMDK (VMWare), OVF (VMWare, others) [23].

The diagram below (figure1.2) shows how the three major components are connected.

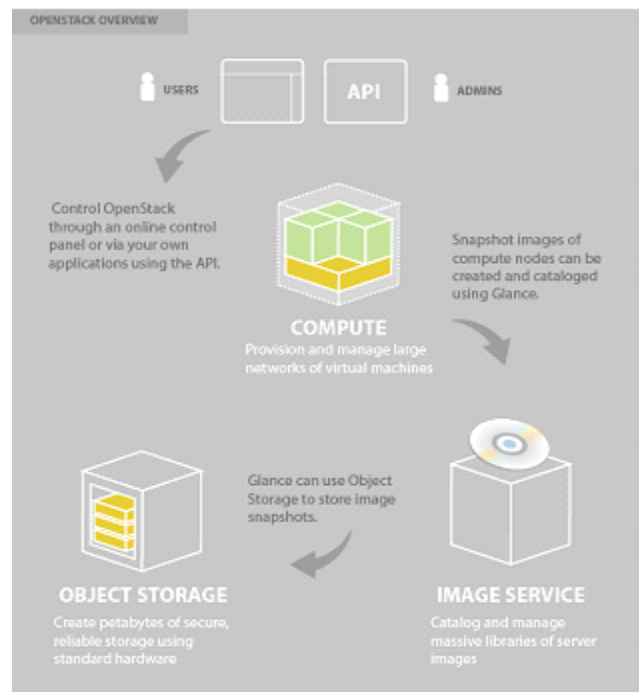


Fig 1.2 The relationship between the thee components [9]

CHAPTER TWO

2.1 Motivations

Though cloud computing is growing very fast, there are still security barriers that discourage more users from adopting this system. For two consecutive years, security was listed as the most important concern of cloud computing according to the survey conducted by International Data Corporation (IDC) [24], [25]. There was also a warning issued by the National Institute of Standards and Technology (NIST) that "security challenges [that] cloud computing presents are formidable", [26]. Researchers from Massachusetts Institute of Technology (MIT) also stated that securing cloud computing is the "information technology's next grand challenge" [27]. These concerns on security issues in cloud computing serves as the primary motivation behind this research.

When considering whether to adopt cloud usage or not, the first question most potential customers ask is 'How secure will our data be in the hands of the service provider?'. Customers have right to ask such question and also to closely examine a vendor's security credentials before making any decision. Any incident pertaining to privacy or security - even a minor one is capable of eroding brand equity and consumer trust.

Furthermore, the privacy and security capabilities possessed by different cloud platforms will affect their acceptance and how innovative applications are developed. For instance, users who critically needs privacy may favor service providers who honestly and seriously undertake to deliver specific levels of security and privacy as part of a bundle of services. Hence it is very important to have more research on the security issues that affects the adoption of cloud computing.

2.2 Objectives and Limitations

The main objective of this research is to analyze how various security issues relevant to cloud computing are been handled in cloud platforms with reference to

OpenStack. It should be noted that what has been considered here for this research is only the Object Storage component, since OpenStack consists of many different projects. The security issues related to other OpenStack projects were not considered in this thesis. Also when analyzing the security issues of OpenStack, two major security areas was investigated; Identity and Access Management, and Data Management.

In identifying cloud computing security issues in general, different cloud computing security-related documents created by individual researchers and organization have been analyzed and compared, with the findings presented accordingly. Only the areas of Identity and Access Management, Access Control, and that of Privacy and Data protection was studied.

2.3 Methodology

The approach towards this research involves two phases;

- i) Theoretical research on security issues of cloud computing
- ii) Experimental investigation on the security issues/weaknesses of OpenStack.

i) Theoretical Research on Security Issues of Cloud Computing and its Platforms

This part of the research investigates the security issues of cloud computing in general. By studying different documents which aim at helping companies and organizations to provide better cloud security solutions for their customers, the issues in certain security features like Authentication and Identity Management, Access Control, and that of Privacy and Data Protection was analyzed. These analysis have been presented in the third chapter.

ii) Experimental Investigation on Security Issues/Weaknesses of OpenStack.

In this phase, the first approach was to Installed the open source software (Open-

Stack) using the configuration files in appendix B. Next, an investigation was then carried in order to obtain the issues related to the Identity and Authentication Method, and that of Privacy and Data Protection in OpenStack Object Storage.

2.4 Deployment of OpenStack

In the experimental setup, only the OpenStack Object Storage was used in order to identify the security issues related to this platform. The software and devices used in performing the experiment were:

- i. Host Computer with 8GB RAM, at least 30GB free disk space, internet connection, VT-X enabled in BIOS. The host Operating System being windows(can also be Linux or Mac which can support VirtualBox).
- ii. VirtualBox 4 : 4.1.12-dfsg-2 (from “universe”).
- iii. Ubuntu 12.04 LTS *Precise Pangolin* ISO image

The First step in the experimental setup was to install virtual box and then configuring it. This gave a complete multi-node cluster which can be accessed and managed from the computer running VirtualBox. Figure 2.1 below shows an illustration of the deployment of VirtualBox on a Host PC.

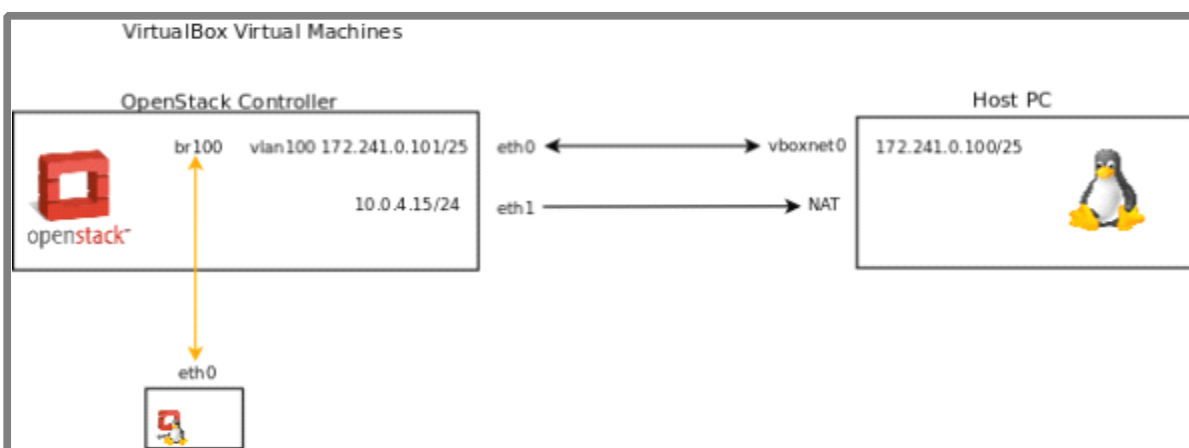


Fig 2.1: VirtualBox installed on a host PC [28]

The next step was to create different virtual machines, configuring and powering them. All of this is performed in the virtualBox which was created initially. This is shown in figure 2.2. These VMs further require the installation of ubuntu as their operating system [29].

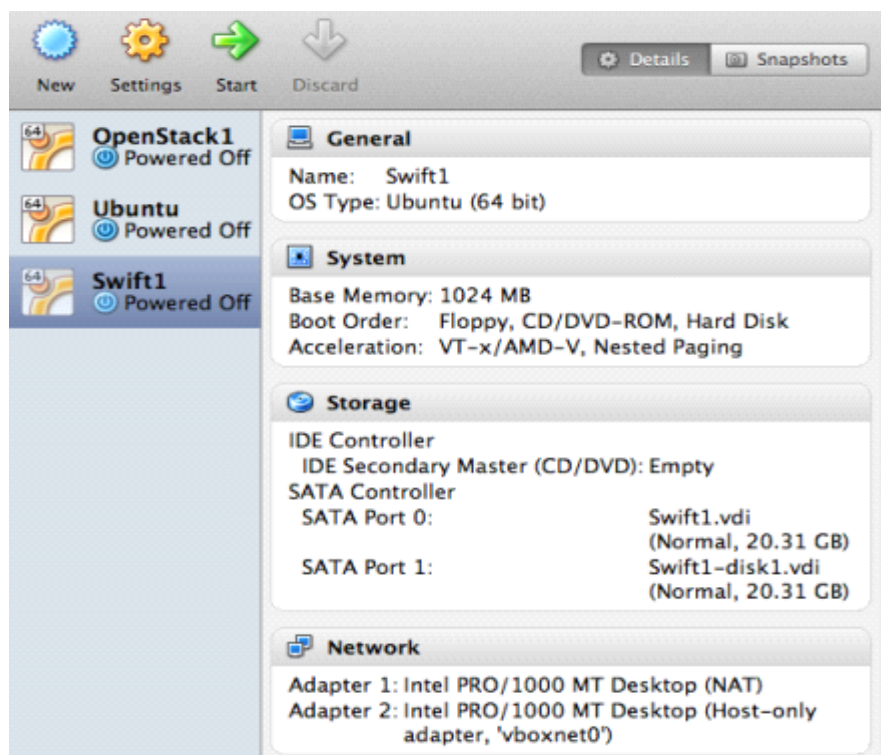


Fig 2.2: Virtual machine creation

Afterwards, with the creation and configuration of the Virtual Machines, the OpenStack Object Storage software also known as swift, was then installed on windows host machine with the help of the virtualBox [29]. In the deployment of OpenStack Swift, the procedure described in "OpenStack Cloud Computing Cook Book" [30] was followed. In order to achieve better efficiency, only three zones were created. Also, three storage nodes were then made within these zones and connected to the windows host machine through the virtual box as shown in figure 2.3 below.

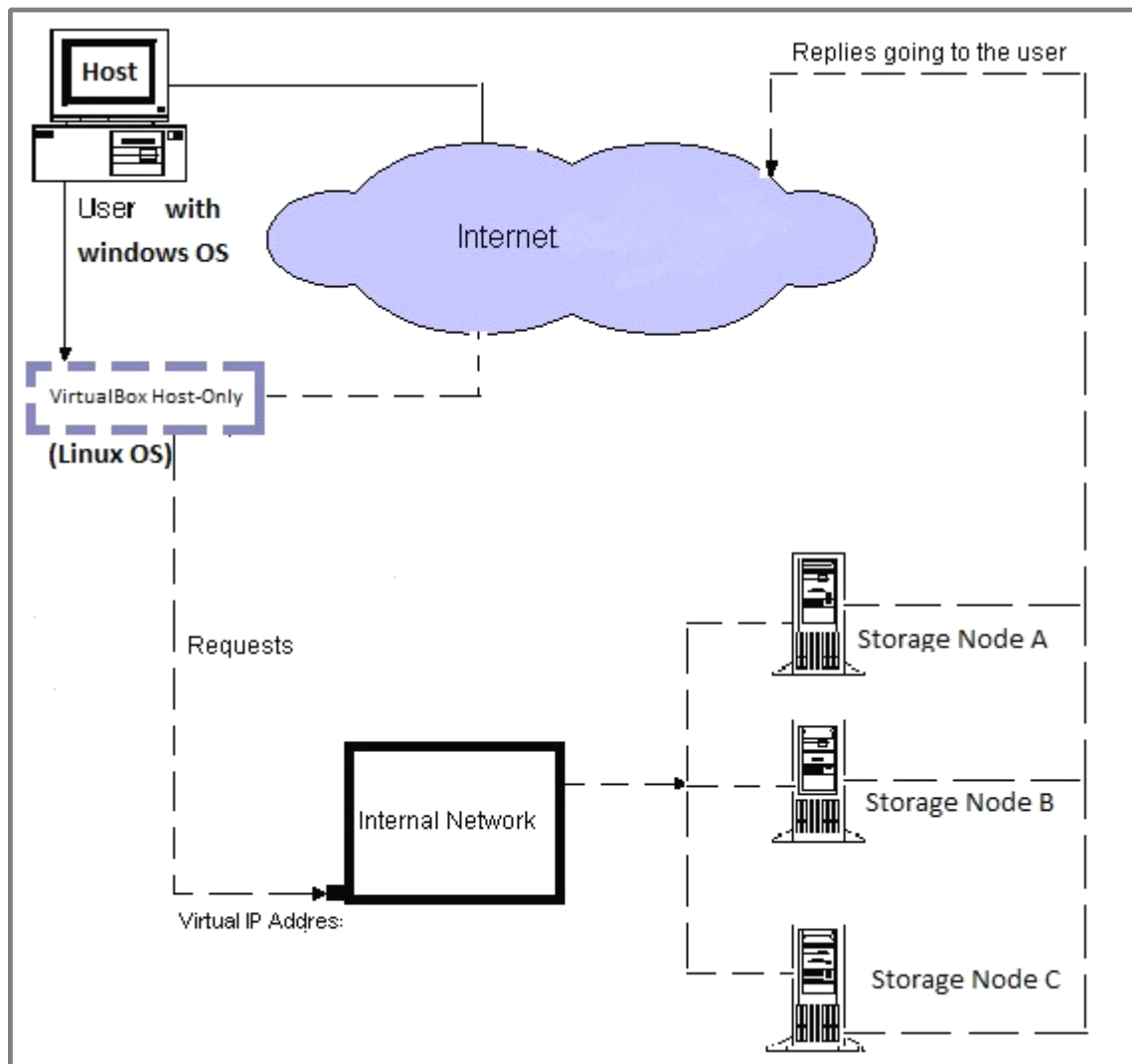


Figure 2.3: OpenStack Object Storage Installed in a virtualized environment

From figure 2.3, the first step in operation is to connect the host machine directly to the internet. The VirtualBox should also be connected to the internet but virtually (using virtual IP addressing method). In the next step of operation, the host machine(User) makes request through the VirtualBox and this request is then forwarded to the OpenStack Object Storage. The Object Storage will then verify the user's credentials and if it proves to be correct, the host machine(user) will be granted access to the the node(s) containing its own data. The host machine or VirtualBox can only access these data via internet connection as shown in the figure 2.3.

