NTNU
Norwegian University of
Science and Technology

Security Issues in OpenStack

Rostyslav Slipetsky

Master in Security and Mobile Computing
Submission date: June 2011
Supervisor: Danilo Gligoroski, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Cloud computing is a relatively novel topic in Information Technology which attracts significant attention within the IT community nowadays. Despite the increasing amount of attention towards cloud computing, different obstacles to its widespread adoption still exist, security being usually named as the most important and challenging issue that prevents numerous enterprises to switch to cloud computing. In order to provide assistance for organizations in adopting cloud technologies various institutions involved into a process of creating guidelines and recommendations for secure transition to cloud services.

In an effort to create an open source cloud computing platform that will be easy to use and massively scalable an OpenStack project was started. OpenStack is a collection of open source technology products delivering a scalable, secure, standards-based cloud computing software solution. OpenStack is currently developing two interrelated technologies: OpenStack Compute and OpenStack Object Storage. OpenStack Compute is the internal fabric of the cloud creating and managing large groups of virtual private servers. OpenStack Object Storage is a software for creating redundant, scalable object storage using clusters of commodity servers to store terabytes or even petabytes of data.

Within this Master thesis the student will have to learn the principles of OpenStack, install it in a virtualized environment that emulates a cloud datacenter and start to use it. Afterwards a study of security issues pertinent to cloud computing by examining evolving standards and recommendations will be performed, which will form a basis for analyzing how identified security issues are mitigated in OpenStack.

Assignment given: January 15, 2011

Supervisor: Prof. Danilo Gligoroski

Abstract

Cloud computing is a relatively novel topic in Information Technology that attracts significant attention from the public and private sectors nowadays. Despite the amount of attention towards cloud computing , barriers to widespread adoption of it still exist, security being named the most important one. OpenStack is an open-source software for building private and public clouds, which was initially created by the source code contribution from NASA, the National Aeronautics and Space Administration of the USA, and Rackspace, a company providing web hosting and cloud computing services. At the moment the number of companies and organizations involved in the OpenStack project has grown to 76, which emphasizes the importance of studying OpenStack and researching how various security issues are handled in this software.

Since cloud computing is a novel and dynamic area which still evolves, there is no worldwide accepted specification for security assessment of cloud computing services. At the same time different organizations try to create their own guidelines for cloud security. We start our work with comparing available cloud security documents and create a list of issues found in all analyzed documents. Based on that list we later analyze two selected areas in OpenStack Object Storage: 1) Identity and access management and 2) Data management.

As a result of our analysis we achieve the following: 1) find a security vulnerability which allows administrators with lower permissions to obtain credentials of administrators with higher permissions; 2) report inadequately high permissions of one type of administrators which allows to read/delete all the files of all the users; 3) find a possibility to compromise isolation of files with subsequent overwrite of one file by another. Besides, we report poor password management procedures employed in available authentication systems and submit our proposals for implementing data location compliance and backup/recovery functionality in OpenStack Object Storage.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) and the Technical University of Denmark (DTU) as a final work of the joint Master's Program in Security and Mobile Computing (NordSecMob) to fulfill requirements for the Master of Science degree.

The work on the thesis has been performed at the Department of Telematics, NTNU, Trondheim, under the supervision of Prof. Danilo Gligoroski (NTNU) and Associate Prof. Christian W. Probst (DTU) .

Author would like to thank his supervisors for their valuable help, suggestions, and feedback, which made this work possible. Besides, author would like to thank the NordSecMob consortium for the possibility to be admitted to the program and pursue the degree in the field of Security and Mobile Computing.

Trondheim, June 2011

Rostyslav Slipetskyy

List of Tables

List of Figures

Abbreviations

ACL	Access Control List
CPU	Central processing unit
CSA	Cloud Security Alliance
CSP	Cloud Service Provider
DoS	Denial of Service
ENISA	European Network and Information Security Agency
FIPS	Federal Information Processing Standards
IaaS	Infrastructure as a Service.
JSON	JavaScript Object Notation
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform as a Service.
ReST	Representational State Transfer
SaaS	Software as a Service.
SAML	Security Assertion Markup Language
SLA	Service Level Agreement
SPML	Service Provisioning Markup Language
TLS	Transport Layer Security
UI	User Interface
UUID	Universally Unique Identifier

VM	Virtual Machine
WSGI	Web Server Gateway Interface
XACML	eXtensible Access Control Markup Language

Chapter 1

Introduction

> "I was standing in what was a prototype of a new kind of power plant [Cloud data center] - a computing plant that would come to power our information age the way great electric plants powered the industrial age"
>
> Nicholas Carr, "The Big Switch"

Cloud computing is a relatively novel topic in Information Technology that attracts significant attention from the public and private sectors nowadays. For example, in a press release dated from January 2011 and published by Gartner, a worldwide information technology research and advisory company, cloud computing was named a number one technology priority for 2011, based on the survey that included responses from more than 2000 CIOs worldwide [10]. Governments also plan to invest into cloud computing. According to "U.S. Federal Cloud Computing Market Forecast 2010-2015", the US government projects that between 2010 and 2015 its spending on cloud computing will be at approximately a 40-percent compound annual growth rate and will pass $7 billion by 2015 [78]. Chinese government is expected to be the main customer of the city-size cloud computing data center, which is predicted to be finished in 2016 and is built by a local company in collaboration with IBM [18].

The immaturity of cloud computing provides many challenges as well as opportunities for breakthrough in innovative ideas and solutions. While industrial companies create cloud computing solutions and make profits, Academia has also recognized the importance of the topic, and numerous scientists have involved into advancing cloud computing research, fostering the big switch from computing being an on-premise facility to computing being an off-premise utility.

1.1 Motivation

Despite the amount of attention towards cloud computing, barriers to widespread adoption of it still exist. For two years in a row, security was named the most important concern of the cloud model according to the survey of IT executives and their line-of-business colleagues conducted by International Data Corporation (IDC) [11, 12]. National Institute of Standards and Technology (NIST) in the guidelines prepared for use by U.S. Federal agencies warns that "security challenges [that] cloud computing presents are formidable" [17]. Researchers from Massachusetts Institute of Technology (MIT) state that securing cloud computing is the "information technology's next grand challenge" [75]. That is why concentrating on security issues of cloud computing software products is of high importance.

Since cloud computing is a novel and dynamic area which still evolves, there is no worldwide accepted specification for security assessment of cloud computing services. In order to provide assistance for organizations in adopting cloud technologies, various institutions involved into creating guidelines and recommendations for secure transition to cloud services. Cloud Security Alliance (CSA), a non-profit organization formed from industry representatives, published its first version of "Security Guidance for Critical Areas of Focus in Cloud Computing" in April 2009 (at the time of writing version 2.1 is available [3]). Government-funded organizations in Europe and USA also developed guidelines for secure usage of cloud computing. European Network and Information Security Agency (ENISA) outlines cloud computing security challenges in the "Cloud Computing Security Risk Assessment" document [5]. In the United States, NIST released a suggested version of "Guidelines on Security and Privacy in Public Cloud Computing" in January 2011 for public consideration [17]. At the time of writing, we are unaware of any comparison between those documents. Such a comparison will aid in creating a comprehensive view on the important security issues pertinent to cloud computing.

Apart from security, another widely recognized challenge that affects adoption of cloud computing is portability problem (see [5, 12, 27]). Since Cloud Service Providers (CSP) use different and often proprietary technologies for cloud computing(for example, for storing data in the cloud), it becomes difficult for customers to switch between different CSPs, which might cause locking in to a specific provider. Eliminating the fear of proprietary lock-in for cloud customers by developing open source cloud computing platform that will meet the needs of public and private cloud providers is one of the goals of cloud computing software OpenStack [39]. Well-known founding members Rackspace and NASA, as well as the broad community support, including companies like AMD, Citrix, Dell, Intel [35], Microsoft [25] and Cisco [76], contribute to the belief that OpenStack might reach their goals. According to the cloud computing predictions for 2011 from CIO.com, a website which aims to serve chief information officers and other IT leaders, "the attractiveness of a complete open source cloud computing software stack will become clear, and interest and adoption worldwide of OpenStack will grow [during 2011]" [14], which emphasizes the importance of studying OpenStack and researching how various security issues are handled in this software.

1.2 Scope and Objectives

During the work on this thesis our main objective was to analyze how various security issues relevant to cloud computing were handled in OpenStack. Since OpenStack consists of many different projects (see section 2.2.2 for background information), we have decided to focus on one of them - OpenStack Object Storage. Security issues in other OpenStack projects are *not* discussed in this thesis.

In order to identify cloud computing security issues, we started with researching emerging guidelines and recommendations aiming to help organization in transition to cloud services. For our study we selected cloud computing security-related documents created by an organization that consists of industry representatives - Cloud Security Alliance, and two government-funded institutions from Europe and the USA: European Network and Information Security Agency, and National Institute of Standards and Technology. As a result of the study, we created a list of security issues that could be used when evaluating security of cloud solutions.

Using the list of identified security issues, we decided to focus on two large security areas: 1) Identity and access management, and 2) Data management. Some of the other cloud computing security issues, such as portability, application security, insider threats, were also given attention in our work; however, our main attention was devoted to the two areas named above.

1.3 Methodology

During the work on the thesis, we used the following approach:

1. Installed OpenStack in a virtualized environment that emulates a cloud datacenter.
2. Examined evolving standards and recommendations to identify a set of problems pertinent to cloud computing security.
3. Evaluated whether identified issues were taken care of in OpenStack.
4. If a given area of security concern was handled in OpenStack:
    - Analyzed implementation for weaknesses using black-box and white-box testing approaches.
5. If a given area of security concern was *not* handled in OpenStack:
    - Raised the issue among the developers using mailing list, IRC channel, etc.

- Suggested possible implementation to handle the identified security issue in OpenStack.

1.4 Our Contribution

In this section we enumerate tangible outcomes that were achieved during work on the thesis. We include only the *most important* achievements, from our point of view, into the list below. For more extensive coverage of the results, please refer to summary sections at the end of each chapter.

1. By studying emerging security recommendations for cloud computing from public and private sector, we were able to compile a list of security issues to be used when evaluating security of cloud solutions.

2. Using the list of security issues, we performed an extensive analysis of two selected areas in OpenStack Object Storage: 1) Identity and access management and 2) Data management.

3. During the analysis of Identity and access management, we accomplished the following:

    - A security vulnerability, which allowed administrators with lower permissions to obtain credentials of administrators with higher permissions, was found. The vulnerability was reported to OpenStack team and fixed afterwards (section 4.3.6).

    - Inadequately high permissions of one type of administrators, which allowed to read/delete all the files of all the users, were reported. OpenStack team claimed that such a functionality is required to be able to perform administrative tasks (for example, migrating user data between accounts), which we treated as an example of security vs. usability trade-off and recommended users to encrypt sensitive data before uploading to OpenStack (section 4.4.3).

    - A study of provided authentication mechanisms showed poor password management procedures, such as storing password in clear text, absent restrictions on password strength, etc.

4. During the analysis of Data Management, we accomplished the following:

    - A possibility to compromise isolation of files with subsequent overwrite of one file by another was found. By harming himself, a malicious user can claim damages to a data storage service provider due to this vulnerability (section 5.2).