For you to be able to connect from the outer world and reach the storage nodes, it is important to select the right networking mode. Because in a real world situation, only proxy node should be able to communicate with storage nodes.

This is shown in figure 2.4.

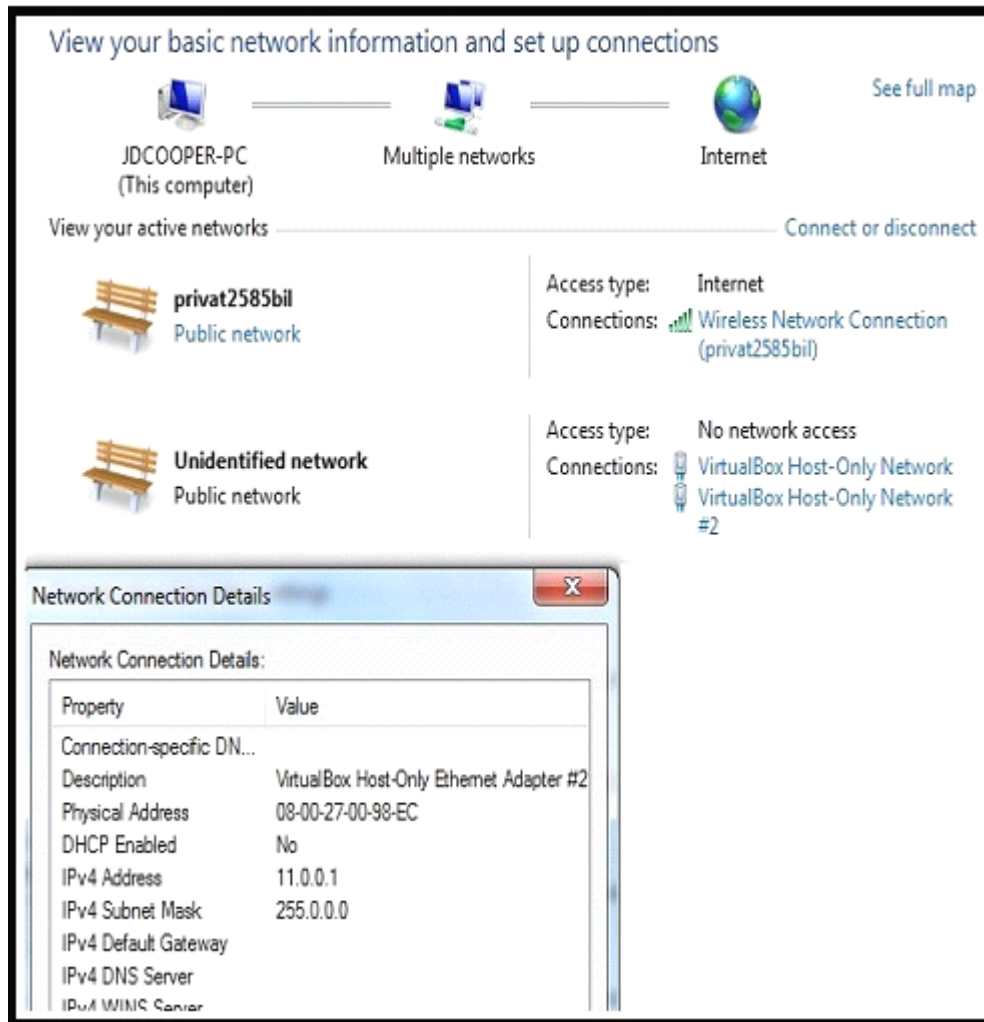


Figure 2.4: VirtualBox Host-Only networking on windows

Finally, In order to observe and make good security analysis of OpenStack, the network traffic should be intercepted between the proxy machine and that of the storage nodes (A, B and C). This was done with the help of the VirtualBox Host-Only Ethernet Adapter. With the use of a Wire-shark Protocol Analyzer, network packets were captured, observed and analyzed.

CHAPTER THREE

This chapter outlines security issues in general that are related to both public and private cloud platforms. Analysis has been made on the areas of Authentication and Identity Management, Access Control, and that of Privacy and Data Protection. This chapter also recommends solutions to these problems, as way of improving cloud operation.

3.1 Security Issues From Public and Private Cloud Service Providers

In order to succeed in cloud computing, it is very important to understand the security and privacy risks involved. In as much as the cloud provides so many good services like avoiding start-up costs, reducing operating costs, and infrastructural resources when needed, they also raise a lot of security and privacy concerns. Making data available on demand is quite important, but there comes in the challenge in ensuring that only authorized persons or certain group are given rights to access it.

Using cloud becomes even more critical since the usage of data and its decisions are made by third parties. As at now, it is actually impossible to guarantee that user's data cannot be misused by the cloud providers themselves, hence the very reason why both technical and nontechnical solutions should be applied in order to resolve the problems of the Cloud.

The following sub-sections outline the technological challenges in the cloud and its associated services.

3.1.1 Authentication and Identity Management

The use of cloud services, makes it possible for users to easily access their personal information and other useful services across the Internet. But for this to happen,

users have to provide their credentials through an identity management (IDM) system [31].

As authentication is done for users, there could be issues of interoperability involved due to the usage of different identity tokens and identity negotiation protocols within the system. An IDM system should be made in such a way that it must be able to protect private and sensitive information. Unfortunately, there are certainly some limitations and risks associated with the password-based authentication system which is currently employed. Also, it is very important to understand how the privacy of identity information can be affected by multi-tenant cloud operations. Multi-jurisdictional issues are capable of complicating protection measures [32]. In other words, user's identity and credentials must be protected whenever these users interact with a front-end service [31], [33]. These challenges can specifically be seen when using cloud as SaaS, PaaS and IaaS, which have been mentioned below[34];

i) Software as a Service

For years now, there has been a constant evolution of complex SaaS but on the other hand, there is a limited proprietary associated with it. Worst of it is that, these complex SaaS have grown faster than their standards. For instance, for three to four years now, standards such as Service Provisioning Markup Language (SPML) is still yet to be updated, but the rapid development of SaaS has created lots of ways of provisioning and managing user profiles. It has even been noticed that leading cloud service providers in SaaS chooses not to abide and work inline with the new emerging set of standards. In cases where such has been utilized, there are still insufficient standards in the areas of provisioning and management [35].

Also when attending to request made by individual users, SPML providers seem to lack authoritative third party source and this poses a very big challenge.

ii) Platform as a Service

PaaS challenges are similar to that of those in SaaS, in terms of Application Programming Interfaces (API) developments. Currently, the APIs that support provisioning of PaaS platforms are lacking. Most PaaS providers only offer simple web forms to create user accounts and profiles [34].

iii) Infrastructure as a Service

In IaaS, it involves provisioning of VPN gateways, hosts, and applications for individuals and business users. It is a requirement that Organizations should follow the proprietary mechanisms when dealing with identity management within the IaaS clouds [34].

3.1.2 Access Control

Providing total access control to data in cloud computing is very challenging and even more difficult to implement when outsourcing sensitive data outside of the same trust domain.

Applying cryptographic methods and sharing data encryption keys can help maintain the confidentiality of sensitive user data against untrusted servers. However, these solutions comes with lots of workload, and cannot be totally depended on as the only solution.

Access control services should be made flexible enough when enforcing the principle of least privilege and it should integrate privacy-protection requirements by use of complex rules. It's vital to make these rules easy to understand and very efficient. Also the system must be made in such a way that issues related to cross-domain access are addressed[35].

3.1.3 Privacy and Data Protection

Nowadays, it is possible for people to access data related to anyone from any source or location. Because of this rapid advancement in technology, we are left with the challenge of “privacy and data protection”. The term privacy, covers the rights and obligations of individuals and organizations with respect to the collection, use, retention, disclosure, and disposal of both personal and collective information. Privacy is considered to be a risk management issue when dealing with the cloud and because of this particular reason it is essential to protect identity information, policy components and transaction histories. It's difficult for organization to trust applications on systems that reside outside of their on-premise data center when storing their data. This is because private information faces high risk of potential unauthorized access and exposure. The only way that users can be sure that their data is protected is by implementing high security measures. For instance on way of doing this is by encrypting sensitive data and granting access to only specific users [37][36]

Providing assurance and high degree of transparency is of high importance.

These days, it's important to know who created a piece of data, modified it and how it was done. This could be used for different purposes including trace back, auditing and history-based access control. Now the challenge will lie in balancing between data provenance and privacy.

3.2 Recommended Security and Privacy Approaches

As a result of the complexity of the current security and privacy issues in the cloud, researchers have been working and finding ways to come up with possible solutions to these problems. This section will focus on proposed schemes (possible solutions) which can be developed and used in order to improve the security conditions in the cloud.

3.2.1 Authentication and Identity Management Solution

A strong user authentication framework, in which a legitimate user proves his/her authenticity before gaining access into the cloud was proposed in a paper: "A Strong User Authentication Framework for Cloud Computing" [38]. The scheme uses a two-step verification and it makes use of password, smart-card and out of band (i.e. strong two-factor) authentication. The purpose of this scheme is to provides mutual authentication, identity management, session key establishment, user privacy and security against many popular attacks.

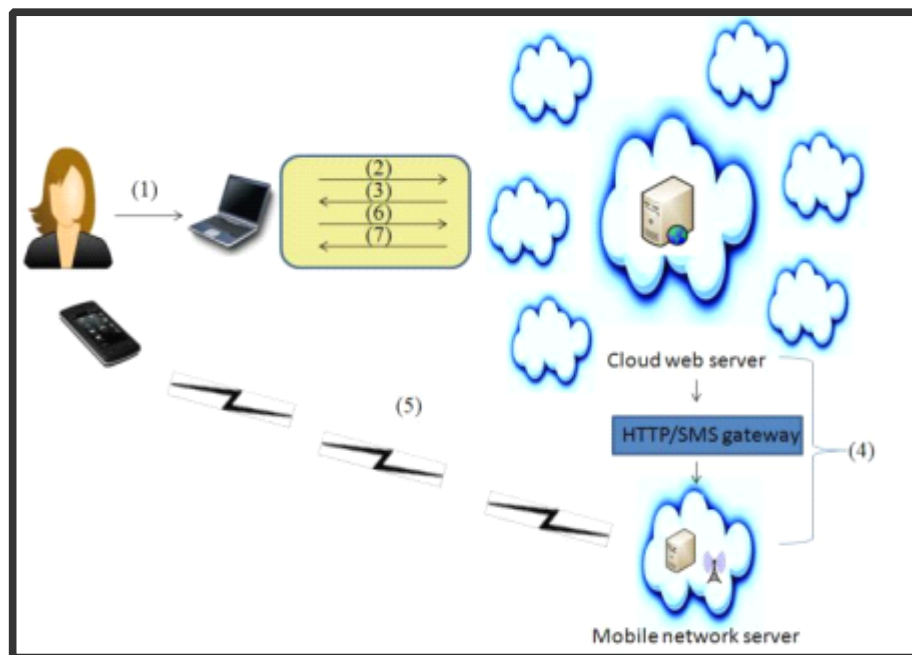


Figure 3.1 Architecture of the proposed scheme [38]

The different steps of authentication as shown in figure 3.1, has been described below[38]:

"Step 1. The user inserts the smart-card in the terminal and enter user ID and Password (PW). Next, the local system verifies the authenticity of the user based on smart-card, ID and Password.

Step 2. Once the local verification is over, the user send login request to the cloud server.

Step 3. Upon receiving the login request, the cloud server sends some authentication data based on the specific user.

Step 4. The cloud server sends the onetime key to the mobile network through HTTP/SMS gateway [42].

Step 5. The mobile network delivers the onetime key to the user via SMS.

Step 6. The user authenticates the server and sends some message based on smart-card, ID and onetime key.

Step 7. The server authenticates user based on data sent by the user in step 6"[38].

3.2.1.1 Details of Proposed Scheme

For this particular scheme, three assumptions must be taken into consideration and must not be violated. They are as follows:

1. Assume that clients and service providers are honest during registration.
2. After registration, no client and server is to be trusted. Clients should be verified during login phase by providing exact identification.
3. Once mutual authentication is attained, the server should always be trusted. It is also assumed that the server never compromises with the network adversaries.

The notations used in this scheme are mentioned below in table 4.1.

Table 3.1 Description of notations used

Notation	Description
A	Represents a user
S	Server
ID	User Identity
PW	User Password
K	Onetime Key
x	User's Secret Number
y	Server's Secret Number
P	A Large Prime Number
G	Primitive Element in the Galois field GF(p)
h(.)	One way hash function, e,g SHA1, SHA2
	Concatenation Operation
$X \rightarrow Y: M$	Message M sent, X to Y through Public Channel
$X \Rightarrow Y: M$	Message M sent, X to Y through Secure Channel
\oplus	XOR Operation

The complete proposed scheme has three phases and one activity, i.e. registration phase, login phase, authentication phase and the activity called password change.

A) Registration phase:

This phase involves registration of users. It is done at the server end, by providing appropriate identification details. At the end of registration, the user details will then be processed accordingly by the server. The procedure is as follows [38]:

1. User **A** generate a random number x and compute $h(PW \oplus x)$.
2. $A \Rightarrow S: ID, h(PW \oplus x), h(x)$
3. S checks $ID(new) = ID(existing)$. If equal, then reject registration request and go back to the step 1. Otherwise, proceed to the step 2.

4. S generates y and compute $J=h(ID \oplus h(PW \oplus x))$, $I= h(ID || y)$ and

$$B= g^{h(I || J) + h(x) + h(y)} \text{ mod } p.$$

5. S store $\{I, J, B, p, g, h(\cdot)\}$ in the smart-card.

6. Upon receiving the smart-card, the user enters x into the smart-card. Now smart card having $\{I, J, B, p, g, h(\cdot), x\}$.

7. S stores ID in the ID table maintained in the server"[38].

B) Login Phase:

This phase describes the point where the user wants to login into the clouds. Every user has to be verified before access is granted to the cloud. The procedure is described below:

"1. User **A** insert the smart-card and enter ID and PW.

2. Local system compute $J1= h (ID \oplus h (PW \oplus x))$, and check if $J1=J$ then proceed to the next step, otherwise abort.

3. Compute, $C=h (I || J)$ and $A \rightarrow S$: $M1$. User **A** sends login request message $M1$, to the server over the public channel. Here, $M1=\langle B, C \rangle$ is login request message,.

4. Subsequently, the server generate K , compute $B''=g^C+h(y)\text{mod } p$, $h(B'')$, $L=h(B'' || K)$, and $h(L)$. And the server generate message $M2= \langle h(B''), h(L) \rangle$.

5. $S \rightarrow A$: $M2$, S sends $M2$ to A using public channel. Also, $S \Rightarrow A$: K , S sends **A**,

onetime key, K using secure OOB channel to user's mobile phone.

6. Upon receiving message M_2 , user **A** computes: $B' = Bg^{-h(x)} \pmod p$ and $h(B')$ and $L^* = h(B' || K)$, and $h(L^*)$.

7. User **A** Check the two conditions, $h(B') = h(B'')$ and $h(L^*) = h(L)$, whether true or not. If both conditions are true, then proceed to the next step, otherwise terminate the login session.

8. User **A** compute $R = h(T || B')$, and generate $M_3 = \langle I, h(R), T \rangle$. $A \rightarrow S: M_3$. User **A** send message M_3 to the server over the public channel. Here T is the user's current time stamp." [38]

C) Authentication Phase:

This is the very last step in which the server decides whether user **A** should be permitted to login to the system or not. The authentication phase process is as follows.

"1. Check if $T' - T \leq \Delta T$ holds true or not. If the condition is false, then the session should be rejected. Otherwise, proceed to the next step. ΔT is the maximum legal time difference for an authentication session defined for a networking system and T' is the server's current time stamp.

2. Compute $I' = h(ID || y)$ and $R^* = h(T || B'')$.

3. Check whether $h(R^*) = h(R)$ and $I' = I$. If both conditions are true, then proceed to the next step. Otherwise, terminate the login session.

4. The server generate a session key $SK = (R \oplus L)$ and compute $h(SK)$, which is message $M_4 = \langle h(SK) \rangle$. $S \rightarrow A: M_4$ The message M_4 is sent to the user over public channel which contain the hash of a session key, i.e., $h(SK)$." [38]

The login and authentication phase is shown in figure 3.2.

For the final authentication of the user, it required of all valid users to have the session key SK for some constant definite time. The session key (SK), can be calculated by the user as $SK = (R \oplus L)$.

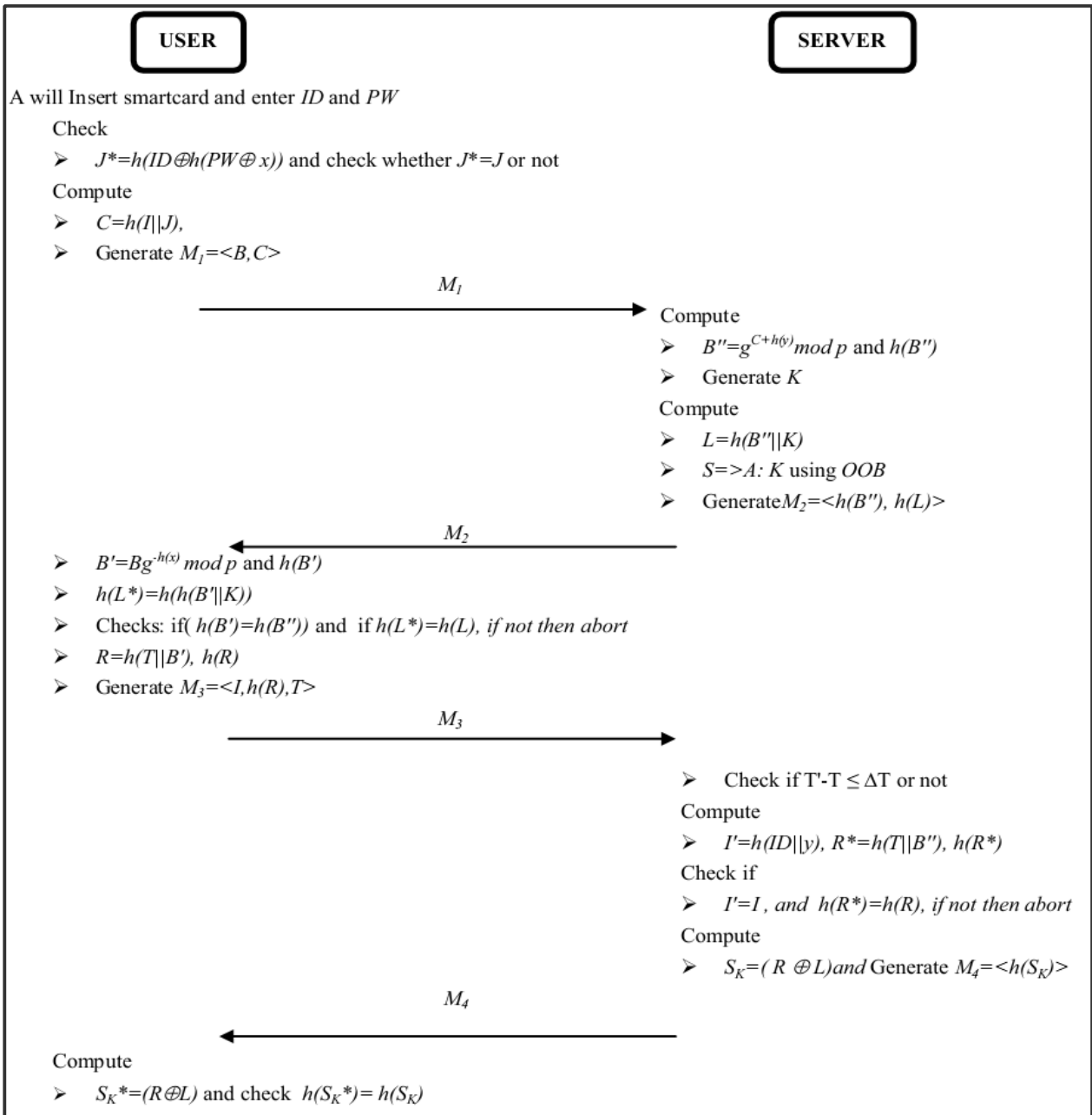


Figure 3.2: Login and authentication phase [38]

D) Password change:

This is a user friendly facility which gives users the right to change their password at anytime. It is a very important requirement in authentication schemes. The procedure for password change is given below.

1. User **A** chooses a change of password, in the self system.
2. User **A** enter ID and PW and compute $J^* = h(ID \oplus h(PW \oplus x))$.
3. Local system checks whether $J^* = J$, if yes, go to the next step, otherwise reject request.
4. User **A** enter new password, PW' and generate x' .
5. Compute, $J' = h(ID \oplus h(PW' \oplus x'))$.
6. Replace J by J' and x by x' in the smart-card." [38]

3.2.1.2 Conclusion

There are two major advantages of this proposed secure cloud architecture. They are as follows:

- 1) The scheme has an extra OOB (out of band) factor (other than only two factors) which undoubtedly provide better security over two factor authentication.
- 2) It has two separate communication channels, which makes it very difficult for an adversaries to attack.

Generally, it ensures Authentication and Identity Management by providing mutual authentication, user privacy, session key agreement and also protecting the system from so many attacks (like Replay attack, man in the middle attack, denial of service attack, stolen verifier attack and data modification attack, etc)[38].

3.2.2 Access Control Approach

A better access control system in cloud computing has been proposed by Wang, Ren and Lou in the paper "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing". It provides security, scalability and a better access control mechanism for outsourced data in the cloud. This is done by combing three different advanced cryptographic techniques: KP-ABE, PRE and lazy re-encryption [39].

3.2.2.1 Scheme Description

Firstly, it is important to note the notations used (see table 3.2).

Table 3.2 Notations used In Access Control scheme

Notation	Description
PK, MK	Public Key & Master Key respectively.
T_i	Public Key Component for Attribute i
t_i	Master Key Component for Attribute i
SK	Secret Key
sk_i	Secret Key Component for Attribute i
E_i	Ciphertext Component for Attribute i
I	Attribute Set assign to Data File
DEK	Symmetric Data Encryption Key of Data File
P	User Access Structure
L_p	Set of Attributes attached to Leaf Node of P
Att_D	Dummy Attribute
UL	System User List
AHL _i	Attribute History List for Attribute i
$rk_{i \leftrightarrow i'}$	Proxy Re-encryption key from current i to updated version i'
δ_o, X	Data Owner's signature on Message X

In this scheme, each data file is associated with a set of attributes, and each user is assigned with an expressive access structure which is defined over these attributes. It is also possible for different data files to have subset of attributes that are common. Each attribute is associated with a version number for the purpose of attribute update. Cloud Servers keep an attribute history list (AHL) which includes the evolution history of each attribute and PRE keys used. In addition, one dummy attribute (AttD) is used for key management purpose. This dummy attribute must be included in every data file's attribute set and never to be updated. The access control structure uses the tree like implementation in which the interior nodes are threshold gates and the leaf nodes are associated with data file attributes. The root node is only needed for the purpose of key management. This root node is implemented as an AND gate (i.e., **n-of-n** threshold gate) with one child being the leaf node and associated with the dummy attribute. The other child node can have any threshold gate. In addition, a user list (UL) will record all valid user IDs in the Cloud Servers [39].

The scheme is presented in two levels: System Level and Algorithm Level. The description of these two are as follows.

1. System Level Operation

The System Level operations entails, system setup, new file creation, new user grant, user revocation and file deletion. These operation have been described as follows:

System Setup

Here, the data owner first chooses a security parameter κ and calls the algorithm level interface $ASetup(\kappa)$. This will then output the system public parameter PK and the system master key MK. At this point, the owner will have to sign each component of PK and then send this PK along with these signatures to Cloud Servers[39].

New File Creation

This operation involves the creation of a file and it is processed as follows:

I. The data owner first selects a unique ID for the data file.

II. Next, a symmetric data encryption key $DEK \xleftarrow{R} \mathcal{K}$ is randomly selected where \mathcal{K} is the key space and DEK is used for encrypting the data file.

III. Finally, a set of attribute I for the data file is defined and DEK encrypted with I using KP-ABE, i.e. $(\tilde{E}, \{E_i\}_{i \in I}) \leftarrow AEncrypt(I, DEK, PK)$

In the end, each of these data files is stored on the cloud and in the format shown in Fig 3.3.



Fig. 3.3 Data File format stored in the Cloud

New User Grant

Adding a new user to the system involves two things; assigning an access structure and generating a secret key to this user. This is given below as follows:

I. All new users are assigned a unique identity w and an access structure P ,

II. A secret key SK is then generated for w , i.e., $SK \leftarrow AKeyGen(P, MK)$;

III. Encrypt the tuple $(P, SK, PK, \delta O, (P, SK, PK))$ with user w 's public key. The ciphertext is denoted as C ;

IV. The tuple $(T, C, \delta O, (T, C))$ is sent to Cloud Servers, where T denotes the tuple $(w, \{j, sk_j\}_{j \in LP \setminus AttD})$. When this tuple $(T, C, \delta O, (T, C))$ is received, the Cloud Servers will then process it as follows:

V. verify $\delta O, (T, C)$ and proceed if correct;

- VI. Store T in the system user list (UL);
- VII. Forward C to the user."[39]

When C is received, it is first decrypted using the user's private key. Next, the signature δO , (P, SK, PK) has to be verified. If correct, the user will then accept (P, SK, PK) as his access structure, secret key and the system public key respectively.

Because the secret key components of SK are stored in the cloud servers except for the dummy attribute $AttD$, cloud servers can actually be able to update these secret key components during user revocation. Now, since one undisclosed secret key component (the one for $AttD$) still exists, it is impossible for the cloud servers to use the known ones to correctly decrypt the ciphertexts. Literally, an unauthorized user cannot decrypt the ciphertext even if he gets to know the disclosed secret key components.

User Revocation

Before a user is revoked, the owner of the data will first find out the minimal set of attributes. After which, he will update these attributes. This is done by redefining their system master key components. The Public key components of these attributes in PK are redefined accordingly. Next, all user secret keys will be updated except for the one which is to be revoked. Lastly, in the affected files, the DEKs are re-encrypted using the recent version of PK .