    - Suggestions on implementing data location compliance and backup/recovery functionality were created and submitted to OpenStack team.

1.5 Thesis Outline

The remainder of this work is organized in the following way:

Chapter 2: We present background information on cloud computing and its essential characteristics, as well as possible deployment and service models. After that we describe OpenStack cloud solution, provide a historical insight of how OpenStack was started, give an overview of the projects that comprise OpenStack family. A description of test deployment of OpenStack Object Storage that was analyzed during this thesis work follows.

Chapter 3: We summarize our study of three documents that deal with cloud computing security issues from public and private agencies in the U.S. and Europe. We give an overview of each of the documents and then provide a compiled list of security concerns created from the analyzed documents.

Chapter 4: We present the results of in-depth analysis of Identity and access management in OpenStack Object Storage. We deal with problems of identity provisioning/deprovisioning, identity federation, authentication, authorization and access control.

Chapter 5: We analyze Data management in OpenStack Object Storage. We start this chapter with data location analysis, where we include a discussion on data location compliance. Then we continue with sections on isolation, backup and recovery, deletion, encryption and key management, and integrity verification.

Chapter 6: Our conclusions are presented in this chapter.

Chapter 2

Background

In this chapter we will present the background information required for understanding the remainder of this work.

We start with the description of cloud computing in section 2.1. We provide a short description of this relatively novel computing model using the definition provided by NIST. Then we continue with deployment and service models for cloud computing.

In section 2.2 we provide background information on OpenStack project. We introduce a reader to a history of OpenStack; give a short overview of the projects that constitute OpenStack family; look at evolution of OpenStack through the first releases. Afterwards, we describe test deployment of OpenStack in a virtualized environment that we have used for our analysis of security issues in this work.

2.1 Cloud Computing

2.1.1 Definition

When it comes to the definition of cloud computing, most of the researchers cite the document "The NIST Definition of Cloud Computing" provided by NIST, where cloud computing is defined as *"a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [24].

To refine the definition of cloud computing, the following essential characteristics are denoted [24]:

- **On-demand self-service.** A customer may alter (increase or decrease) the amount of desirable computing resources automatically without the human interaction with the provider.

- **Broad network access.** Access to the computer resources is done over the network using standard mechanisms.
- **Resource pooling.** Provider's resources are shared between numerous customers; different physical and virtual resources are dynamically assigned and reassigned according to consumer demand. As a mechanism for enabling resource pooling, *virtualization* (abstraction of resources, such as CPU, memory, database, etc. in a way that several virtual resources utilize common physical one) is often employed.
- **Rapid elasticity.** Provider can rapidly provision more (scale out) or release (scale in) computing resources according to the customer demand. From the customer point of view, resources available for provisioning are unlimited and can be purchased at any time.
- **Measured Service.** Provider measures the amount of consumed resources by each customer, thus enabling the *pay as you go* pricing model.

Researchers from Cloud Security Alliance (CSA) note that NIST definition lacks another essential characteristic of cloud computing - *multi-tenancy* [3], which implies intermixture of data from different customers on the same storage medium. However, in our opinion, multi-tenancy is implicitly present as the characteristic of *resource pooling*, which is why we do not classify it as a separate characteristic.

2.1.2 Deployment Models

Based on the access to cloud infrastructure, researchers typically distinguish between the following cloud deployment models [24]:

- **Private cloud.** Cloud infrastructure is utilized by a single organization. It can be managed by the organization itself, or by a third-party provider.
- **Community cloud.** Cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns.
- **Public cloud.** Cloud infrastructure is made available to the general public.
- **Hybrid cloud.** Cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities, but are bound together. As stated in [17], by using hybrid cloud, the complexity increases, and special care should be given to secure the links between composed infrastructures.

Depending on the utilized cloud deployment model, organizations have different level of control over the infrastructure and computing resources (private cloud deployment gives greater control than public one) and thus different level of responsibility for applied security measures.

2.1.3 Service Models

Based on the delivery type, researchers typically distinguish between the following service models [24]:

- **Software as a Service (SaaS).** Customer can use provider's applications which are running on a cloud infrastructure. Access to an application can be done from various client devices (PCs, mobile phones, PDAs, etc.) Examples: Google Docs, Microsoft Office Live, Salesforce Customer Relationship Management.
- **Platform as a Service (PaaS).** Customer can use provider's development environment to create applications and deploy them on provider's cloud infrastructure. Examples: Google App Engine, Microsoft Azure, Salesforce Force.com.
- **Infrastructure as a Service (IaaS).** Customer can use provider's computing resources (processing, storage, networks, etc.) to deploy and run arbitrary software which can include operating systems and applications. Examples: Amazon Simple Storage Service (S3), RackSpace Cloud Servers.

2.2 OpenStack

OpenStack is an open source software to build private and public clouds [39]. The mission 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" [7]. The U.S National Aeronautics and Space Administration (NASA) and Rackspace, an IT hosting company and cloud service provider, were the founders of the project. The community support has over 50 participating members including companies like AMD, Citrix, Dell, Intel [35], Microsoft [25] and Cisco [76].

2.2.1 History

NASA was building a private cloud on top of Eucalyptus - an open source clone of Amazon's EC2 compute cloud and S3 storage cloud [28]. However, according to the CTO of NASA, Eucalyptus did not scale as well as NASA hoped, and it was not as open as the agency wanted it to be [26]. Because of the above issues, NASA decided to build a cloud controller from scratch and started a project named "Nova", which was later released under open source license [28].

At about the same time when NASA open sourced its Nova project, Rackspace was preparing to do the same with its both compute engine and storage controller. Rackspace made business by providing cloud compute and storage services and operated at a scale which allowed not to worry about open sourcing the software underneath their services [26].

Two companies decided to cooperate in efforts to create an open source cloud infrastructure software. In July 2010 the launch of OpenStack was announced [57] and in October of the same year the first release named "Austin" went public.

2.2.2 Projects

OpenStack consists of different projects that have the same aim of delivering "a massively scalable cloud operating system" [39]. In the given section we describe projects that exist in OpenStack family.

OpenStack Object Storage. OpenStack Object Storage is an open source software for "creating redundant, scalable object storage using clusters of standardized servers to store petabytes of accessible data" [38]. The codename for the project is "Swift", which is the original name under which the project was originally developed by Rackspace. Swift can be used as a back-end for storing emails, photographs, documents, etc. Object Storage provides a ReSTful API for uploading and downloading files into the cluster. A number of language-specific libraries were provided by Rackspace to be used on top of base API, including support for Java, C#, PHP, Python and Ruby [34].

In our discussion of security issues in OpenStack, we have concentrated on the OpenStack Object Storage project.

OpenStack Compute. OpenStack Compute aims to "provision and manage large networks of virtual machines, creating a redundant and scalable cloud computing platform" [36]. "Nova" - the original name which project had while being developed at NASA - is the codename of the project. Nova provides the main part of every IaaS system - cloud computing fabric controller [43]. An important idea behind Nova is to be hardware and hypervisor agnostic. Currently, the following virtualization technologies are supported: Microsoft Hyper-V, KVM, QEMU, UML, Xen, Citrix XenServer, VMWare and LXC containers [50].

OpenStack Image Service. OpenStack Image Service was created to "provide discovery, registration, and delivery services for virtual disk images" [37]. So as previous projects from OpenStack family, OpenStack Image Service also received a codename - "Glance". Glance provides a ReSTful API for manipulating VM images [42]. Images can be stored in different back-end stores, including OpenStack Object Storage. Image registry allows multiple formats, including the following: Raw, VHD (Hyper-V), VDI (VirtualBox), qcow2 (Qemu/KVM), VMDK (VMWare), OVF (VMWare, others) [37].

2.2.3 Releases

In the given section we look at the evolution of OpenStack based on the release history.

Austin. The first release of OpenStack was named Austin and released in October 2010 [70]. The initial release consisted of two projects: Object Storage and Compute. While Object Storage was claimed to be ready for production in Austin, the initial release of Compute was intended primarily for testing and limited deployment [70].

Bexar. The second release of OpenStack got name Bexar and was released in February 2011 [51]. In Bexar release, a new project was introduced to OpenStack family - OpenStack Image Service. Beside adding new project, a number of changing were made in Swift and Nova. For OpenStack Object Storage, the possibility to store files larger than 5Gb and a new authentication and authorization service named "swauth" were added. OpenStack Compute project received a number of enhancements in the API and added support for new hypervisors and supported disk images [51].

Our analysis of security issues in OpenStack is based on Bexar release.

Cactus. The third and latest OpenStack release, as of the time of this writing, appeared in April 2011 under name Cactus [50]. No new projects were added to Cactus release other than three already existing projects. OpenStack Object Storage introduced the option to serve static website content using container listings, improved requests performance through refactoring of the proxy service, added support for content checksum validation during object GET actions. OpenStack Compute continued improving API and added two additional virtualization technologies - LXC containers and VMWare. Glance introduced new command-line interface tool for direct access to its services, added support for multiple image formats, and image verification against a client-provided checksum to ensure integrity of the transfer [50].

Diablo. The fourth release of OpenStack is expected to get a name Diablo [50]. Database-as-a-Service project under name "RedDwarf" is actively discussing within OpenStack community at the time of writing, and it might become a part of Diablo release. RedDwarf project will allow users to "utilize the features of a relational database without the burden of handling complex administrative tasks" [71]. Users will obtain a possibility to manipulate databases, tables, users via a public API accessible over HTTP.

2.2.4 Test Deployment

For our analysis of security issues in OpenStack, we concentrated on the Object Storage (swift) project. In order to test, how various security issues are handled in OpenStack Object Storage, we installed it in a virtualized environment that emulates a cloud data center. This section describes the selected installation architecture.

In Figure 2.1 on page 11 we show how OpenStack Object Storage was installed on a Windows 7 host machine with the help of VirtualBox virtualization software. We put Authentication and Proxy servers on one virtual machine, and added three Storage nodes for storing user files. Each of the virtual machines had Ubuntu 10.04 LTS installed as the operating system, which was a prerequisite for installing OpenStack.