There is a lot of computational work to be done in order to undertake this action, but one way of resolving this is by combining proxy re-encryption with KP-ABE technique, and then by using Cloud Servers to do data file re-encryption and user secret key update.

File Deletion

For this operation to take place, only the data owner has to make the request. If at all there is need to delete a file, the file's unique ID is sent along with the data owner's signature on this ID to Cloud Servers. When this signature has been proven to be correct, the Cloud Servers can then go further to delete the data file.

2. Algorithm Level Operations

This level has eight algorithms: ASetup, AEncrypt, AKeyGen, ADecrypt, AUpdateAtt, AUpdateSK, AUpdateAtt4File and AMinimalSet. The first four are actually the same as Setup, Encryption, Key Generation, and Decryption of the standard KP-ABE respectively. Much attention is paid on the last four during the implementation.

AUpdateAtt: this algorithm is responsible for updating attributes to a newer version. This is done by redefining its system master key and public key component. With the use of both the old and new attribute versions, the algorithm can also outputs a proxy re-encryption key. The algorithm used for this operation is given as:

AUpdateAtt (i, MK)

Randomly pick $t_i' \xleftarrow{R} \mathbb{Z}_p$;

Compute $T_i' \leftarrow g^{t_i'}$, and $rk_{i \leftrightarrow i'} \leftarrow \frac{t_i'}{t_i}$;

Output t_i' , T_i' , and $rk_{i \leftrightarrow i'}$.

AUpdateAtt4File: Using this algorithm, the ciphertext component of an attribute i of a file is translated from old to latest version. This is done by first checking the attribute history list(AHL) and then locating the old version. Next, by using the old and latest versions, all the PRE keys are multiplied in order to obtain a single PRE key. This single PRE key is finally applied to the ciphertext component E_i and returns $E_i^{(n)}$. This coincides with the latest definition of attribute i . The algorithm used for this operation is given as:

AUpdateAtt4File(i, E_i, AHL_i)

If i has the latest version, exit;

Search AHL_i and locate the old version of i ;

// assume the latest definition if i MK is $t_{i(n)}$.

$$r k_{i \leftrightarrow i(n)} \leftarrow r k_{i \leftrightarrow i} \cdot r k_{i \leftrightarrow i} \dots r k_{i \leftrightarrow i}^{(n-1)} \leftrightarrow i^{(n)} = \frac{t_i^{(n)}}{t_i} ;$$

Compute $E_i^n \leftarrow (E_i)^{r k_{i \leftrightarrow i(n)}} = g^{t_i^{(n)} s}$;

Output E_i^n .

AUpdateSK: This algorithm is responsible for the translation of the secret key component with attribute i in the user secret key SK from old version to the latest version. This implementation is similar to AUpdateAtt4File with the exception that, the last step applies $(r k_{i \leftrightarrow i(n)})^{-1}$ to SK_i instead of $r k_{i \leftrightarrow i(n)}$. This is as a result of the denominator of the exponent part of SK_i while in E_i it is a numerator.

AMinimalSet: This algorithm is responsible for determining a minimal set of attributes needed for the condition of an access tree to hold. This is done by

constructing the conjunctive normal form (CNF) of the access tree, and then returning the attributes in the shortest clause of the CNF formula as the minimal attribute set.

3.2.2.2 Conclusion of proposed scheme

If applied correctly, the scheme can address the following :

I)Fine-grained Access Control; it is possible to get a well defined and expressive, and flexible access structure.

II)User Access Privilege confidentiality; only the leaf node information of a user's access tree is disclosed to cloud servers, which makes it hard for cloud servers to recover the access structure. And thus user access privilege confidentiality is ensured.

III)User Secret Key Accountability; By using the enhanced construction of KP-ABE , this property can be achieved[56]. This can then be used to disclose the identities of key abusers.

IV)Data confidentiality; This scheme is secured enough to protect the system even if there is collusion attacks between Cloud Servers and malicious users.

3.2.3 Privacy and Data Protection solution

In 2009, the International Business Machine Corporation (IBM) proposed a solution on privacy and data protection, in which they developed a fully homomorphic encryption scheme. This scheme allows data to be processed without being decrypted [40].

Notwithstanding that, a Cloud Data Protection System (CDPS) has also been proposed in another paper[42]. This scheme is described in the following sections.

3.2.3.1 Architecture of the Scheme

Figure 3.4 shows the architecture of Cloud Data Protection System (CDPS) in which the upper part determines a composition of encryption algorithm and

division numbers which is used to protect users' data. The lower half protect the flow of data by means of selective security composition and Data Division mechanisms.

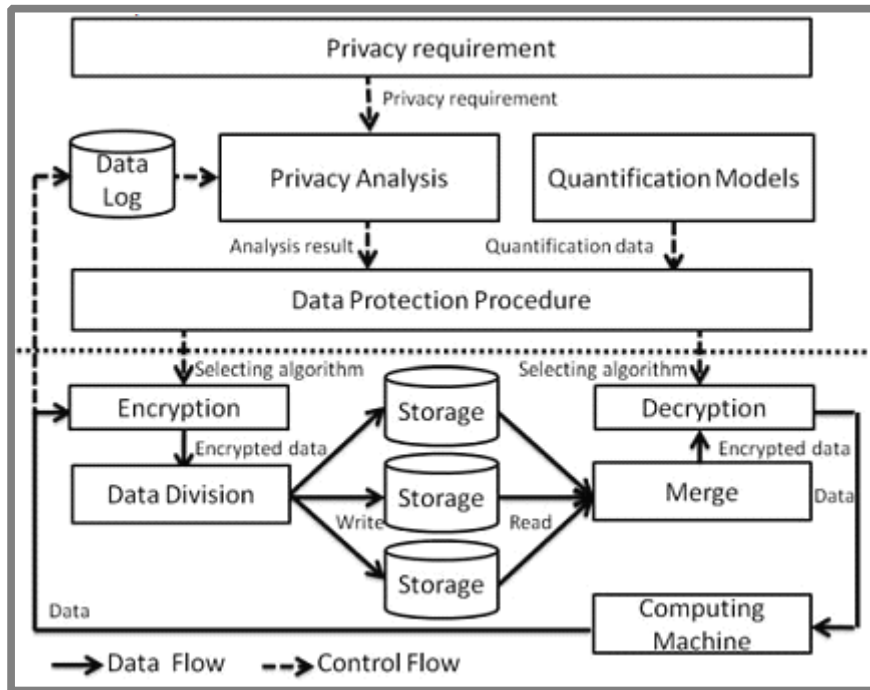


Fig 3.4 Cloud data protection system architecture [42]

Furthermore, the diagram describes the privacy requirement and frequency of the key used to encrypt data. The quantification model is used to achieve both the security and algorithm execution speed. Basically, when executing encryption algorithm, a measure of the cracking year of each encryption algorithm used by CDPS is taken into account. Also the speed quantification mechanism will measure the mega clock cycles per megabyte when executing each encryption algorithm in specific machine [42]. This scheme adds more security to the data by using data division techniques as shown in the CDPS architecture.

3.2.3.2 Privacy Requirement and Analysis

The sensitivity of data varies from one to the other and this is largely based on the degree of importance. By using encryption algorithm and data division, it is

actually possible to obtain data confidentiality [42].

Addressing the privacy requirements of user's data is also essential. This leads us to the different entities responsible for the provision of privacy protection. These entities has been described as follows:

1) Privacy Level

There are basically three privacy levels mentioned below and these are to be selected based on the requirement by each user. These different levels can be selected in line with the degree of security based on the data sensitivity. These have been divided into three levels, speed, hybrid, and security [42].

In the Speed level, the requirement is that there should be no sensitive information in the data. In order words this level deals with less sensitive data. A very weak encryption is applied in this case. In the case of the Hybrid level, some amount of sensitivity must be present in order for this level to be selected by a user. Meaning weak encryption mechanism must not be applied here. Now for the last level(Security level)to be applied, there must be high amount of sensitivity in the data. This is used only when effective protection of data is needed. In this particular case,users have to focus more on obtaining confidentiality instead of performance.

2) Key Update Frequency

After a specific privacy level have been selected, the range of security required is calculated as:

$$Security_{range} = \frac{Security_{max}}{\| P_{level} \|} \dots\dots\dots(eq. 1)$$

The maximum security which is the numerator part, is the security score obtained by the CDPS when the most strong encryption algorithm is used. This value is 100. Also, the denominator value obtained is refereed to as the privacy levels.

Another important factor which should be looked at is that of the key update frequency. This parameter tends to affect the performance overheads when data is frequently written in the cloud. This is only solved by revising the encryption algorithm based on the obtained key update frequency. In order to solve this problem, the encryption algorithm is revised according to the key update frequency since the more the encryption key is updated, the shorter the average key life cycle becomes. The frequency of the key update is thus calculated using equation 2[42].

$$Frequency_{KeyUpdate} = \frac{Write\ data}{APeriod\ \Delta t} \dots\dots\dots(eq. 2)$$

Writing the data frequently, will obviously reduce the life cycle of key. Thus, it is better to select the high performance security algorithm in order to provide better I/O performance.

Quantification Models

This model constitute two types; the Security Quantification and that of Speed Quantification. The security strength of any encryption algorithm is measured using the time taken for it to be cracked. This is normalized to map the security strength into a range between 0 and 100. Speed Quantification deals with the time taken for an encryption algorithm to be performed. In this case, the CPU consumption rate can be used to determine the encryption time.

Data Division

This is the case where a particular data is split into different segments and then stored in different location so as to improve the speed taken to execute them. This is applied after encryption operation has been done on the data.

Mathematically, it can be found that the probability of hacking all divided parts becomes smaller as the number of divided parts increases[42]. As a result, it

makes it very difficult for the attacker to obtain all of the divided encrypted parts. Hence applying this method in any cloud environment can effectively enhance data security.

3.2.3.3 Conclusion

Base on simulation work done, it shows that this scheme can effectively achieve data privacy in cloud environment. It provides confidentiality of users' data without further increase in system performance overhead. Hence when compared to other security schemes, the improved performances is up to 50% and at least 35%[42].

CHAPTER FOUR

Identity and Access management

This chapter looks at identity and access management(IAM) implementation in OpenStack Object Storage (swift). Upon conducting a security investigation, different issues related to IAM in swift were found out.

The following sections in this chapter provides description of IAM in Openstack Object storage and the issues associated with it.

4.1 User Identity Provisioning/De-provisioning

4.1.1 Overview

The process of creating users, registering users, and managing access to resources in a system is know as user provisioning. The opposite operation is called de-provisioning. It is very important to automate user management tasks. In OpenStack, the authentication/authorization system is pluggable and thus, it does not depend on the project itself. Figure 4.1 describes how the swift authentication mechanism is done. Firstly, the client submit his user credentials to the public authentication system. When these credentials have been verified, he will then send a token request to the proxy server. The proxy server further ask the the private authentication for a token. A token will then be generated and sent to the client. The received token is what the client uses before it can be granted access to the storage nodes that he has permissions to.

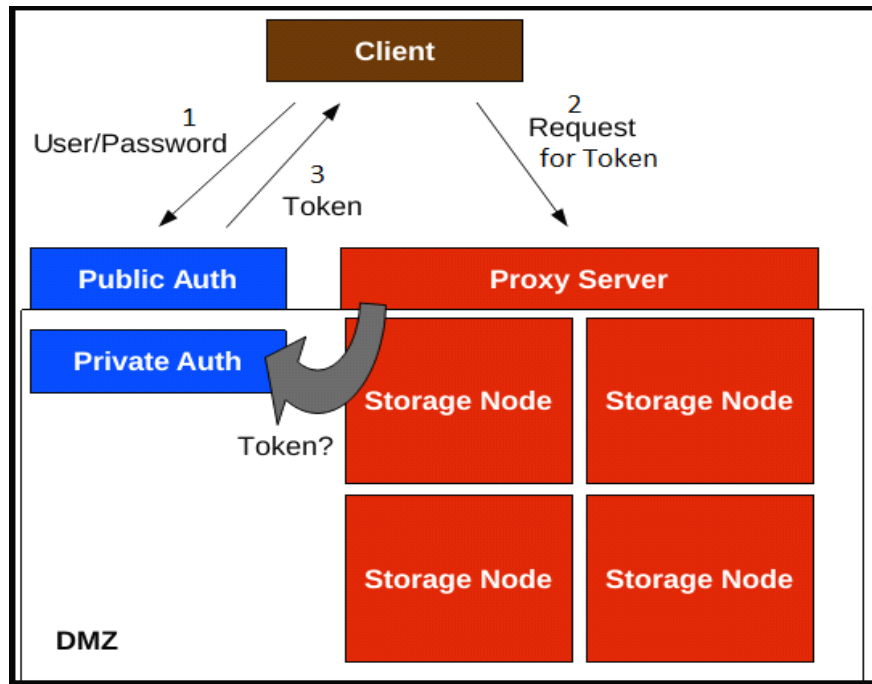


Fig 4.1 Swift authentication (pluggable) [44]

OpenStack Object Storage uses both DevAuth and SwAuth as its authentication/authorization systems [45]. The main difference between the two lies in how the data is stored. DevAuth uses SQLite database as a user data storage, while SQLite stores data in a single cross-platform file on disk [46]. One significant issue with SQLite is that it allows multiple reads but allows only a single write operation to be done to a file. This means using DevAuth in a system which will require simultaneous registration of many users can definitely cause performance slowdown. Though this problem exist, OpenStack Object Storage still uses DevAuth as its default authentication system. SwAuth seems to be a more "scalable authentication and authorization system when compared to DevAuth, and it uses Swift itself as its backing store" [47]. Swift account is created on a cluster in which the information from users are stored as JSON -encoded data text files.

User roles are based on OpenStack Object Storage and its source code. The following types of user roles exist:

1) **Users:** These are considered to be the ordinary users of the service and they are

not granted any administrative rights.

II) **Admin:** these set of people have administrative rights but they can only add users to accounts which they have rights to administrate. In SwAuth, users can be barned from administering accounts.

III) **Reseller Admin:** For these set of users, they have Administrative permissions on all of the accounts but they are not allowed to add other Reseller Admins

IV) **Super Admin:** These are the most powerful users/administrators. They can perform all user management tasks, to the extent of even adding Reseller Admins.

4.1.2 Elevation of Privileges in OpenStack Object Storage

When adding or removing users from a system, a possible arising issue is elevation of privileges. OpenStack object storage user management is role based, and also the access permissions are specified directly in the code. When ever someone tries to add another user into the system, the code will check to see if the user executing a given action is acting in a role of Super Admin or not. It is feasibly impossible to delegate the permission of adding a Reseller Admin, except if a Super Admin power is assign to the user which is to be added.

Devauth

User information are stored by Devauth in SQLite database and these files are located in etc/-swift/auth.db. In the case where a user is acting in a role of Super Admin, his password will be checked from the configuration file of the Swift authentication service. Otherwise, the user data will be retrieved from the database. It has been found that the code that connects to database might be susceptible to SQL injection. As a result of this, the source code used during installation was examined to see whether such attacks are really possible. By using the python code below, it can be verified whether users have administrative privileges or not [48]:

```
1 row=conn.execute(''  
2 SELECT reseller_admin , admin FROM account
```

```

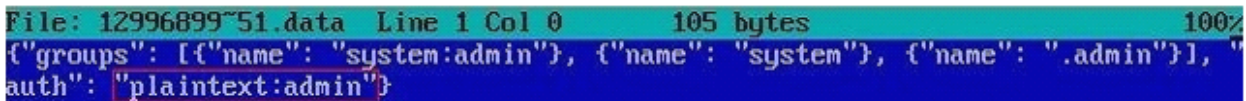
3 WHERE account = ? AND user = ? AND password = ? ' ' ,
4 (account, user, request.headers.get( ' X-Auth-Admin-Key ' ) ) )
5 .fetchone ()

```

The user data is passed to the SQL code without filtering the inputs (see `request.headers.get('X-Auth--Admin-Key')`) and this makes one to think that the code is vulnerable. However, after thorough checking, it was found that actually there are no vulnerabilities in the `sqlite3` library itself [50] and at the same time, there are no possibilities to elevate privileges using SQL injection attacks.

Swauth

The figure 4.2, shows how User information are stored by Swauth as JSON-encoded data. The user groups are stored in an array of objects with single attribute ***name*** in which the attribute's value represents the name of user groups . In the case where the user is a *Reseller Admin*, he will belong to the group ***reseller_admin*** but if he is an *Admin*, then he will belong to the group ***admin*** (the user in Figure 4.2 has Administrative privileges). Super Admin information is stored in the configuration file of OpenStack Object Storage just as in `devauth`.



```

File: 12996899~51.data Line 1 Col 0      105 bytes      100%
{"groups": [{"name": "system:admin"}, {"name": "system"}, {"name": ".admin"}], "
auth": "plaintext:admin"}

```

Fig 4.2 Swauth authentication system password file contents[48]

In order to examine whether it is possible to inject `.reseller_admin` or `.admin` values into the `groups` array, the following Python code was used to create JSON-encoded string with user data[48]:

```

1 groups = [ ' %s : % ' (account , user) , account ]
2 if admin:
3 groups . append( '.admin' )
4 if reseller_admin:

```

```
5 groups . Append( ' , reseller_admin ' )
6 json . dumps ( { ' auth ' : ' plaintext :%s ' % key , ' groups ' : [ { ' name ' : g}
    for g in groups ] } )
```

The protection against injections is found in ***json.dumps*** method. This method accepts an object and converts its properties to a string, performing proper escaping of input, which is why we could not find a way to affect the value of one property by injecting input via another property (for example, we can not affect group information by injecting data to key input). If a JSON data was created using simple string concatenation, then injection attacks would be feasible.

To conclude this section, it was found that there are no vulnerabilities in devauth and swauth systems that lead to the elevation of privileges during identity provisioning.

4.2 Identity Federation

The process of exchanging identity information between identity providers and that of service providers is termed as identity federation[51]. SAML or WS-Federation are mostly recommended as enabling technologies in Cloud computing [52]. OpenStack Object Storage does not support identity federation. But using the PySAML2 codes[53], it is feasible to incorporate PySAML2 into OpenStack in order to add Identity Federation functionality to the software, Since authentication system in OpenStack Object Storage is written as WSGI middleware itself [54].

4.3 Authentication In OpenStack Object Storage

4.3.1 Overview

Both DevAuth and SwAuth authentication systems use username and password during authentication. When a user has been successfully authenticated into the system, a token will be sent to him, which he will further use to identify himself .

Normally, the received token will last for a period of 24 hours (which in fact is the default expiration time). Also, an account URL is created upon account registration and not supposed to be ever changed [55].

Almost all documents related to cloud security make it clear on the need to allow authentication delegation by accepting confirmations in SAML format. Notwithstanding that, this SAML feature has still not yet been implemented in OpenStack.

4.3. 2 Password Strength

It is important to take a close study on the password strength requirement since both DevAuth and SwAuth in OpenStack Object Storage uses username and password combination to authenticate users. There is currently a guideline which provides rules as to how users should choice their passwords. This includes checking passwords in order to avoid possible dictionary attack, making emphases on minimal password length, and also applying a combination of different characters (lower-case, upper-case, non-alphabetic) [56]. Unfortunately no such requirements (password length and special characters) exist in OpenStack. Also there is no dictionary checks, this makes it possible for users to register with password as short as one character even.

4.3.3 Password storage

Storing passwords in a very secured way has been an issue throughout the history of information systems. In general, the golden rule in Information Security is "don't store passwords in clear-text", and this must be ensured by the administrator. It is better to use encryption methods and also very important to provide a limited access to locations of stored passwords. In order words only administrators with special access rights should be able to locate these passwords.

DevAuth stores username and password in a configuration file, located at *etc/swift/auth.db*. Interestingly, it can be noticed that passwords are stored in plain text format. An instance of stored username and password in DevAuth is

shown below:

```
Jdcooper@openstack:~/swift$ locate proxy-server.conf
/etc/swift/proxy-server.conf
/home/jdcooper/swift/doc/proxy-server.conf.4
/home/jdcooper/swift/etc/proxy-server.conf-sample
jdcooper@openstack:~?swift$ cat/etc/swift/proxy-server.conf |grep admin
operator_roles = admin
admin_token = ADMIN
#user_admin_admin = admin.admin.reseller_admin
#user_test_tester = testing.admin
#user_test2_tester = testing2.admin
jdcooper@openstack:~/swift$
```

Another problem also is that, the default settings in DevAuth actually give users the right to read from this file. This makes it possible for system users to easily obtain password and gain access to accounts of other users. Hence the reason why DevAuth is not usually considered for developing purposes. This weakness should be well noted and documented in order to warn users about it consequences . Initially during the setup, the access rights should be changed such that only system administrators must have read permissions to such sensitive file.

On the other hand, *Super User's* password is stored in */etc/swift/auth-server.conf* for DevAuth and */etc/swift/proxy-server.conf* for SwAuth. This is also done in different manner compared to how other user's passwords are stored. Only that the password in this configuration file is also stored in clear text. Which brings us back to the need of changing passwords after OpenStack installation and making sure that access rights are reviewed, such that only legitimate owners of file(s) should have read/write permissions.

Unlike DevAuth, SwAuth possesses properly configured access rights in securing password data. The only security concerns with swAuth is that passwords are also stored in clear text in the file as shown in figure 4.2 . As a result, this can lead to an insider attack, in the case where a system administrator uses UNIX superuser (root) account to find out user passwords.

In a simple conclusion, both DevAuth and SwAuth lack appropriate protection scheme for storing passwords.

4.3.4 Analysis of Authentication Tokens

Tokens are received when a user authenticate successfully into a system. And they are used inline with a service request, in which username and password are given as an input to the interface. Tokens can become invalid in two cases:

I) When they run out of time. In order words, they become expired.

II) If revoked.

It is important to note that authentication should be performed over a secure channel (TLS), otherwise, there are chances that an attacker could retrieve a user token, and use it to perform attacks (like "man in the middle" attack), and ends up impersonating the actual user who received the token from the authentication system.

However, in an experiment conducted by Rostyslav Slipetsky [48], the algorithms from token generation were first imported and examine using webscarab and later exported to Burp Sequencer in order to run randomness test on the token data. This is shown in figure 4.3.

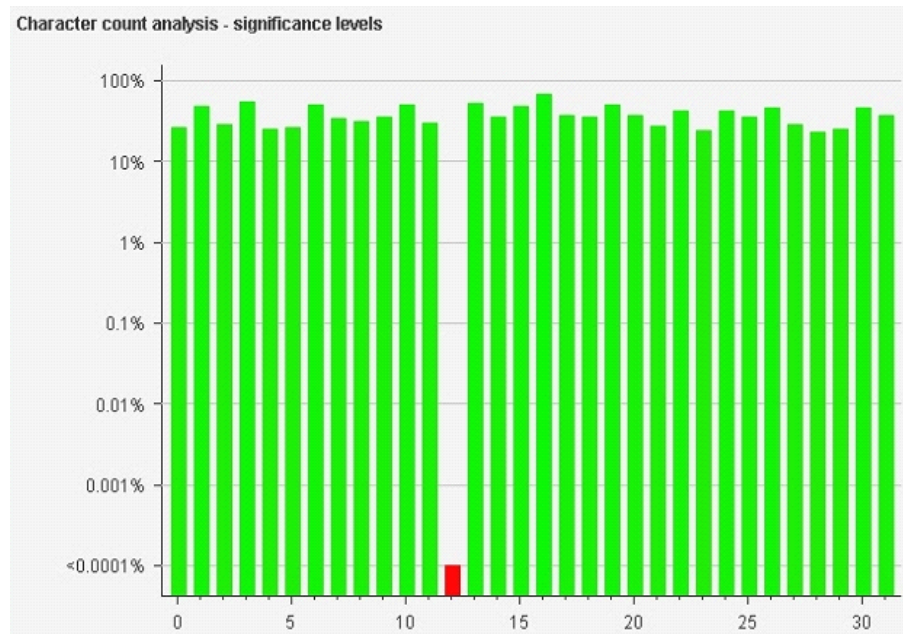


Figure 4.3: Analysis of authentication tokens from openStack using burp Sequencer[48]

According to the results obtained from the tool, the overall quality of randomness within the sample is estimated to be excellent and it was summed up that the approach taken to generate UUID uses a solid source of randomness which has no known weaknesses. With such result, one can conclude that the system is pretty much secured[48].

4.4 Authorization and Access Control

4.4.1 Overview

This section looks at the authorization procedures with regards to accounts, containers and objects. In OpenStack Object Storage, accounts are created both for single group usage. Objects in OpenStack Object Storage looks like files in traditional operating systems, and containers resemble folders (or directories).

The different actions which can be undertaken by different types of users with accounts are:

User: these kind of people have account management rights.

Admin: these types are permitted to obtain only account information (URL, users, etc.).

Reseller Admin: they have rights that can enable them to add and remove accounts. They also have Administrative permissions to get information about an account.

Super Admin : they have the same permissions just as Reseller Admin.

And the actions that users in different roles can perform with containers are:

User: For this type, they can only access containers which they have permissions to based on the access control list(ACL).

Admin: This type of users are permitted to undertake all operations within there own account (add/remove containers, upload/download objects, etc.)

Reseller Admin: For this type, they have permissions to all accounts

Super Admin: These users have no permissions to undertake container/object management. Figure 4.4 shows the dashboard view of the admin(access control and security) section of OpenStack.

In the authorization system, it is not possible to make permissions at the object level, but rather at the container level. Users are granted access right only by the administrator and it depends on the access control lists (ACL).

Specific separate ACLs for read and write operations can be made. Each item in ACL can either grant or deny access based on any of the following parameters:

- Referrer designation based on HTTP Referrer header.
- All users of a specific account.
- Specific user of a specific account.

Security documents recommend the use of eXtensible Access Control Markup Language (XACML) to control access to cloud resources [58], but unfortunately, OpenStack Object Storage does not use any of them since it relies on a proprietary solution.