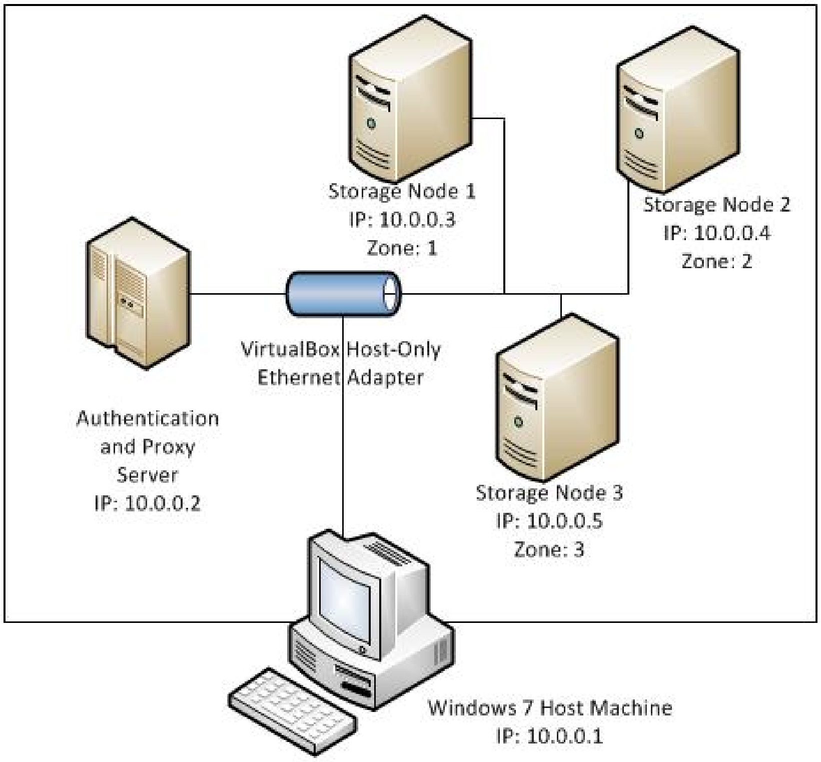


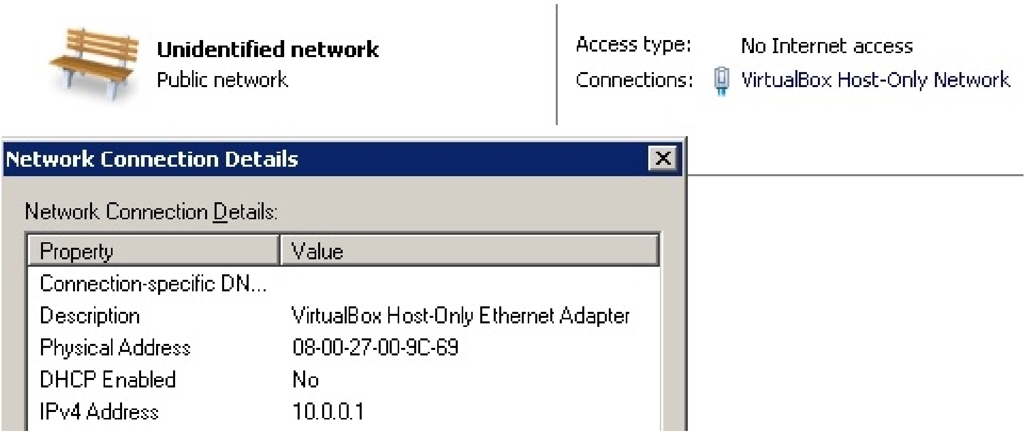
Figure 2.1: Installation of OpenStack Object Storage in a Virtualized Environment

To be able to analyze how OpenStack works, we needed the possibility to intercept network traffic between Proxy machine and Storage nodes. In order to do this, we decided to use VirtualBox Host-Only Ethernet Adapter for enabling networking between the nodes (see Figure 2.2 on page 12). As a result, we received a virtual network adapter on Windows machine, using which we are able to capture network packets with Wireshark protocol analyzer [81].

When deploying OpenStack Object Storage, we created three zones for Storage servers. According to the documentation, "Zones can be used to group devices based on physical locations, power separations, network separations, or any other attribute that would lessen multiple replicas being unavailable at the same time" [41]. We decided to use separate zones for separate virtual machines (storage nodes), since it would follow the idea to use zoning to achieve greater efficiency from replication.

In our deployment of OpenStack Object Storage (see Figure 2.1), we generally followed the example installation architecture from Administrator Guide [46]. Authentication and Proxy services were installed on the same machine. We introduced two changes to the example installation:

1. With the networking mode that we selected, it was technically possible to connect from the outside world to each of the Storage servers. In a real-world deployment, only proxy node should be able to communicate with Storage nodes. As was


Unidentified network
Public network

Access type: No Internet access
Connections: VirtualBox Host-Only Network

Network Connection Details

Network Connection Details:

Property	Value
Connection-specific DN...	
Description	VirtualBox Host-Only Ethernet Adapter
Physical Address	08-00-27-00-9C-69
DHCP Enabled	No
IPv4 Address	10.0.0.1

Figure 2.2: VirtualBox Host-Only Networking on Windows

described above, we needed to intercept network traffic between Proxy and Storage nodes for the sake of our analysis, which is why we made this deviation.

2. We used only three zones, instead of recommended five. Having three zones was enough to test how security issues are mitigated in OpenStack. If we wanted to add more zones, we would have needed to add new virtual machines, which would require additional resources on the host machine, and increase administrating costs, but did not contribute to the quality of analysis.

We provide configuration files that we used with our deployment in Appendix F.

Chapter 3

Cloud Computing Security Issues

In order to make a thorough analysis of the OpenStack software from a security viewpoint, first and foremost, we need to identify security-related issues that should be taken care of when using cloud computing solutions. To do such this, we study and then compare documents that aim to aid companies and organizations in transition to cloud services.

For our comparison we selected cloud computing security-related documents created by an organization that consists of industry representatives - Cloud Security Alliance (CSA), and two government-funded institutions from Europe and the USA: European Network and Information Security Agency (ENISA), and National Institute of Standards and Technology (NIST).

In the sections below, we present a short overview of the documents from CSA, ENISA, and NIST. We arrange the sections according to the chronological order when documents were first published. Afterwards, we make a comparison of the issues defined in three studied documents and present the results of our analysis.

3.1 Security Issues Identified by CSA

Cloud Security Alliance is a non-profit organization formed in late 2008 after an initiative to secure cloud computing was announced at information security conference [6]. The founding members were mainly industry representatives, and, as of now, most of well-known IT companies that operate in cloud computing field are corporate members of CSA, including Google, Microsoft, IBM, Salesforce.com, VMware, etc [6].

Security issues identified by CSA are available in "Security Guidance for Critical Areas of Focus in Cloud Computing" document [3] (in the remainder of this section we refer to this document as *Guidance*). The first version of *Guidance* was published in April 2009, but for our analysis we use version 2.1 from December 2009, latest available version at the time of writing.

The *Guidance* starts with an editorial note, shortly describing the steps necessary to determine whether a company is ready to perform transition to cloud services, which consist of the following items: identifying and evaluating assets for deployment into cloud, mapping assets to available deployment models, assessing potential cloud providers, and sketching data flows [3]. Security issues are presented afterwards (in the document they are named domains). After a short description of each issue, recommendations for mitigating it are provided.

The initial domain is named "Architectural Framework" and contains overview information about cloud computing, as well as distinctions between different types of cloud service models from a security perspective (for a background information on cloud service models please refer to section 2.1.3 on page 8). A summary of the remaining issues is given in Table 3.1 on page 15.

Security issues defined in *Guidance* are divided into two top-level categories: governance (items 1-5 in Table 3.1) and operational (items 6-12 in Table 3.1). As specified in the document, governance domains address "strategic and policy issues", and the operational domains deal with "tactical security concerns and implementation within the architecture" [3].

A short discussion of the security issues from Table 3.1 is given below. For a full description, we refer the reader to [3].

Governance (item 1 from Table 3.1) and Enterprise Risk Management deal with organizational processes for information security. Service Level Agreements (SLA) are discussed in detail, stating that measurable security requirements should be specified as part of SLA and be contractually enforceable. For the risk management, it is advised not only to assess cloud service provider itself, but also the dependencies on which provider relies (e.g. third-party CSPs). Besides, it is recommended that a part of cost savings from utilization of cloud computing is invested into continuous security assessments of CSP [3].

Items 2 and 3 from Table 3.1 discuss compliance with governmental regulations and legal issues (intellectual property, subpoena responses, etc.) Information Lifecycle Management (4) describes various issues related to data manipulation (creation, storage, usage, sharing, archiving and deletion). Compartmentalization and isolation of data in multi-tenant cloud environments are also discussed here. Discussion of Portability and Interoperability (5) starts with specifying possible reasons for switching CSP. It is also stated that currently cloud computing field lacks interoperability standards, which is why transitioning between cloud providers might be not an easy process.

In the item 6 from Table 3.1 CSA researchers discuss insider threats, business continuity, and disaster recovery. Data Center Operations (7) discussion deals mainly with issues to consider during selection of CSP and recommends to pay attention to audits, compartmentalization, handling of resource democratization to predict availability and performance, patch management, and service desk support. In the discussion of next issue (8), researchers from CSA recommend to become aware with tools for incident detection and analysis that provider uses; check whether it is possible to get access to log files

Security Issue	Discussed Sub-Issues
1. Governance and Enterprise Risk Management	1.1 Governance 1.2 Enterprise Risk Management 1.3 Information Risk Management 1.4 Third Party Management
2. Legal and Electronic Discovery	
3. Compliance and Audit	
4. Information Lifecycle Management	4.1 Data Security 4.2 Data Location 4.3 Data Remanance 4.4 Data Commingling 4.5 Data Backup/Recovery 4.6 Data Discovery 4.7 Data Aggregation
5. Portability and Interoperability	
6. Traditional Security, Business Continuity and Disaster Recovery	
7. Data Center Operations	
8. Incident Response, Notification and Remediation	
9. Application Security	
10. Encryption and Key Management	10.1 Encrypting Data in Transit 10.2 Encrypting Data at Rest 10.3 Encrypting Backup Data 10.4 Secure Key Stores 10.5 Access to Key Stores 10.6 Key Backup and Recovery
11. Identity and Access Management	11.1 Identity Provisioning 11.2 Authentication 11.3 Federation 11.4 Authorization and User Profile Management
12. Virtualization	

Table 3.1: Cloud Computing Security Issues Identified by Cloud Security Alliance

and retrieve snapshots of the customer's virtual environment for forensics analysis; verify ability of system restoration to previous states.

Item 9 from Table 3.1 Application Security emphasizes on changes to the Software Development Lifecycle that are to be introduced when utilizing cloud computing. Another warning relates to increased difficulties in configuration management. Vulnerability assessment is also recommended. In the discussion of Encryption and Key Management (10) it is strongly advised to secure inter-host communication even within cloud provider's network. OASIS Key Management Interoperability Protocol is mentioned as an emerging standard for key management in the cloud. Issue 11 starts with a warning that organizations will not be able for a long time to operate in the cloud efficiently without a good identity and access management strategy. Apart from sub-issues specified in the Table 3.1, short discussion on Identity-as-a-Service is given.

Finally, virtualization-related issues (12) conclude the discussion of security issues defined by CSA.

3.2 Security Issues Identified by ENISA

The European Network and Information Security Agency (ENISA) is an organization which aims to "enhance the capability of the European Union, the EU Member States and the business community to prevent, address and respond to network and information security problems" [9]. ENISA prepared and published in November 2009 "Cloud Computing Security Risk Assessment" document [5] outlining both security benefits and key security risks of cloud computing (in the remainder of this section we refer to this paper as *Document*).