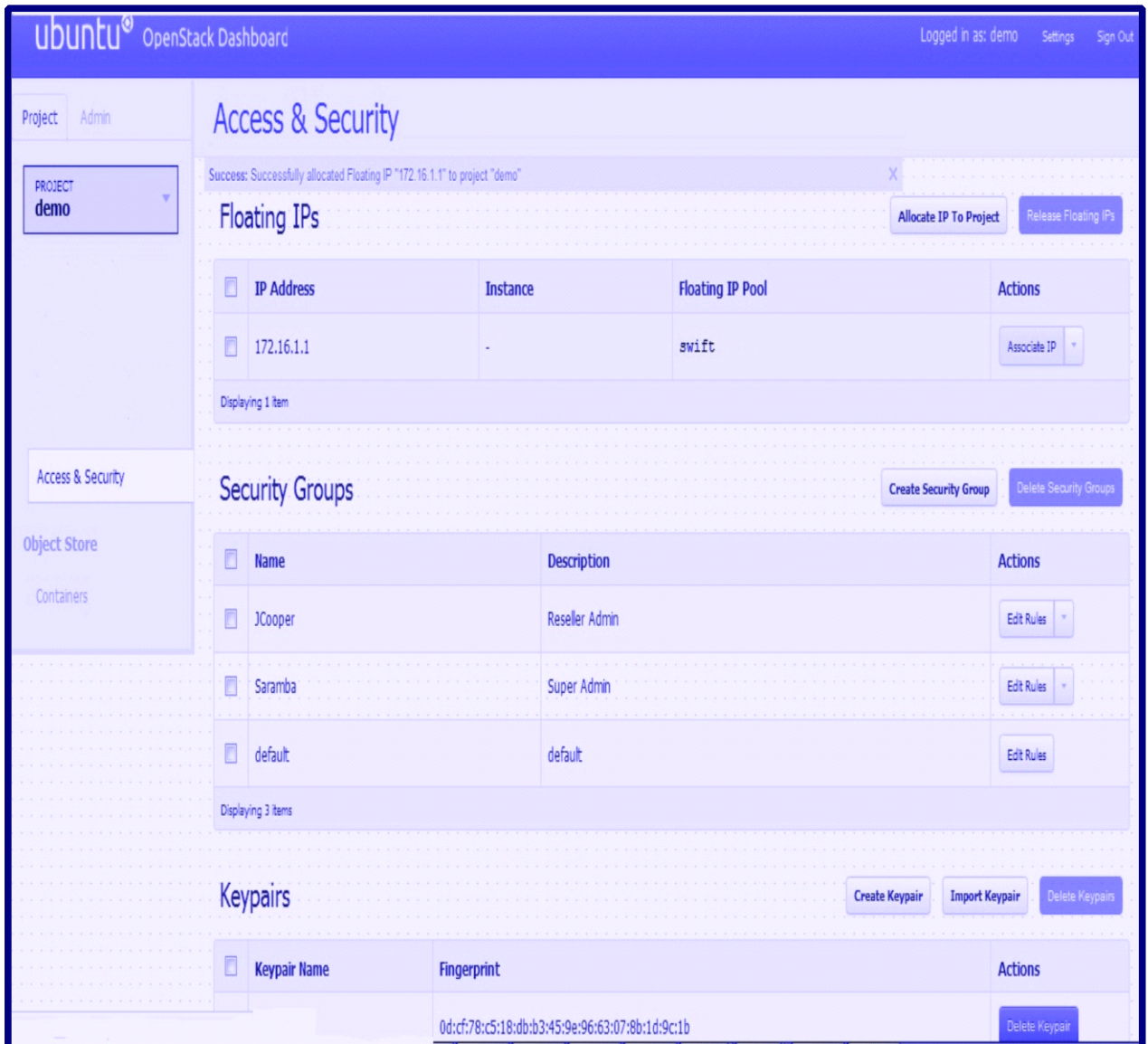


Figure 4.4 OpenStack admin section prototype for Access Control & Security

4.4.2 Too Much Access Rights for Reseller Admins

Reseller Admins have permissions to all accounts. They can do anything within any account as long as they possess the URL to that account. The documentation of OpenStack Object Storage does not mention in detail the actual permission of Reseller Admins. Only Admins and Users Permissions are mention in it [40].

In order to check whether this was true, an investigation was conducted on the system as follows:

Firstly, a user is added in the place of a Reseller Admin with name **saramba** to **accounta**. Secondly, container **filesb** is created on **accountb** and then files are uploaded to it. Afterwards, an attack was performed by using authentication service to obtain token and account URL for Reseller Admin as shown below:

```
=== HTTP Request ===
GET /v1 . 0 HTTP / 1 . 1
X Storage User : accounta : saramba
X Storage Pass : passaramba
Host : 127 . 0 . 0 . 2 : 11000
=== HTTP Response ===
HTTP / 1 . 1 204 No Content
X Storage Url : http : // 1 27 . 0 . 0 . 2 : 8 0 8 0 / v1 /
AUTH_e6595be640324be4abf5c4faa6cdc525
X Storage Token : AUTH_tk02393a74c8664132ab1a4d4144e5b1ea
X Auth Token : AUTH_tk02393a74c8664132ab1a4d5144e5b1ea
```

The HTTP response above, reveals that the authentication service is saying that the user **saramba** should use the URL from **X-Storage-Url** header to get access to the account.

Using the received authentication token (X-Auth-Token), but changing the storage URL to the one pointing to **accountb**, can literally allow a Reseller Admin to successfully receive statistics of **accountb** and the name of containers within this account (filesb):

```
=== HTTP Request ===
GET /v1 /AUTH_c075d948231a42d4b4386e890dfe7f34 HTTP / 1 . 1
X Auth Token : AUTH_tk02393a74c8664132ab1a4d5144e5b1ea
Host : 127 . 0 . 0 . 2 : 8080
=== HTTP Response ===
```

```
HTTP / 1 . 1 200 OK
X Account Object Count : 3
X Account Bytes Used : 20
X Account Container Count : 1
Content Length : 5
Content Type : text / plain ; charset=utf8
```

filesb

The files names from this container was obtain using the container name. Because these file names can be known, it is possible for anyone to issue HTTP requests in order to download these files or even delete them. Hence this confirms that any user with *Reseller Admin* role and with knowledge of the URL to this account has permissions to perform any action with the objects within any account.

This severely violates privacy of users who store their files on OpenStack Object Storage. Wether files are encrypted or not, it is not correct for Reseller Admins to have permissions to view other account details.

4.4.3 Elevation of Privileges Protection

In OpenStack Object Storage system, account management is similar to that of user management, and the protection against illegal elevation of privileges is literally the same for the two, since access control to objects is done based on access rights to a container where an object is stored. Access rights to containers are stored in ACLs. ACL itself is stored in a container database that is created for each of the existing containers. The source code that manipulates ACLs

(common/db.py) was checked and found that there is no possibility for injections [48].

4.5 Recommendations for IAM Issues in OpenStack Swift

The following recommendations can be used to mitigate the issues that have been found in the Identity and Access Management in OpenStack Object Storage. The following provides recommendations for the issues experienced by both users and developers of OpenStack:

Firstly, the file(etc/swift/auth.db) in DevAuth which stores user information must be examine in order to make a change to the access right. Only the swift user actually needs read right to this file. This mitigates the risk of obtaining user informations(like passwords) by other users.

Secondly, during installation of the OpenStack swift the default super user's password must be changed. The access right to the file having this information must be set such that only the owner should have read/write permission and users should take note of the excess rights given to Reseller Admins in viewing other account details. Also though ACL manipulation seems to be impossible, it is still recommended that users use HTTPS instead the HTTP Referrer field when making authorization decisions, since the HTTP field can easily be forge.

Thirdly, developers should implement user provisioning functionality using Service Provisioning Markup Language (SPML) and for that of authentication functionality, they should use Security Assertion Markup Language (SAML) to allow identity federation and avoid locking into proprietary solutions. The PySAML2 Python library can be used by the developers.

Lastly, Having a system which can check for the strength of password before registering users, can help eliminate weak passwords (python-crack Library can be

very useful) and these passwords should be hashed before stored in either DevAuth or SwAuth authentication system. It is even wiser for passwords to be concatenated to a salt and/or username and then hashing them with an appropriate algorithm, before storing [56]. Preferably, the SHA-256 algorithm can be used to hash them. And in the case where a salt value is used (not username), it can be stored alongside hashed password [48].

CHAPTER FIVE

Data Management In OpenStack Object Storage

This chapter analyses data management in OpenStack Object Storage. It looks at the legal issues of data location compliance, data isolation, backup, recovery and deletion, encryption and key management and data integrity.

The issues have been analyzed and recommendations provided as well.

5.1 Locating Data In OpenStack

Locating user's data in OpenStack Object Storage is done via HTTP calls to a Proxy server. The logical path to an account, container, or object is known using obtained storage URL from the authentication server. This path normally ends with the account name for which the user is registered. For instance, the code below shows logical path to the user object myObject, which is stored in container myContainer:

https://<PROXY-IP>:8080/v1/AUTH_e5825ba750324be4abf5c4faa6cdc524/myContainer/myObject

Having the storage account, the proxy server is then used to translate the logical path to a physical location where the data actually reside in the cluster. The proxy server uses the concept of rings to determine the physical data location in the cluster[59]. This cluster is divided into partitions, and each of these partition is then mapped to a device (a hard disk on one of the nodes in the cluster). The ring structure is used to determine on which nodes the partition has to be replicated.

When the proxy server finally finds the physical location, it will then contact a dedicated server process on a Storage node. The data is then sent from Proxy server to a corresponding service entity.

5.2 OpenStack Isolation and Possible Attacks

5.2.1 How Isolation is Done In OpenStack Object Storage

One of the main features of cloud computing is sharing resources among different users (customers). When it comes to OpenStack Object storage, its all about storage sharing. The issue of data isolation is irrelevant when dealing with private cloud deployment, since utilization of the cloud storage is done only by a single organization. However, for public deployment, this issue is of high importance. The separation of data that belong to different users (customers) has been given enough attention in most security documents. Data isolation can be referred to as words isolation, or compartmentalization. But the main idea here, is that of "data separation".

The directory structure for OpenStack Object Storage entities is shown in figure 5.1. From figure 5.1, accounts, containers, and objects have different directory on the storage device. All information about accounts and containers are stored in SQLite database files, while those about objects are stored as files with extension ".data". The temporary directory is used for storing file chunks during upload. There are partitions within each of the directories of OpenStack entities and these partitions are referred to as a directory whose name equals to a number between 1 and $2^{\langle \text{max-partition-number} \rangle}$ [48]. The **Max-partition-number** is selected when creating the ring structure. Since the partition number is determined by bit shift arithmetic, it is possible that files belonging to different users from different accounts will be stored within the same partition directory.

```
drwxrwxrwx 3 swift swift 12 22-02-2013 06:08 accounts/  
drwxrwxrwx 3 swift swift 12 22-02-2013 06:08 containers/  
drwxrwxrwx 3 swift swift 12 22-02-2013 06:08 object/  
drwxrwxrwx 3 swift swift 4 22-02-2013 06:08 tmp/
```

Fig. 5.1 OpenStack Object Storage directory structure

5.2.2 Attacks on OpenStack Isolation

OpenStack relies only on path hashing to isolate files belonging to different users[48]. In order to investigate this further whether OpenStack approach to isolation can be abused, let's assume that there is an attacker within the system. Two possible attacks can be considered: the preimage attack and that of collision attack.

In the Preimage Attack, attacker needs to have knowledge of the path to a file and then find the names for a container and object that will hash to the same value as that of the user's file of known path. But as at now, there are no such algorithm to find an input which will match the desired output of a hash function. It's hard to conclude that the said algorithm is unknown to any intelligence agencies or that it will not be found in the nearest future. However, there are no reasons to state that this approach taken by OpenStack is flawed.

The Collision Attack involves the case where an attacker tries to generate names of files, which will eventually result in the same hash value and then changing the "container/object" suffix. This attack can possibly be performed on MD5 [60]. Fortunately enough, this attack can actually be prevented by OpenStack. It is done by adding the *hash_path_suffix*, to a file path before passing the input to a hash function[61].

However, it is important to note that the best measure to take if one must avoid such attack is by keeping the *hash_path_suffix* value as secret [64]. Besides, other hashing functions than MD5 can also be considered [63].

5.3 Backup and Recovery

Data backup is the process of making a complete secondary copy of a data. The objective of this mechanism is to prevent "data loss, unwanted data overwrite, and destruction" [65]. By doing backup operations, data can be easily recovered if there is any kind of disaster in the system.

OpenStack Object Storage prevents data loss and provides availability by use of replication mechanism. That is, storing data in several location across a cluster. A dedicated server process called replicator is responsible to propagate data copies to different nodes within the Object Storage system. And of course there are separate processes responsible for the replication of accounts, containers, and objects.

Literally, OpenStack does not support data backup/recovery and this is a big problem. When a new object is uploaded with a name equal to an existing one, all the previous versions of this file will be deleted. Though replication prevents data loss, in the case of unwanted data overwrite, it is impossible to restore file to a previous version [66] [67].

Backup/recovery operation can possibly be added to OpenStack without much changes in the source code. After a file has been successfully uploaded via PUT method, the stored file will then receive a name equal to timestamp, and all the previous versions of the file will be deleted.

5.4 Data Deletion

Data deletion basically deals with the removal of all copies of the data that exists in a cluster. One should take note of the fact that it's possible for the data to be deleted in all, but one storage node becomes restored afterwards due to recovery procedure. Also a major issue with deleting data is dealing with proper storage recycling since files (or some parts of it) that were deleted can be recovered from the hard disk later.

In OpenStack Object Storage, the files are first written to a temporary location on a storage device, which by default is set to `/srv/node/sdb1/tmp`. The temporary directory is common for all the users (customers) on the system.