The *Document* starts with specifying benefits of cloud computing from security point of view. Afterwards, a description of risk assessment process follows. Based on the likelihood of incident scenario and the estimated negative business impact, risk level (low, medium, high) is assigned to an issue. The list of issues follows and contains risks that are specific to cloud computing, as well as those that are not. Since in this work we are interested not in security issues in general, but rather with security issues pertinent to cloud computing, we omit the risks outside cloud domain. Sections on vulnerabilities, assets, and a set of recommendations conclude the *Document*. We have to note that some of the security issues defined by ENISA (for example identity and access management) were discussed not in the risks section, but in the Information Assurance Framework from the Recommendations section (see [5]).

Security issues (or *risks*, as they are named in the *Document*) relevant to cloud computing are shown in Table 3.2 on page 17. Contrary to the documents created by CSA [3] and NIST [17], security issues discussed in *Document* from ENISA are granularly defined, which is why we do not categorize them into sub-issues. As seen from the Table 3.2, all the issues are classified into three top-level categories: Policy and Organizational issues, Technical issues, and Legal issues.

Policy and Organizational Issues
1. Lock-in
2. Loss of governance
3. Compliance challenges
4. Loss of Business Reputation due to Co-Tenant Activities
5. Cloud Service Termination or Failure
6. Cloud Provider Acquisition
7. Supply Chain Failure
Technical Issues
8. Resource Exhaustion
9. Isolation Failure
10. Cloud Provider Malicious Insider
11. Management Interface Compromise
12. Intercepting Data in Transit
13. Data Leakage on Up/Download
14. Insecure or Ineffective Deletion of Data
15. Distributed Denial of Service
16. Economic Denial of Service
17. Loss of Encryption Keys
18. Undertaking Malicious Probes or Scans
19. Compromise Service Engine
20. Conflicts between Customer Hardening Procedures and Cloud Environment
Legal Issues
21. Subpoena and e-Discovery
22. Risk from Changes of Jurisdiction
23. Data Protection Risks
24. Licensing Risks

Table 3.2: Cloud Computing Security Issues Identified by ENISA

A short discussion of the security issues from Table 3.2 is given below. For a full description, we refer the reader to [5].

Lock-in (item 1 from Table 3.2), or data and service portability issue, start the list of Policy and Organizational risks identified by ENISA. Portability issues are discussed in context of a used cloud service model (for a background information on cloud service models please refer to section 2.1.3 on page 8). Afterwards, Governance (2), Compliance (3) and indirect threats from other customers that are using the same CSP are discussed (4). Availability challenges that might be the result of CSP's own problems (5-6), or the third-party it relies on (7), conclude Policy and Organizational risks identified by ENISA.

The list of Technical issues (items 8-20 from Table 3.2) starts with availability threats from resource exhaustion (8). Failures of mechanism to isolate shared resources between customers (VM monitor vulnerability) continue in the list (9). Insider threats are presented in item 10. Some of the presented risks can be grouped into more broad issues. For example Management Interface Compromise (11) and Compromise Service Engine (19) deal with application security, namely vulnerabilities in software that is used for providing cloud services. Two analogous threats discuss network data interception, both within CSP infrastructures (12) and on the link customer-to-CSP (13). Problems related to secure deletion of data on all the storage mediums are specified in issue 14. Denial of Service threats are also presented as two risks, where an attack to cause availability problems (15) is separated from a case when extra traffic from malicious service requests can increase the cost of cloud services for a customer, thus having negative economical impact on its business (16). Other technical issues deal with management of encryption keys (17); malicious network scans internal to cloud infrastructure (18); insufficient efforts by a customer to secure execution environment (20).

The list of Legal issues (items 21-24 from Table 3.2) starts with procedures to respond to subpoena and e-discovery processes (21). The other three identified issues deal with various data manipulation concerns, such as data location compliance (22), data protection compliance (23), and risks from losing intellectual property when data is stored in the cloud (24).

At this point we finish our summary of the security issues identified by ENISA. As we have noted previously, Information Assurance Framework that follows in the *Document* after security risks, contains additional information in regard to security issues.

3.3 Security Issues Identified by NIST

National Institute of Standards and Technology prepared document with guidelines for security and privacy in cloud computing [17] in response to the United States Chief Information Officer request to accelerate governmental adoption of cloud services [2]. The draft version of the "Guidelines on Security and Privacy in Public Cloud Computing" (in the remainder of this section we refer to this document as *Guidelines*) was published on January 2011, and one month period was reserved to accept comments and suggestions from the general public [2]. To facilitate discussion about cloud computing, a collaboration web-site was set up by NIST [31].

The purpose of the *Guidelines* is to "provide an overview of public cloud computing and the security and privacy challenges involved" [17]. NIST states that organizations have greater level of control when private cloud deployment model is used, which is why the document concentrates on security issues for public clouds [17] (for a background information on cloud deployment models please refer to section 2.1.2 on page 7).

The document gives an overview of cloud services, discusses advantages and disadvantages of cloud model from a security viewpoint, outlines key security and privacy issues, and then provides recommendations on outsourcing data and applications to CSP.

In this section we are interested in security issues identified by NIST in the *Guidelines*. Contrary to the other discussed documents from CSA and ENISA (see section 3.1 on page 13 and section 3.2 on page 16 respectively), NIST does not make a top-level classification of security issues (i.e. organizational, technical, etc.) However, each of the 9 identified issues groups many relative sub-issues, which other documents usually specify as separate items. A summary of the issues identified by NIST is given in Table 3.3 on page 19.

Security Issue	Discussed Sub-Issues
1. Governance	
2. Compliance	2.1 Data Location Compliance 2.2 Electronic Discovery
3. Trust	3.1 Insider Access 3.2 Data Ownership 3.3 Composite Services 3.4 Visibility 3.5 Risk Management
4. Architecture	4.1 VM Monitor Protection 4.2 Virtual Network Protection 4.3 Ancillary Data Protection 4.4 Client-Side Protection 4.5 Server-Side Protection
5. Identity and Access Management	5.1 Authentication 5.2 Access Control
6. Software Isolation	6.1 VM Monitor Quality 6.2 Threats from Co-Tenant VMs
7. Data Protection	7.1 Data Isolation 7.2 Secure Deletion of Data
8. Availability	8.1 Outages 8.2 DoS Attacks 8.3 Availability Threats from Data Collocation
9. Incident Response	

Table 3.3: Cloud Computing Security Issues Identified by NIST

A short discussion of the security issues from Table 3.3 is given below. For a full description, we refer the reader to [17].

Issues 1 and 2 from Table 3.3 deal with Governance, Compliance, and Audit. Discussion of Trust (3) contains wide range of topics, including the following issues: insider threats, both from personnel and other co-tenant customers that use the same CSP (3.1); maintaining data ownership and intellectual property rights over all data residing in the cloud (3.2); assessment of the entities on which CSP relies (3.3); gaining visibility into security controls used by CSP (3.4); risk management (3.5).

Issue on Architecture (item 4 from Table 3.3) discusses software systems that are utilized for cloud computing. The description of this issue starts with a warning that the introduction of virtual machine monitor (hypervisor) increases attack surface in cloud computing, compared to traditional data center operations (4.1). Sub-issue 4.2 states that traffic flowing over virtual networks might not be visible to traditional network security tools (firewalls, intrusion detection/prevention systems, etc.), which might warrant separate deployment of tools dedicated for virtual network monitoring. Virtual machine images and data about cloud service customers might also contain valuable information to attackers, and thus requires adequate protection (4.3). Client-Side Protection (4.4) discusses mostly web browser security, and Server-Side Protection (4.5) deals with server and application security. Besides, prevention of leaks between composed infrastructures, in case when hybrid model of cloud deployment is used (information on cloud deployment models is available in section 2.1.2 on page 7), is discussed in this sub-issue.

Identity and Access Management (item 5 from Table 3.3) focuses on usage of SAML for authentication and XACML for access control. Software Isolation (6) discusses threats related to multi-tenancy. Data Protection (7) discussion starts with highlighting differences between multi-tenant model, where data from different customers is intermingled in single data repository, and multi-instance model, where each customer is using separate data repository and has administrative control over it (7.1). Afterwards, secure deletion of data is discussed (7.2). In the discussion of Availability (8) our attention was brought to the indirect threats from sharing the same CSP with an organization that has a high probability of becoming a target for DoS attacks (8.3). As a consequence of multi-tenancy, it is possible that such attack might result in availability problems for your own service. Finally, issue 9 deals with Incident Response.

3.4 Comparison of Security Issues Identified by CSA, ENISA, and NIST

In this section we provide a comparison between the security issues identified in analyzed documents from CSA (see section 3.1 on page 13), ENISA (see section 3.2 on page 16), and NIST (see section 3.3 on page 18). The reader has to be aware that none of the issues discussed below was specified by us, but rather the whole list of issues was compiled after studying the above documents.

Inspired by the top-level classification into operational, technical, and legal issues from ENISA document, we separate the discussion of the security concerns into two subsections: Policy, Organizational, and Legal Issues (see section 3.4.1 on page 21), and Technical Issues (see section 3.4.2 on page 22).

3.4.1 Comparison of Policy, Organizational, and Legal Security Issues Identified by CSA, ENISA, and NIST

In this subsection we look at the comparison of policy, organizational, and legal issues identified in the documents from CSA, ENISA, and NIST. The results of the comparison are provided in Table 3.4.1 on page 21 and discussed below.

Policy, Organizational, and Legal Issues	**Sub-Issues**	**CSA**	**ENISA**	**NIST**
1. Governance and Risk Management		+	+	+
2. Compliance and Audit	2.1 Laws and Regulations Compliance	+	+	+
	2.2 Data Location Compliance	+	+	+
3. Legal and Electronic Discovery		+	+	+
4. Insider Threats		+	+	+
5. Data Handling	5.1 Protection of Data about Customers	-	-	+
	5.2 Intellectual Property	+	+	+
	5.3 Availability of Data for Forensics Analysis	+	+	+
6. Incident Reporting		+	+	+
7. Patch Management		+	+	+
8. Assessment	8.1 Assessment of CSP	+	+	+
	8.2 Assessment of CSP's Dependencies	+	+	+
Note: '+' denotes that a given issue is discussed in the document, while '-' means that it is omitted				

Table 3.4: Comparison of Policy, Organizational, and Legal Security Issues Identified by CSA, ENISA, and NIST

We start our policy, organizational, and legal issues with Governance and Risk Management (1), which deals with policies, processes, and procedures to maintain information security. Compliance (2) with official Laws and Regulations (2.1), as well as appropriate audit procedures, follow in our list. Since cloud services involve redundant storage in multiple locations, a situation where sensitive information stored in the cloud leaves the physical borders of one country and is copied to servers in another country is possible, which might be a violation of Data Location Compliance (2.2) requirements. Even though not directly specified as a risk according to ENISA, data location discussion is still present in the document from ENISA afterwards, which is why we make a conclusion that the first two issues were identified in all the analyzed documents.

Another topic in the list of policy, organizational, and legal issues concerns subpoena responses and e-discovery (3), which is followed by Insider Threats (4). These two issues are mentioned in all the analyzed documents, which is also the case for most of the other issues, including Incident Reporting (6), Patch Management (7), and Assessment of CSP's adherence to the declared responsibilities (8). The only difference we have found between the documents concerns handling of data *about* cloud customers (5.1). While CSA and ENISA mention protection of data which belongs to the customer (for example, data that customer decided to store in the cloud), only NIST emphasizes the need of appropriate protection of data about customer itself (for example, stolen contact data can be used in subsequent social engineering attacks) [17]. Other Data Handling issues, including Intellectual Property (5.2) and Availability of Data for Forensics Analysis (5.3), are mentioned in all the analyzed documents.

To sum up the results obtained from comparing policy, organizational, and legal issues discussed in the analyzed documents, we will say that only one difference was found in the discussion of Protection of Data about Customers (5.1).

3.4.2 Comparison of Technical Security Issues Identified by CSA, ENISA, and NIST

In this subsection we look at the comparison of technical issues identified in the documents from CSA, ENISA, and NIST. The results of the comparison are provided in Table 3.4.2 on page 23 and discussed below.

As seen in Table 3.4.2, we start the list of technical issues with Availability (1). We select the following sub-issues in regard to availability: Outages (1.1), Resource Exhaustion (1.2), Availability Threats from DoS Attacks (1.3), Availability Threats from Data Collocation (1.4). Outages (1.1) may be caused by hardware or facility (e.g. electricity) failures, or even natural disasters, and are discussed in all the documents that we analyzed. Resource exhaustion (1.2) can happen when CSP does not adequately predict how much resources the customers will need, and demand from the latter exceeds supply from the former. In theory, resource provisioning should be done automatically, however, as CSA warns, "cloud computing theory still somewhat exceeds its practice: many customers make incorrect assumptions about the level of automation actually involved" [3]. During our analysis we have found out that NIST document does not mention resource under-provisioning issue in availability discussion. Denial of Service attacks (1.3) are another threat to availability, which is present in ENISA and NIST documents, but omitted in CSA discussion. Besides, in ENISA document, a type of a DoS attack is described when malicious requests are used not to cause availability problems, but to increase the cost of service in case when payment depends on the number of served requests. Finally, indirect threats from other customers that are using the same CSP are considered in all three documents. An example of such a threat would be a DoS-attack on the neighbor customer with whom resources are shared, and the resulting insufficiency of resources for your own service.

Technical Issues	Sub-Issues	CSA	ENISA	NIST
1. Availability	1.1 Outages	+	+	+
	1.2 Resource Exhaustion	+	+	-
	1.3 Availability Threats from DoS Attacks	-	+	+
	1.4 Availability Threats from Data Collocation	+	+	+
2. Portability	2.1 Data Portability	+	+	+
	2.2 VM Image Portability	+	+	-
	2.3 Application Portability	+	+	+
3. Data Management	3.1 Isolation	+	+	+
	3.2 Backup and Recovery	+	+	+
	3.3 Deletion	+	+	+
	3.4 Encryption	+	+	+
	3.5 Key Management	+	+	+
	3.6 Integrity Verification	+	+	+
4. Identity and Access Management	4.1 Identity Provisioning and Deprovisioning	+	+	-
	4.2 Identity Federation	+	+	+
	4.3 Authentication	+	+	+
	4.3 Authorization and Access Control	+	+	+
5. Application Security	5.1 Client-Side Protection	+	+	+
	5.2 Server-Side Protection	+	+	+
	5.3 VM Image Protection	+	+	+
	5.4 Log-Files Protection	+	+	-
6. Virtualization	6.1 VM Hypervisor Protection	+	+	+
	6.2 Guest OS Protection	+	+	+
	6.3 Virtual Network Protection	+	+	+
Note: '+' denotes that a given issue is discussed in the document, while '-' means that it is omitted				

Table 3.5: Comparison of Technical Security Issues Identified by CSA, ENISA, and NIST

Portability issue (2) continues our list of technical issues. As stated in [3], customers may decide to switch a provider of cloud services because of various reasons, for example increase in cost, decrease in quality, or even bankruptcy of CSP. Of course, the portability questions to consider vary with the cloud service model used (SaaS, PaaS, IaaS). To be able to transfer data and application to another CSP, customer will need a sufficient level of data (2.1) and application (2.3) portability. In case of IaaS service offering, portability of VM images (2.2) should be considered as well, since some providers might use non-compliant extensions to VM images [3]. Based on our evaluation of the documents from CSA, ENISA and NIST, we may conclude that NIST does not develop portability issue: data and application portability is mentioned in one sentence during the discussion of data protection. The portability of VM images is not present in NIST document at all.

Data Management (3) is the next concern we look at in our compilation of technical issues relevant to cloud computing. The importance of data isolation in a multi-tenant environment (3.1) and secure data deletion (3.3) is discussed in all the analyzed documents. Backup and Recovery (3.2) issue is given the most comprehensive level of attention in CSA document, where, among the others, the importance of data encryption on backup media and secure data deletion on all the backup copies concerns are raised [3]. Encryption of data-in-transit, data-at-rest, and backup data (3.4); Key Management, including creation, storage, backup/recovery of keys (3.5), and Integrity Verification (3.6) conclude our list of data management issues and are described in all the analyzed documents.

In the Identity and Access Management (4) topic we have selected the following sub-issues to be considered: Identity Provisioning/Deprovisioning (4.1), Identity Federation (4.2), Authentication (4.3), Authorization and Access Control (4.4). While comparing different documents we have found out that automated identity provisioning is not discussed in the document from NIST. Besides, we note that in the ENISA document, identity and access management issues are discussed in the Information Assurance Framework section (see [5]).

Next issue focuses on Application Security (5), including the following sub-issues: Client (5.1) and Server (5.2) protection, VM Image Protection (5.3) for IaaS model, and Log-File Protection (5.4). As stated in [3], the information stored in log-files can contain sensitive information which is why it is recommended to pay attention to the management of those files, since the information stored there might belong to various customers sharing resources of one CSP. Our analysis shows that application security issues are discussed in all the documents; however, issues regarding log-file management are absent in NIST document.

Virtualization-related issues (6) conclude our list of technical issues pertinent to cloud computing. The importance of appropriate Hypervisor Protection (6.1), hardening of guest operating system in case when IaaS service model is used (6.2), and Virtual Network Protection (6.3) issues are addressed in all the analyzed documents.

To sum up the results obtained from comparing technical issues discussed in the analyzed documents, we will say that differences were found in the discussion of the following sub-issues: Resource Exhaustion (1.2), Availability Threats from DoS Attacks (1.3), VM Image Portability (2.2), Identity Provisioning/Deprovisioning (4.1), and Log-Files Protection (5.4).

Summary

In this chapter we aimed to find out which security issues should be taken care of when using cloud services. In doing this we looked into three documents that were created to facilitate adoption of cloud computing by a Cloud Security Alliance (CSA), an organization consisting of industry representatives, and two governmental institutions: European Network and Information Security Agency (ENISA), and National Institute of Standards and Technology (NIST).

During our analysis we found out that some issues were mentioned in one document, but missing in the other documents (see Table 3.4.1 on page 21 and Table 3.4.2 on page 23 for comparison). However, we conclude that with varying degree of details the main security concerns are discussed in all the analyzed documents; even though, we believe that the study of all the documents contributes to obtaining the complete picture of security issues pertinent to cloud computing.

The results obtained during work on this chapter will be the input to analyzing OpenStack cloud solution from security point of view in the subsequent chapters.

Chapter 4

Identity and Access Management

In this chapter we look into identity and access management implementation in OpenStack Object Storage (sometimes in our discussion we use original name of the project - *swift*). During the study of security issues pertinent to cloud computing in chapter 3, we found out which issues were discussed in regard to identity and access management. In this chapter we provide a separate section for each of the sub-issues defined for identity and access management from Table 3.4.2 on page 23. Besides, where relevant, we also include discussions on portability, application security and insider threats.

We start this chapter with description of Identity Provisioning/Deprovisioning (section 4.1), continue with Identity Federation (section 4.2), Authentication (section 4.3), Authorization and Access Control (section 4.4). The recommendations for developers and users of OpenStack Object Storage that were discovered during our analysis are presented in section 4.5.

4.1 Identity Provisioning/Deprovisioning

4.1.1 Overview

User provisioning is the process of registering new users in the system. Deprovisioning is the reverse process when user is removed from the system. During study of security issues relevant to cloud computing (see Chapter 3), we have seen the necessity to automate user management tasks. Before discussing identity management, we need to get acquainted with authentication/authorization in OpenStack Object Storage.

As stated in the documentation for OpenStack Object Storage, the authentication/authorization system is pluggable and thus not dependent on the project itself. However, two implementations are provided out-of-the-box: devauth and swauth [40]. The difference between the two is the back-end repository where user data (authentication data, access permissions, etc.) is stored.

Devauth uses SQLite database as a user data storage. SQLite is a light-weight database engine that stores data in a single cross-platform file on disk [72]. Among the benefits of storing authentication data in SQLite, one gets the *portability* of data itself - in order to transfer users to another server one will have to copy a single SQLite file. However, relying on SQLite has one significant scalability issue - SQLite allows multiple reads from the database file, but only single write to the file. In practice, this means that using devauth in a system where many users have to be registered simultaneously can cause significant performance slowdown. Despite scalability issues with devauth, this is the authentication system used by default in OpenStack, and the only one described in OpenStack Object Storage administrator guide, which is why we include it into our study of security issues in OpenStack.

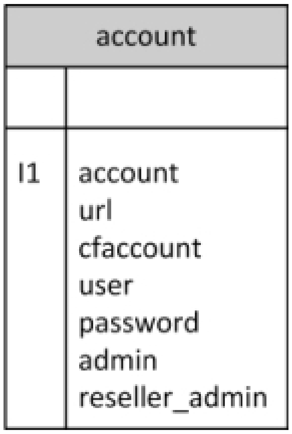
Unlike devauth, swauth claims to be a "scalable authentication and authorization system that uses Swift itself as its backing store" [40]. In order to use swauth, a dedicated Swift account has to be created on a Swift cluster. User information is stored as JSON-encoded data in text files, which are Swift objects.

User provisioning/deprovisioning in OpenStack Object Storage is role-based. Unfortunately, Swift documentation has little information about available roles and their permissions, which is why the source code had to be consulted to extract the information about types of users that exist in OpenStack Object Storage, along with their permissions in regard to user provisioning/deprovisioning (permissions for account, container, and object management will be discussed in section 4.4.1). There are the following roles in OpenStack Object Storage:

- **User** has no permissions relative to user management.
- **Admin** can add users to an account where he is an administrator. In swauth can delete users from administered accounts.
- **Reseller Admin** has *Admin* permissions on *all* the accounts. Cannot add other *Reseller Admins*.
- **Super Admin** is the most powerful user, who can perform all user management procedures, including adding *Reseller Admins*.

4.1.2 Compliance with Industry Standards

OpenStack Object Storage provides the possibility for on demand identity provisioning in both devauth and swauth systems, which is a necessary precondition for automating cloud services. Deprovisioning and update of user profile information is only possible is swauth (in devauth this can be done only via *manual* update of the database). We have to note that timely identity deprovisioning might be viewed as even more important process than its counterpart. For example, a person who has been fired from an organization might possess personal reasons to harm its former employer using insider knowledge and previous access



account	
I1	account url cfaccount user password admin reseller_admin

Figure 4.1: Database with Authentication Information in OpenStack Object Storage

credentials. Thus, such a person should be denied access to the system as soon as possible, which might not be possible in devauth.