Afterwards, when all the file chunks are uploaded, this file is then moved to a new location by using the Python `os.rename` function. Also the new location for the file

is determined by the hashing approach that was described earlier in section 5.1. The new file will take a name equal to the timestamp specified in the HTTP header X-Timestamp.

When a user wants to delete a file, OpenStack will then create a new zero-size file with extension *.ts (tombstone) and new timestamp as a name. Afterwards, the algorithm which is used to delete files with older time-stamps is run. With the help of the tombstone file, OpenStack can safely resolve the problem of removing file from all the nodes: the tombstone file can later be propagated to other nodes using the replication process, and at the same time the actual content of the file is been deleted.

5.5 Encryption and Key Management

In OpenStack Object Storage, files are not encrypted before they are stored in a cluster [68]. In other words, users having sensitive information should first encrypt their files before storing them in OpenStack swift system and also manage their encryption keys themselves.

One reason for users to encrypt their files before storing in OpenStack is that, users in role of Reseller Admin can view any file on any of the accounts, as noted in section 4.4.2.

5.6 Data Integrity

In OpenStack, it's a possible to perform integrity verification on the server side. This is done by providing the MD5 hash of the uploaded object in ETag HTTP header [69]. OpenStack will then calculate the MD5 hash on the server and compare the two values. If the user-supplied value and calculated value are not the same, an HTTP response code 422 is returned. According to the RFC 4918, it literally means that the "server understands the content type of the request entity, and the syntax of the request entity is correct, but was unable to process the

contained instructions" [70].

OpenStack Object Storage provides the following possibilities for integrity checks:

1. End-to-end data integrity checks during the transfer, done on the server side.
2. End-to-end data integrity checks during the transfer, done on the client side.
3. Continuous integrity monitoring of objects stored in the cluster.

In the case of the client-side, users do the integrity checks by using the ETag value from the HTTP response. However, in case of integrity errors, the erroneous version of a file will be stored and replicated across OpenStack cluster. As a way of preventing others from downloading incorrect data, users will have to upload the same file again. Hence the need to perform server-side integrity checks during upload.

When analyzing the source code on the Storage server, a daemon process is responsible for making integrity checks and is called auditor. The accounts, containers, and objects all have their separate auditors. The account and container auditors are limited to checking the possibility of reading data from the database. While that of the Object auditor is responsible for checking file size and hash value by comparing the actual size and MD5 hash value to that which is stored in the object meta-data. Object meta-data is known to be stored as extended attributes on a file-system, and this is the reason why only those file-systems that support the said attributes can be used. If the server process discovers that the actual hash value of an object different from that stored in meta-data, a file will be quarantined and a correct version of it will be reloaded correctly from another replica in a cluster.

5.7 Recommendations for Data Management

This chapter has been concluded with the following recommendations:

It must be ensured that the file `/etc/swift/swift.conf` is protected from modifications. If the `hash_path_suffix` value is modified, previously uploaded files to OpenStack Object Storage will become inaccessible. It is necessary to encrypt all important and sensitive files before uploading them to OpenStack, in order to stop users (those acting in a role of Reseller Admin) from viewing files which belong to other users.

Data location compliance can be assured by either making modification to ring structure, or building a wrapper over the Ring class that reuses `get_more_nodes` method since this will allow restriction of data location to only specific zones.

Finally, it is recommended to make a number of kept backup copies configurable, and to change `unlinkold` method in `DiskFile` class to delete only those previous versions of files that exceed the configured value. This will allow backup and recovery operations to take place in OpenStack Object Storage.

CHATER SIX

Summary of Contributions and Future Work

6.1 Contributions

This thesis contributes towards improving the security related issues which are present in cloud computing in general and also to those within the Open-Stack Object Storage. It attempts in showing that there is still more to be done in implementing proper security and privacy mechanisms the cloud.

The research work looks at possible security flaws found in OpenStack Object Storage(swift). Different documents which are geared towards facilitation of the adoption of cloud computing were examined and the most important security issues were considered. These issues were recorded in chapter three.

Also, by installing the OpenStack cloud, several security weaknesses were found in the areas of Identity and Access Management, and that of Data Management. These two major areas were covered in chapter four and five respectively.

From the investigation done using the installed OpenStack Object Storage, it was found that there are possibilities for administrators with lower permissions to obtain credentials of other administrators with higher permissions. Another instance of security weakness, is the possibility of a particular administrators to be granted with too much access rights. This subsequently gives the said administrator the permission to read/delete any file belonging to any user. Also, issues of poor password management procedures for both authentication systems (SwAuth and DevAuth) was a found.

With further investigations of how data is managed in OpenStack, it was found that there is possibility for isolated files to be compromised. This could possibly

lead to legal claims from dishonest customers(users). Another major problem with most cloud providers is that they don't afford to encrypt user's data. OpenStack is not an exception to this problem.

Nevertheless, some security measures can be implement in order to mitigate these weaknesses. First and foremost, it is essential to have a system which can check the strength of user's password during registration. The file in DevAuth which stores user information must be properly examined in order to make change to the access rights. Only the swift user actually needs read right to this file. This will prevent unauthorized people from obtaining user informations.

For user provisioning functionality, developers should use Service Provisioning Markup Language (SPML) and for that of authentication functionality, Security Assertion Markup Language (SAML) can be recommended. The PySAML2 Python library can be used. Lastly, users must ensure to encrypt all sensitive data/files before storing them in cloud. More details of the findings and recommendations can be found in chapter three and four of this thesis.

Conclusion

Both the theoretical and practical findings have clearly shown that there is great need to improve the security and privacy situation in cloud computing . Since cloud computing is relatively new, it is necessary to undertake more studies of security issues in cloud computing. This will improve the current security status of the cloud and as a result more people will find it necessary to adopt and become users of cloud services,thereby creating a steady economic growth for both the service providers and users.

6.2 Future Work

Cloud computing is still a new innovative approach on IT usage and there are many issues still to be considered for its improvement. It has many security concerns which involve both technical (for instance, reliability, security and privacy etc) and non-technical (for instance, legal and economic) issues. Therefore, more research work should be done in the direction of Security.

Consideration of other OpenStack Project: This research considers only the OpenStack Object Storage (swift) which is just one of the three projects. In the future, the OpenStack Compute and OpenStack Image Service should also be investigate for security issues.

Provide better data encryption methods: more research should be done in order to develop better data encryption methods. And, cloud service providers must endeavour to implement these data encryption methods, for proper protection of user(customer) data.

Glossary

Collision Attack

An attack on hash function that aims to find two messages that have same hash values.

Hypervisor

See Virtual Machine Monitor

Infrastructure as a Service (IaaS)

Cloud service delivery model under which customer can use provider's computing resources to deploy and run arbitrary software which can include operating systems and applications

Platform as a Service (PaaS)

Cloud service delivery model under which customer can use provider's development environment to create applications and deploy them on provider's cloud Infrastructure.

Preimage Attack

An attack on hash function that aims to find a message that has a specific hash value.

Private Cloud

Cloud deployment model under which cloud infrastructure is utilized by a single organization.

Public Cloud

Cloud deployment model under which cloud infrastructure is made available to the general public.

Software as a Service (SaaS)

Cloud service delivery model under which customer can use provider's applications running on a cloud infrastructure.

User De-provisioning

A process of un-registering users from a system.

User Provisioning

A process of registering new users in a system.

Virtual Machine Monitor

A layer of software between an operating system and hardware that is used to operate virtual machines

Bibliography

- [1] : http://en.wikipedia.org/wiki/Cloud_computing. (Accessed date 12/10/2012)
- [2]http://www.matthewb.id.au/index.php?option=com_content&view=article&id=31:cloudcomputin. (Accessed date:04/01/2013)
- [3] Minqi Zhou, Rong Zhang, Wei Xie, Weining Qian, Aoying Zhou: " Security and Privacy in Cloud Computing": A Survey, 2010.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5663489>. (Accessed date 10/10/2012)
- [4] C. Wang, "Forrester: A close look at cloud computing security issues,"
<http://www.csoonline.com/article/496388/forrester-a-close-look-at-cloud-computing-security-issues>
(Accessed date 10/11/2012)
- [5]Peter Mell and Tim Grance. The NIST definition of cloud computing. October 2009.
<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.(Accessed date 10/06/2012)
- [6]N.Robinson, L. Valeri, J.Cave & T. Starkey ,H.Graux, S. Creese & P. Hopkins, " The Cloud: Understanding the Security, Privacy and Trust Challenges" Nov, 10'.
<http://tofudi.net/read-file/the-cloud-understanding-the-security-privacy-and-trust-challenges-pdf-757583/>. (Accessed date 17/11/2012)
- [7] H.Takabi and J.b.d. Joshi (University of Pittsburgh) and Gail-Joon Ahn," Security and Privacy Challenges in Cloud Computing Environments" Nov/Dec 10'.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5655240> (Accessed date 10/11/2012)
- [8]Bevan Barton, "Security and Privacy in Cloud Computing"(Middlebury College), May 2010. <http://vision.middlebury.edu/~cs702/>. (Accessed date 15/01/2013)
- [9]Cigoj P. Security Issues in OpenStack. Master's Seminar (2012).
http://kt.ijs.si/markodebeljak/Lectures/Seminar_MPS/2012_2013/Seminars/Seminar%20I_Primoz_Cigoj.pdf. (Accessed date 05/02/2013)
- [10]G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," Cryptology ePrint Archive, Report 2007/202, 2007, <http://eprint.iacr.org/>. (Accessed date 10/01/2013)

- [11] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in Proc. of ESORICS'09, Saint Malo, France. <http://eprint.iacr.org/2009/281.pdf> (Accessed 14/01/13)
- [12] H. Shacham and B. Waters, "Compact proofs of retrievability," in Proc. of Asiacrypt 2008, vol. 5350, Dec 2008, pp. 90–107.
- [13] A. Juels and J. Burton S. Kaliski, "Pors: Proofs of retrievability for large files," in Proc. of CCS'07, Alexandria, VA, October 2007, pp. 584–597.
- [14] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in Proc. of HotOS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–6.
- [15] 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPPA)," Online at <http://aspe.hhs.gov/admsimp/pl104191.htm>, 1996. (Accessed :16/07/ 2011).
- [16] OpenStack. OpenStack Open Source Cloud Computing Software. <http://openstack.org/>. (Accessed 10/11/ 2012).
- [17] J. Curry. OpenStack Blog: Introducing OpenStack. <http://www.openstack.org/blog/2010/07/introducing-openstack/>. (Accessed 09/10/12).
- [18] C. Metz. The New Linux: OpenStack aims for the heavens. <http://www.theregister.co.uk/2011/01/08/openstack/>, January 2011. (Retrieved February 2013).
- [19] T. Morgan. NASA and Rackspace open source cloud fluffer. http://www.theregister.co.uk/2010/07/19/nasa_rackspace_openstack/, July 2010. (Retrieved February 2013).
- [20] <http://en.wikipedia.org/wiki/OpenStack>. (Retrieved February 2013).
- [21] John David Cooper "A Comparison and Analysis of Security Features and Solutions Provided by Current Cloud Platforms" Dec 2012.
- [22] J. Purrier. OpenStack Announces Cactus Release. <http://www.openstack.org/blog/2011/04/openstack-announces-cactus-release/>. (Retrieved February 2013)
- [23] OpenStack. OpenStack Image Service. <http://openstack.org/projects/image-service/> (Accessed Feb 2013).

- [24] F. Gens. IT Cloud Services User Survey, pt.2: Top Benefits & Challenges. <http://blogs.idc.com/ie/?p=210>, October 2008. (Retrieved February 2011).
- [25] F. Gens. New IDC IT Cloud Services Survey: Top Benefits and Challenges. <http://blogs.idc.com/ie/?p=730>, December 2009. (Retrieved February 2011).
- [26] W. Jansen and T. Grance. Guidelines on security and privacy in public cloud computing. Technical report, National Institute of Standards and Technology, January 2011. Draft Special Publication 800 -144. Available at http://csrc.nist.gov/publications/drafts/800-144/Draft-SP-800-144_cloud-computing.pdf.(Accessed date 02/11/2012)
- [27] D. Talbot. Security in the Ether. Technology Review, pages 36–42, February 2010.
- [28]<http://uksysadmin.wordpress.com/2011/02/17/running-openstack-under-virtualbox-a-complete-guide/> (Accessed date 04/01/2013)
- [29]<http://www.tikalk.com/alm/blog/expreimenting-openstack-essex-ubuntu-1204-lts-under-virtualbox> (Accessed date: 04/01/2013)
- [30] Jackson K. (2012) OpenStack Cloud Computing Cookbook. PACKT: Birmingham- Mumbai
- [31] E. Bertino, F. Paci, and R. Ferrini, “Privacy- Preserving Digital Identity Management for Cloud Computing,” IEEE Computer Society Data Engineering Bulletin, Mar. 2009, pp. 1–4.
- [32.] P.J. Bruening and B.C. Treacy, “Cloud Computing: Privacy, Security Challenges,” Bureau of Nat’l Affairs, 2009; www.hunton.com/files/tbl_s47Details/FileUpload265/2488/CloudComputing_Bruening-Treacy.pdf. (Accessed date 02/02/2013)
- [33.] E. Bertino, F. Paci, and R. Ferrini, “Privacy- Preserving Digital Identity Management for Cloud Computing,” IEEE Computer Society Data Engineering Bulletin, Mar. 2009, pp. 1–4.
- [34.] Guidance for Identity & Access Management V2.1 Prepared by theCloud Security Alliance April 2010: <https://cloudsecurityalliance.org/guidance/csaguide-dom12-v2.10.pdf>
- [35.] J. Joshi et al., “Access Control Language for Multi-domain Environments,” IEEE Internet Computing, vol. 8, no. 6, 2004, pp. 40–50