OpenStack Object Storage identity management solution is proprietary and does *not* follow any industry standard. In "Security Guidance for Critical Areas of Focus in Cloud Computing" Cloud Security Alliance warns customers against using proprietary solutions for identity provisioning, since it "exacerbate management complexity" [3]. Instead, solutions build on Service Provisioning Markup Language (SPML) schema is recommended to avoid customers being locked to custom solutions. We investigated whether any existing Python libraries that can ease using SPML in OpenStack exist; however, we were unable to find any.

4.1.3 Protection against Elevation of Privileges

A direct threat to user management is the elevation of privileges during adding/removing users to/from a system. Below we provide a summary of our study how this issue is mitigated in OpenStack Object Storage. As we noted in section 4.1.1, access to adding/removing users is done based on user's role. That is why, in this section we look at the possibility to circumvent role checks.

It is worth mentioning that access permissions are specified directly in the code. For example, if *Reseller Admin* has to be added to a system, the code makes a check whether the user executing given action acts in a role of *Super Admin*. There is no other way to delegate the permission to add *Reseller Admins* for some principal, except assigning this principal to *Super Admin* group.

Devauth.

Devauth keeps user information in SQLite database. Database file is stored in `etc/swift/auth.db`. The database table which stores authentication information is shown in Figure 4.1 on page 28. For current discussion we are interested only in the fact that if user has administrator privileges, a textual value of `t` will be present in `admin` or `reseller_admin` columns in the `Account` table.

In devauth, database has to be consulted to check user credentials for identity management whenever user acts in a role of *Admin* or *Reseller Admin*. If user acts in a role of *Super Admin*, his password is checked from the configuration file of the Swift authentication service. Otherwise, database is checked to retrieve user data. As the code that connects to database might be susceptible to SQL injection, we tried to examine source code for such kind of attacks. As an example, we show python code to verify whether user has administrative privileges:

```
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()
```

One might have the feeling that the above code has vulnerabilities, since user data is passed to the SQL code without filtering user input (see `request.headers.get('X-Auth-Admin-Key'))`). However, in the above code parameters are passed to `execute` method in the second argument (line 4 in the above code), and proper escaping is done by the library [56], which in our case is `sqlite3` python module. We have also checked SecurityFocus Vulnerability Database [60] to verify that vulnerabilities do not exist in the sqlite3 library itself.

All the devauth code that makes database communication was checked to verify that developers use only parameterized queries for database access. During our analysis we have not found a possibility to elevate privileges using SQL injections attacks.

Swauth. As it was noted previously, swauth stores user information in OpenStack Object Storage file as JSON-encoded data. An example of the content of such a file is provided in Figure 4.7 on page 34. In this section we are mainly interested how group information is manipulated, since user roles depend on the group membership. As seen from Figure 4.7, user groups are stored in an array of objects with single attribute `name`. The value of this attribute is the name of the group to which user belongs. If the user is a *Reseller Admin*, he will belong to the group `.reseller_admin`. If the user is an *Admin*, he will belong to the group `.admin` (user from Figure 4.7 has *Admin* privileges). As in devauth, *Super Admin* information is stored in configuration file of OpenStack.

As with database storage, it might also be possible to inject values into JSON. A malicious user might aim to provide an input that will successfully inject `.reseller_admin` or `.admin` values into groups array. In order to examine whether this is possible, we have analyzed code that manipulates user data. For example the below Python code creates JSON-encoded string with user data:

```
groups = ['%s:%s' % (account, user), account]
if admin:
 groups.append('.admin')
```

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

Protection against injections is contained 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, than injection attacks would be feasible.

To sum up, we state that during our analysis we have found no vulnerabilities in devauth and swauth systems to elevate privileges for identity provisioning.

4.2 Identity Federation

Identity federation is the process that enables exchange of identity information between identity and service providers [3]. Cloud computing documents that were studied in Chapter 3 recommend using SAML or WS-Federation as enabling technologies [3]. The analyzed version of OpenStack Object Storage does not support identity federation.

We investigated whether any existing Python libraries that can help utilizing SAML in OpenStack exist. PySAML2 [19] provides code to set up Service and Identity providers and can be used as Web Server Gateway Interface (WSGI) middleware. Since authentication system in OpenStack Object Storage is written as WSGI middleware itself [40], we predict that it is feasible to incorporate PySAML2 into OpenStack to add Identity Federation functionality to the software.

4.3 Authentication

4.3.1 Overview

Authentication process is shown in Figure 4.2 on page 31. Both authentication systems available out-of-the-box in OpenStack Object Storage use username/password combination to authenticate users. Upon successful authentication, a user will receive: 1) a token, which will later identify user to the system; 2) an URL to use to access user's account, which is needed to work with OpenStack Object Storage after authentication. Token has configurable expiration time, and the default value is set to 24 hours. Account URL is created upon account registration and not supposed to be ever changed (as was confirmed by the developers, see [44]).

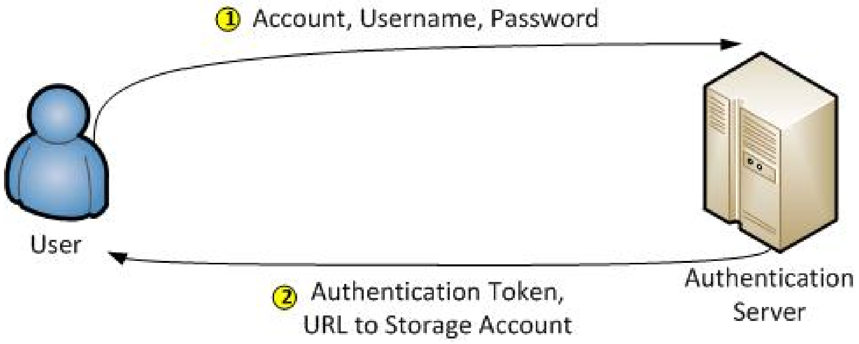

① Account, Username, Password
User
Authentication Server
② Authentication Token, URL to Storage Account

Figure 4.2: Authentication in OpenStack Object Storage

Unfortunately, both available systems are custom implementations specific to OpenStack. This situation might result in a problem when an enterprise will have to use two authentication systems: one internal (organizational) and the other external (cloud-based), which, according to NIST, can only be seen as a temporary solution [17].

4.3.2 Compliance with Industry Standards

All of the cloud computing security documents studied in Chapter 3 emphasize on allowing authentication delegation, for example, by accepting assertions in SAML format. Our evaluation shows that this functionality is *not* available in OpenStack.

Besides, according to "Security Guidance for Critical Areas of Focus in Cloud Computing" from Cloud Security Alliance, authentication service utilized by the cloud provider should be OATH compliant in order to avoid lock-in of authentication information [3]. Currently, the authentication systems provided by OpenStack Object Storage are *not* compliant with OATH, and the intention to implement such a feature in future is in its initial phase, as seen from the OpenStack mailing list discussions (see for example [80]).

As we have already indicated in section 4.2, PySAML2 library can be considered for adding SAML functionality to the project.

4.3.3 Password Strength

Since both authentication systems provided in OpenStack Object Storage use username/-password combination to authenticate users, it becomes necessary to study password strength requirements imposed by these systems. "Electronic Authentication Guideline" created by NIST provides some rules to prevent users from choosing bad passwords, including checking a password against dictionary of commonly used passwords, specifying minimal password length, and requiring to use different characters (lower-case, upper-case, non-alphabetic) [4].

During out study we have found out that OpenStack Object Storage has *no* requirements on password length and selected characters. Likewise, dictionary checks are also not implemented. In our experiments we were able to register users with passwords as short as 1 character only. Because of the lack of password selection checks upon registration, many customers would have either to enhance existing devauth/swauth modules, or to integrate other authentication system with OpenStack Object Storage .

We analyzed whether any existing libraries exist that can facilitate adding password strength checks to OpenStack Object Storage. Python-crack is a module that brings the availability to evaluate password strength in Python [55]. This module uses the cracklib toolkit [30] which is available for Linux. Cracklib allows password evaluation against a number of rules, including dictionary checks. According to the python-crack documentation, the library can be simply integrated into any project, since password check is done with a single API call and exception handling [52]. Because of the ease of use and flexibility, we recommend using python-crack to bring password strength checks to OpenStack Object Storage.

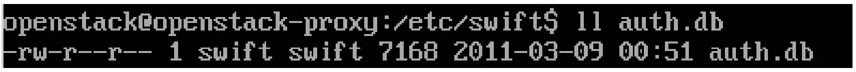
4.3.4 Password Storage

Password storage is not an issue unique to cloud computing, but rather to all information systems that use password authentication. Information Security practices recommend not to store passwords in clear text and make appropriate access control to the password storage medium. In this subsection we look at OpenStack Object Storage approach to store passwords.

Devauth. As we have noted previously, devauth uses SQLite database for password storage. The database file is stored in `etc/swift/auth.db`. By default, each user in the system has read access to this file, as seen on Figure 4.3 on page 33. Such an approach violates the principle of least privilege, and, as a result, allows other users registered in the system to obtain passwords of OpenStack Object Storage users. Besides, password is stored in clear text in `etc/swift/auth.db` file, as shown in Figure 4.4 on page 33 (the registered user has both username and password set to `admin`, with red box we show where the password is stored in the file).

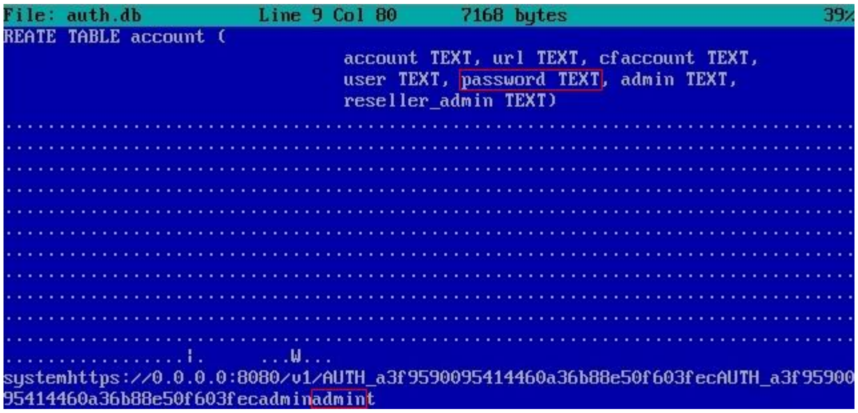
Most probably, because of the scalability issues described in Section 4.3.1, devauth was never considered by the developers to be an option for production deployment of OpenStack Object Storage, which is why it has such poor approach to password storage. However, the documentation does not warn about the weaknesses of devauth, and the only way to find about these weaknesses is to look into source code.

Super User's password is stored differently than the password of other users. The location of *Super User*'s password is `/etc/swift/auth-server.conf` file for devauth, or `/etc/swift/proxy-server.conf` for swauth. The password in this configuration file is stored in clear text, which is why administrator should make sure that default password is changed after installation of OpenStack. Access rights to that file should be changed to allow only the owner of the file read and write access.



```
openstack@openstack-proxy:/etc/swift$ ll auth.db
-rw-r--r-- 1 swift swift 7168 2011-03-09 00:51 auth.db
```

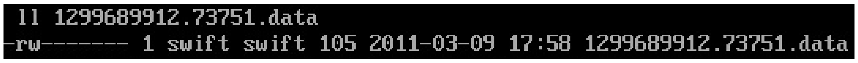
Figure 4.3: Access Rights to the File with Passwords in Devauth Authentication System



```
File: auth.db             Line 9 Col 80       7168 bytes                     39%
REATE TABLE account (
                                           account TEXT, url TEXT, cfaccount TEXT,
                                           user TEXT, password TEXT, admin TEXT,
                                           reseller_admin TEXT)
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
.................!.     ...W...
systemhttps://0.0.0.0:8080/v1/AUTH_a3f9590095414460a36b88e50f603fecAUTH_a3f95900
95414460a36b88e50f603fecadminadmint
```

Figure 4.4: Contents of the File with Passwords in Devauth Authentication System

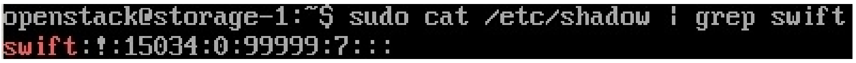
Swauth. In order to analyze password storage in swauth, we found a file with stored password and checked its access rights. As seen in Figure 4.5 on page 33, only the owner of the file, which is `swift` user, can access this file. While trying to check the password for `Swift` user, we looked at `etc/shadow` file. As shown in Figure 4.6 on page 34, the `swift` user can not login manually, as shown by an exclamation mark in the password field for `swift` user. Further analysis showed that `swift` user is the one under which OpenStack Object Storage services are running, which is why only those services can access files with passwords. We conclude that swauth makes appropriate usage of access control to secure password data.



```
 ll 1299689912.73751.data
-rw------- 1 swift swift 105 2011-03-09 17:58 1299689912.73751.data
```

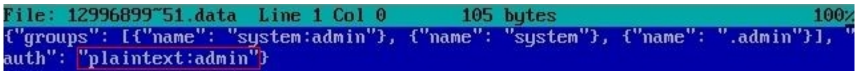
Figure 4.5: Access Rights to the File with Passwords in Swauth Authentication System

Until looking into the contents of the file with password itself, we haven't found any security concerns in swauth password storage. However, as we show in Figure 4.7 on page 34, the password is stored in clear text within this file (we use red box to show, where the password is stored). This gives rise to insider attack, when a system administrator can use UNIX superuser (root) account to find out user passwords. Because of the `plaintext` text specified before the password itself, we checked whether other options are possible. However, manual source code inspection of `common/middleware/swauth.py` showed us that other options (like password hashing before storing in file) are not implemented yet.



```
openstack@storage-1:~$ sudo cat /etc/shadow | grep swift
swift:!:15034:0:99999:7:::
```

Figure 4.6: Password for swift User According to /etc/shadow File



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

Figure 4.7: Contents of the File with Passwords in Swauth Authentication System

Based on our analysis of password storage in OpenStack Object Storage, we conclude that both devauth and swauth authentication systems lack appropriate protection of passwords. In devauth, file with passwords in clear text can by default be accessed by any user in the OS. In swauth, the access to files with passwords is available only to system user, but password is stored in clear text.

A recommendation for devauth and swauth authentication systems taken from NIST "Electronic Authentication Guideline" would be to store passwords "concatenated to a salt and/or username and then hashed with approved algorithm so that the computations used to conduct a dictionary or exhaustion attack on a stolen password file are not useful to attack other similar password files" [4]. Python module hashlib [53] can be used to hash password with SHA-256 algorithm before storing it. If salt value is used (not a username), it can be stored alongside hashed password.

4.3.5 Authentication Tokens

For OpenStack tokens play the same role as session identifiers for web applications - an attacker knowing the token can impersonate the user who received this token from authentication system. Thus, not only token protection over the network is necessary, which is done in case communication is made via TLS, but also the algorithms for token generation come into importance. If it is possible to compute the next generated token by observing previous values, then protecting token in transit does not contribute much to the security of the system. In this section we look at OpenStack Object Storage approach to generate authentication tokens.

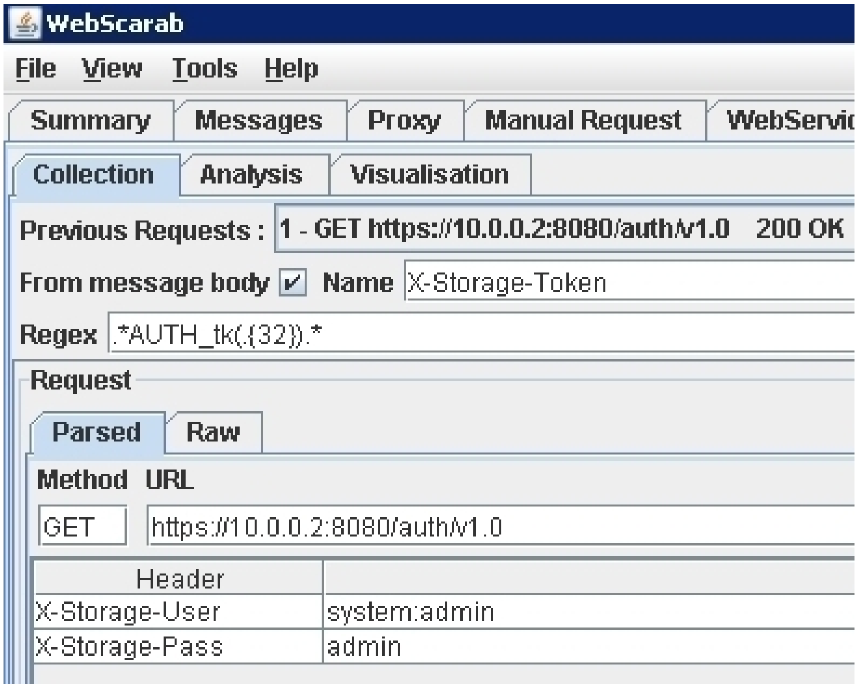
For the analysis of the token generation, we have used the Session ID Analysis feature of OWASP WebScarab tool [48]. WebScarab allows to record a number of session identifiers received via HTTP calls to an application, convert IDs to numbers, and plot them on a graph [47]. By analyzing the graph visually, one can be able to find patterns among generated IDs. Unfortunately, WebScarab does not allow to retrieve identifiers from HTTP headers, which is why we had to modify source code to add this functionality (see Appendix G for the modified source code).

Another tool for analyzing generated tokens is Burp Sequencer from PortSwigger [49]. This tool runs Federal Information Processing Standards (FIPS) tests, analyzing the degree of randomness present in generated security tokens. Even though Burp Sequencer implements tests from the old NIST "Security Requirements for Cryptographic Modules" document (also known as FIPS 140-1) [29], which was afterwards substituted by the NIST publication "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" [59], the outcome from the tests still provides us with additional facts about token generation mechanism used in OpenStack.

To analyze token generation we performed the following experiment using a black-box approach:

1. Set token expiration time to 0 seconds to receive different tokens during consecutive calls to the authentication system.
2. Obtain from the authentication system 10000 tokens generated for the same user.
3. Analyze generated tokens with OWASP WebScarab tool to check whether patterns are present among the generated tokens.
4. Analyze generated tokens with Burp Sequencer tool to check the level of randomness among generated tokens.

Both devauth and swauth authentication systems return generated token in the HTTP response header `X-Storage-Token`. The generated token consists of two parts: a static value `AUTH_tk`, which denotes authentication token, and a dynamic 128-bit value consisting of 32 hexadecimal characters. We split the static and dynamic values before passing the latter to the analysis tools.



Previous Requests : 1 - GET https://10.0.0.2:8080/auth/v1.0 200 OK

From message body ☑ Name X-Storage-Token

Regex .*AUTH_tk(.{32}).*

Method	URL
GET	https://10.0.0.2:8080/auth/v1.0

Header	
X-Storage-User	system:admin
X-Storage-Pass	admin

Figure 4.8: Collecting Authentication Tokens from OpenStack using WebScarab

User interface for Session ID Analysis module is shown in Figure 4.8 on page 35. Our modification to the tool did not affect the UI, only the logic to search tokens in the HTTP response was changed by our code (see Appendix G for details). As seen in the figure, we provide username and password in the `X-Storage-User` and `X-Storage-Pass` header fields respectively, and send HTTP request to the authentication service. We also specify the name of the header where the generated token is present (`X-Storage-Token`) and provide regular expression used to retrieve dynamic part of the token (`.*AUTH_tk(.{32}).*`).

The results of token analysis are shown in Figure 4.9 on page 36. The tool converted each token to an integer number according to an algorithm specified in [47]. Then each token was plotted on a graph and shown with a small red square. For the human eye it is impossible to see any patterns among the generated tokens, which is why we can state that token generation algorithm utilized by OpenStack Object Storage seems to be unpredictable according to OWASP WebScarab tool.

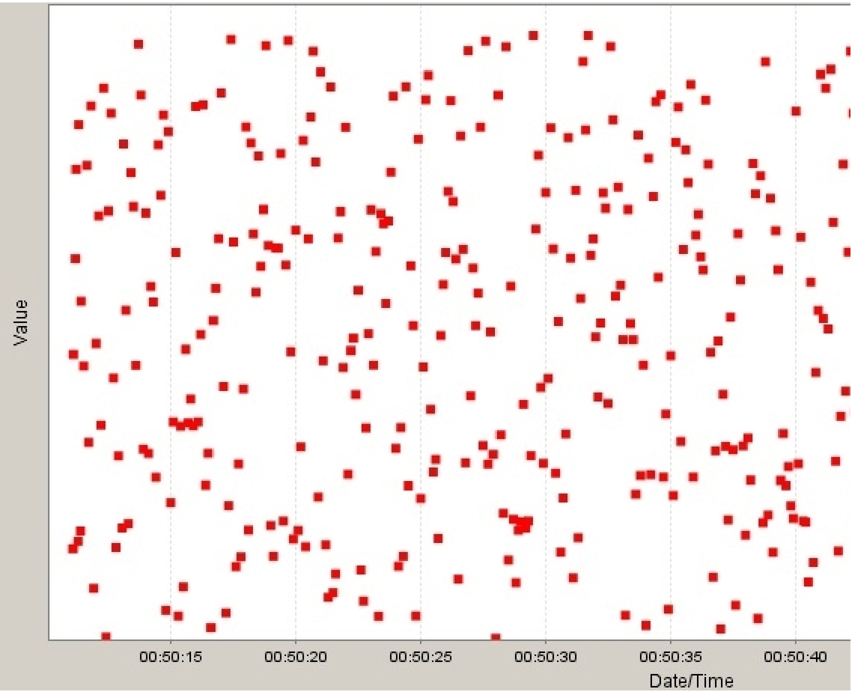


Figure 4.9: Visualizing Collected Authentication Tokens from OpenStack using WebScarab

Afterwards, we exported authentication tokens from WebScarab to Burp Sequencer and run randomness tests on the token data. According to the results from the tool, the overall quality of randomness within the sample is estimated to be excellent. However, the character level analysis (see Figure 4.10 on page 37), which gives the distribution of characters used at each position of the token, showed that character at position 12 does not contribute to the randomness of the data. By looking at the generated tokens, we found out that character at this position always has a value `4`.