- [36] Hassan Takabi and James B.D. Joshi: 'Security and Privacy Challenges in Cloud Computing Environments', November/December 2010: 26-28.<http://www.sis.pitt.edu/~jjoshi/courses/IS2620/Spring13/S&P.pdf>. (Accessed 10/11/2012)
- [37.] Krešimir Popović, Željko Hocenski: 'Cloud Computing Security Issues and Challenges', May 24-28, 2010, Opatija, Croatia: 344-345.
- [38.] A. Jyoti Choudhury et al: "A Strong User Authentication Framework for Cloud Computing" in IEEE Asia -Pacific Services Comp Conf., pp 110 -115, 2011.
- [39.] S. Yu, C. Wang, K. Ren and W. Lou "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing" the IEEE INFOCOM proceedings, 2010.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5462174>. (Accessed 20/12/2012)
- [40] "IBM Discovers Encryption Scheme That Could Improve Cloud Security, Spam Filtering," at
<http://www.eweek.com/c/a/Security/IBM-Uncovers-Encryption-Scheme-That-Could-Improve-Cloud-Security-Spam-Filtering-135413/>. (Accessed 02/12/2012)
- [41] Roy I, Ramadan HE, Setty STV, Kilzer A, Shmatikov V, Witchel E. "Airavat: Security and privacy for MapReduce," In: Castro M, eds. Proc. of the 7th Usenix Symp. on Networked Systems Design and Implementation. San Jose: USENIX Association, 2010. 297.312.
- [42] I-H Chuang, S-H Li, K-C Huang, Y-H Kuo; "An Effective Privacy Protection Scheme for Cloud Computing" in Center for Research of E-life Digital Technology (CREDIT), Taiwan, pp 260-265, Feb 2011
- [43] S. Lee, I. Ong, H.T. Lim, H.J. Lee, "Two factor authentication for cloud computing", International Journal of KIMICS, vol 8, Pp. 427-432.
- [44] http://www.usebox.net/jjm/charlas/deploying_openstack_object_storage.pdf
. (Retrieved date: 26/02/2013)
- [45] OpenStack. The Auth System - Swift v1.2.0 documentation.
http://docs.openstack.org/developer/swift/overview_auth.html. (Accessed 26/02/2013).
- [46] SQLite. Features. <http://www.sqlite.org/features.html>. (Accessed Date 02/09/12).

- [47] OpenStack. Open Source Cloud Computing Software.
http://swift.openstack.org/1.2/overview_auth.html (Accessed date 15/10/2011).
- [48] Slipetsky, R : Security Issues in OpenStack. Master's Thesis (2011)
http://nordsecmob.aalto.fi/en/publications/theses_2011/thesis_slipetsky.pdf.
(Accessed date 03/04/2013)
- [49] Python Software Foundation. sqlite3 - DB-API 2.0 interface for SQLite databases. <http://docs.python.org/library/sqlite3.html>. (Retrieved March 2013)
- [50] SecurityFocus. SecurityFocus Vulnerability Database.
<http://www.securityfocus.com/bid>. (Accessed March 2013)
- [51] Sky Worth: Identity Federation; <http://www.skyworthttg.com/en/solutions/identity-federation/>. (Accessed March 2013).
- [52] G. Brunette and R. Mogull. "Security Guidance for Critical Areas of Focus in Cloud Computing", Version 2.1. Technical report, Cloud Security Alliance, December 2009. <http://www.cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf>. (Accessed date 07/03/2012)
- [53] Launchpad Project. Python implementation of SAML2.
<https://launchpad.net/pysaml2>. (Accessed March 2013)
- [54] OpenStack. The Auth System - Swift v1.2.0 documentation.
http://swift.openstack.org/1.2/overview_auth.html. (Accessed March 2013).
- [55] OpenStack. IRC Log for April 26, 2011.
<http://eavesdrop.openstack.org/irclogs/%23openstack/%23openstack.2011-04-26.log>. (Retrieved March 2013).
- [56] Burr, W. E.; Dodson, D. F.; Polk, W. T. Electronic Authentication Guideline. Technical report 800-63, National Institute of Standards and Technology (2006)
- [57] Python Software Foundation. hashlib - Secure hashes and message digests.
<http://docs.python.org/library/hashlib.html>. (Accessed March 2013).
- [58] W. Jansen and T. Grance. Guidelines on security and privacy in public cloud computing. Technical report, National Institute of Standards and Technology, January 2011. Draft Special Publication 800-144.
http://csrc.nist.gov/publications/drafts/800-144/Draft-SP-800-144_cloud-computing.pdf. (Accessed date 09/05/12)

- [59] OpenStack. The Rings - Swift v1.2.0 documentation. http://swift.openstack.org/1.2/overview_ring.html. (Accessed date April 2013).
- [60] M. Stevens, A. Lenstra, and B. Weger. Chosen-prefix collisions. <http://www.win.tue.nl/hashclash/ChosenPrefixCollisions/>, February 2007. Accessed April 2013.
- [61] M. Stevens, A. Lenstra, and B. Weger. Chosen-prefix collisions. <http://www.win.tue.nl/hashclash/ChosenPrefixCollisions/>, February 2007. (Accessed 26/03/2013)
- [62] M. Stevens, A. Lenstra, and B. Weger. Predicting the winner of the 2008 US Presidential Elections using a Sony PlayStation 3. <http://www.win.tue.nl/hashclash/Nostradamus/>, November 2007. (Accessed April 2013).
- [63] R. Slipetsky. Two paths to the files hashing to the same value. <https://answers.launchpad.net/swift/+question/156307>, April 2011. (Accessed April 2013).
- [64] J. Dickinson. Ensure that the docs explicitly caution to keep the hash path suffix secret. <https://bugs.launchpad.net/swift/+bug/791620>, June 2011. (Accessed April 2013).
- [65] G. Brunette and R. Mogull. Security Guidance for Critical Areas of Focus in Cloud Computing, Version 2.1. Technical report, December 2009. Available at <http://www.cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf>. (Accessed date 07/03/2012)
- [66] R. Slipetsky. Re: Suggestion for data backup/recovery in swift. <https://lists.launchpad.net/openstack/msg02664.html>, May 2011. (Accessed April 2013).
- [67] R. Slipetsky. Suggestion for data backup/recovery in swift. <https://lists.launchpad.net/openstack/msg02632.html>, May 2011. (Accessed 04/ 2013).
- [68] OpenStack. Containers and Objects. <http://docs.openstack.org/cactus/openstack-object-storage/admin/content/containers-and-objects.html>. (Accessed April 2013)
- [69] OpenStack. Create/Update Object. <http://docs.openstack.org/cactus/openstack-object-storage/developer/content/create-update-object.html>. (Accessed May 2011).
- [70] IETF Network Working Group. RFC 4918: HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). <http://tools.ietf.org/html/rfc4918>, June 2007. (Retrieved May 2011).

Appendix A

Suggestion for Implementing Backup/Recovery in OpenStack

The information below is only an extract from the mailing list discussion analyzing backup and recovery in OpenStack.

openstack team mailing list archive

Thread Date

 openstack team » Mailing list archive » Message #02632

suggestion for data backup/recovery in swift

[Thread Previous](#) • [Date Previous](#) • [Date Next](#) • [Thread Next](#)

To: OpenStack <openstack@xxxxxxxxxxxxxxxxx>
From: Rostyslav Slipetsky <rslipetsky@xxxxxxxx>
Date: Mon, 30 May 2011 14:56:10 -0700 (PDT)

As far as I see swift does not support data backup/recovery. Sometimes backup/recovery might prevent damages from unwanted data overwrites, which is why I suppose that many users would like to see this functionality in OpenStack.

It looks like backup/recovery can be easily added to swift without many changes in the source code. After successful file upload via PUT method, stored file receives name equal to timestamp and all the previous versions of the file are deleted by DiskFile#unlinkold method (swift/obj/server.py). If one wants to allow keeping up to N backup versions (can be configurable value), DiskFile#unlinkold method can be modified to keep last N versions of a file and delete all the others. Afterwards, recovery would mean deleting last version of

the file based on timestamp and will be allowed up to depth N-1 (this will not allow "recovery from recovery" though). Of course, modifications to ObjectController will be necessary to allow dedicated HTTP call to recover object to

its previous version.

Best Regards,
Rostik

Re: suggestion for data backup/recovery in swift

[Thread Previous](#) • [Date Previous](#) • [Date Next](#) • [Thread Next](#)

To: Michael Barton <mike-launchpad@xxxxxxxxxxxxxxxx>

From: Rostyslav Slipetsky <rslipetsky@xxxxxxxx>

Date: Tue, 31 May 2011 14:49:21 -0700 (PDT)

Cc: OpenStack <openstack@xxxxxxxxxxxxxxxx>

In-reply-to: <BANLkTikcoE79aU9dWArh21cRFyT1uDB3Ew@mail.gmail.com>

> Let's say you're backing up a Nova instance to Swift every day using
> versioning, and each backup is 5gb. After a few weeks, that hard
> drive may be storing over 100gb for one "file". Swift has no way of
> taking that into account when placing files, so the distribution is
> going to get clumpy, like one drive might get full while another one
> is only half used.

I assume that in this scenario there is only one user/file of huge size and all the other users/files are much smaller (in other case, there will be not much difference in distribution among drives). But the suggested backup/recovery approach can work even in this scenario. If we have a configurable value that specifies maximum number of backups/versions, then in a setup where such a use case is possible, the max_file_backups value may be set to disallow backups at all (by default backups can be disabled). In other case, when OpenStack is used as a backend for let's say text documents, backups can be enabled and another cloud provider might benefit from this feature.

- Rostik

Appendix B

Configuration Files Used for the Deployment of OpenStack Object Storage

Base configuration file for each node in OpenStack Object Storage:

```
1 [swift-hash]
2 swift_hash_path_suffix = qwerty123456asdfgh7890zxcvbn
```

Configuration file for Proxy server and Devauth authentication/authorization system:

```
1 [DEFAULT]
2 cert_file = /etc/swift/cert.crt
3 key_file = /etc/swift/cert.key
4 bind_port = 8080
5 workers = 4
6 user = swift
7
8 [pipeline:main]
9 pipeline = healthcheck cache auth proxy-server
10
11 [app:proxy-server]
12 use = egg:swift#proxy
13 allow_account_management = true
14 set log_name = swift-proxy
15 set log_level = DEBUG
16
17 [filter:auth]
18 use = egg:swift#auth
19 ssl = true
20
21 [filter:healthcheck]
```

```
22 use = egg:swift#healthcheck
23
24 [filter:cache]
25 use = egg:swift#memcache
26 memcache_servers = 10.0.0.2:11211
```

Configuration file for Proxy server and Swauth authentication/authorization system:

```
1 [DEFAULT]
2 cert_file = /etc/swift/cert.crt
3 key_file = /etc/swift/cert.key
4 bind_port = 8080
5 workers = 4
6 user = swift
7
8 [pipeline:main]
9 pipeline = healthcheck cache swauth proxy-server
10
11 [app:proxy-server]
12 use = egg:swift#proxy
13 allow_account_management = true
14 set log_name = swift-proxy
15 set log_level = DEBUG
16 set log_headers = True
17
18 [filter:swauth]
19 use = egg:swift#swauth
20 set log_name = swift-swauth
21 set log_level = DEBUG
22 set log_headers = True
23 default_swift_cluster = local#https://10.0.0.2:8080/v1
24 super_admin_key = swauth
25
26 [filter:healthcheck]
27 use = egg:swift#healthcheck
28
29 [filter:cache]
30 use = egg:swift#memcache
31 memcache_servers = 10.0.0.2:11211
```

Configuration file for Authentication server (needed when Devauth is used):

```
1 [DEFAULT]
2 cert_file = /etc/swift/cert.crt
3 key_file = /etc/swift/cert.key
4 user = swift
5
6 [pipeline:main]
```

```
7 pipeline = auth-server
8
9 [app:auth-server]
10 use = egg:swift#auth
11 default_cluster_url = https://10.0.0.2:8080/v1
12 super_admin_key = devauth
13 set log_name = swift-auth
14 set log_level = DEBUG
```

Configuration files for Account service on Storage node:

```
1 [DEFAULT]
2 bind_ip = 10.0.0.5
3 workers = 2
4
5 [pipeline:main]
6 pipeline = account-server
7
8 [app:account-server]
9 use = egg:swift#account
10
11 [account-replicator]
12
13 [account-auditor]
14
15 [account-reaper]
```

Configuration files for Container service on Storage node:

```
1 [DEFAULT]
2 bind_ip = 10.0.0.5
3 workers = 2
4
5 [pipeline:main]
6 pipeline = account-server
7
8 [app:account-server]
9 use = egg:swift#account
10
11 [account-replicator]
12
13 [account-auditor]
14
15 [account-reaper]
```

Configuration files for Object service on Storage node:

```
1 [DEFAULT]
2 bind_ip = 10.0.0.5
```

```
3 workers = 2
4
5 [pipeline:main]
6 pipeline = object-server
7
8 [app:object-server]
9 use = egg:swift#object
10
11 [object-replicator]
12
13 [object-updater]
14
15 [object-auditor]
```