We used both devauth and swauth to test for token generation and found no differences

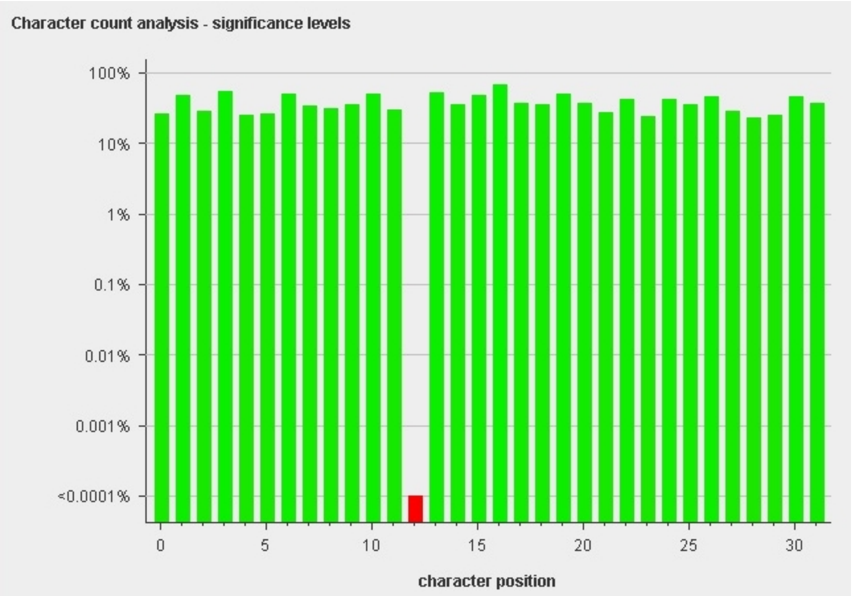


Figure 4.10: Character-Level Analysis of Collected Authentication Tokens from OpenStack using Burp Sequencer

among the results. In a variation of the above experiment we also generated tokens when different users were making requests to the authentication service at the same time; however, the obtained results did not differ from the previous run of the experiment.

After performing black-box approach we switched to white-box testing by looking into OpenStack Object Storage source code. The code for devauth authentication module is present in `auth/server.py` file, while `common/middleware/swauth.py` contains the code for swauth. Both files have the same code to create new token:

```
token = '%stk%s' % (self.reseller_prefix, uuid4().hex)
```

As seen in the above code, the dynamic part of the code is generated using version 4 of Universally Unique Identifiers (UUID) (the static part is created using reseller prefix). According to RFC 4122, "The version 4 UUID is meant for generating UUIDs from truly-random or pseudo-random numbers" [21]. Unlike the other versions of UUID, version 4 contains neither time, nor clock sequence, and specifies the rule for setting only 6 bits to a fixed values, having all the other bits generated randomly (or pseudo-randomly) [21]. Using Burp Sequencer we found out that character at position 12 did not contribute to the randomness of the token. According to RFC 4122, this character represents the version of the used UUID and is fixed for all the generated tokens to the binary string `0100` (decimal `4`). By comparing the results of black-box and white-box approach we increase the level of confidence in the received results.

RFC 4122 contains security notes on UUID, "[UUID] should not be used as security capabilities (identifiers whose mere possession grants access)" [21]. Basically, this means that RFC writers predicted the case when a powerful attacker with high capabilities (intelligence agencies, for example) can synchronize a local pseudo-random number generator with that used on the server and thus predict UUID generated next. In our situation this will mean that an attacker is able to illegally obtain authorization token generated next and pretend being a legitimate user to OpenStack. This will lead to catastrophic consequences making the other security countermeasures irrelevant.

Having the above described threat in mind, we have looked into Python implementation of UUID version 4 to find out how randomness is achieved, and whether the used approach has any weaknesses. The code to generate UUID can be found in `Lib/uuid.py` file of Python source code distribution. While studying the code, we have found out that Python tries to get system implementation of UUID first, which will be the implementation that ships with Ubuntu 10.04 LTS (current version of OpenStack Object Storage has this version of Ubuntu as an installation prerequisite). For this version of Ubuntu, the implementation ships as part of `util-linux` package in `libuuid.so.1.3.0` library.

Further investigation of UUID generation revealed that `/dev/urandom` device was used as a source of randomness in the code. According to the manual pages on Ubuntu for `/dev/urandom`, "The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. The generator also keeps an estimate of the number of bits of noise in the entropy pool. From this entropy pool random numbers are created" [77].

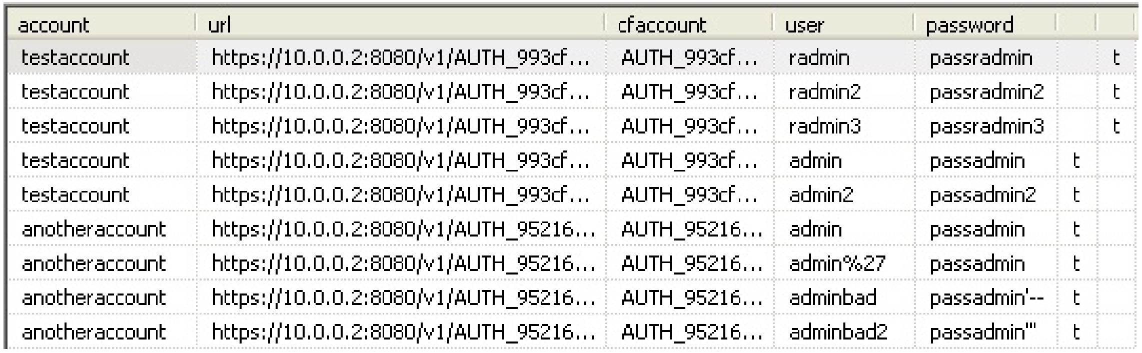
Based on the above, we can sum up that the approach taken to generate UUID uses a solid source of randomness which has no known weaknesses, and thus is deemed to be secure by us.

4.3.6 Portability of Authentication Data

During study of security issues pertinent to cloud computing (see Chapter 3), we have seen the importance of data and application portability. In order to achieve smooth application portability, one needs to transfer authentication data from one provider to another.

Both devauth and swauth authentication systems are custom implementations developed within OpenStack project, which may result in difficulties with data portability. Since those custom implementations do not follow well-known industry standards, the transition to another provider will require additional processing to retrieve authentication data from one provider and persist it with another provider. However, if another provider uses OpenStack software, the portability of authentication data will be easier.

Devauth. In devauth all the authentication data is contained within a `/etc/swift-/auth.db` file. However, there are two obstacles to data migration with devauth. First of all, there is no automated way to retrieve data from `auth.db` by a customer without cooperation with its provider. Second, database stores authentication data for all the customers, which is why a processing will be required on a provider's side to retrieve users for accounts controlled by a customer.



account	url	cfaccount	user	password		
testaccount	https://10.0.0.2:8080/v1/AUTH_993cf...	AUTH_993cf...	radmin	passradmin		t
testaccount	https://10.0.0.2:8080/v1/AUTH_993cf...	AUTH_993cf...	radmin2	passradmin2		t
testaccount	https://10.0.0.2:8080/v1/AUTH_993cf...	AUTH_993cf...	radmin3	passradmin3		t
testaccount	https://10.0.0.2:8080/v1/AUTH_993cf...	AUTH_993cf...	admin	passadmin	t	
testaccount	https://10.0.0.2:8080/v1/AUTH_993cf...	AUTH_993cf...	admin2	passadmin2	t	
anotheraccount	https://10.0.0.2:8080/v1/AUTH_95216...	AUTH_95216...	admin	passadmin	t	
anotheraccount	https://10.0.0.2:8080/v1/AUTH_95216...	AUTH_95216...	admin%27	passadmin	t	
anotheraccount	https://10.0.0.2:8080/v1/AUTH_95216...	AUTH_95216...	adminbad	passadmin'--	t	
anotheraccount	https://10.0.0.2:8080/v1/AUTH_95216...	AUTH_95216...	adminbad2	passadmin'''	t	

Figure 4.11: An Example of Devauth Database with two Registered Accounts.

For example, in Figure 4.11 we show the content of the `account` table from `/etc/swift/auth.db` file where two different accounts are registered: `testaccount` and `anotheraccount`. These two accounts might represent two different customers. If any of the customers, say the owner of `testaccount`, wanted to retrieve authentication data for his users, he could not just obtain the whole database file from a provider. Provider will have to do a preprocessing and remove users belonging to `anotheraccount` (and other accounts, if any) before.

For devauth users, we recommend that a customer specifies authentication data migration possibility in SLA to avoid lock-in of authentication data.

Swauth. In swauth all authentication data is stored as objects in Object Storage cluster. Unlike devauth, swauth provides the possibility to retrieve authentication data automatically by administrator of an account. Administrator can request account data via a ReST call and retrieve all users for the account. In the below code admininstrator `admin` with password `passadmin` makes a call to get a listing of users for `thirdaccount`:

```
GET https://10.0.0.2:8080/auth/v2/thirdaccount HTTP/1.1
X-Auth-Admin-User: thirdaccount:admin
X-Auth-Admin-Key: passadmin
Host: 10.0.0.2:8080
```

The body of the HTTP result contains a JSON-encoded string which shows that account has two registered users: `userA` and `userB`:

```
{"services":
  {"storage": {"default": "local", "local": "https
      ://10.0.0.2:8080/v1/AUTH_ba939c8d-85e0-4fb6-a47a-89312
      fca004a"}},
  "account_id": "AUTH_ba939c8d-85e0-4fb6-a47a-89312fca004a",
  "users": [{"name": "userA"}, {"name": "userB"}]}
```

Afterwards, for each of the users, another ReST call can be made to retrieve user groups and password (as it was noted in section 4.3.4, password is stored in clear text). For example in the below code we retrieve data for `userA`:

```
GET https://10.0.0.2:8080/auth/v2/thirdaccount/userA HTTP/1.1
X-Auth-Admin-User: thirdaccount:admin
X-Auth-Admin-Key: passadmin
Host: 10.0.0.2:8080
```

As seen from the body of the HTTP response, `userA` has password `passuser`:

```
{"groups": [{"name": "thirdaccount:userA"}, {"name": "
   thirdaccount"}], "auth": "plaintext:passuser"}
```

While studying portability of authentication data in swauth, we found a security vulnerability that allowed mallicious *Admin* to obtain credentials of *Reseller Admin*. OpenStack Object Storage allowed *Admin* to retrieve user data for *all* the users that belonged to the owned account. If it happens that *Reseller Admin* is registered in the given account, malicious *Admin* can issue a HTTP GET call and obtain his password.

Actually, *Reseller Admin* does not need to belong to any given account, since he has administrator rights on *all* of the accounts; however, OpenStack requires each *Reseller Admin* to belong to at least one account. This means that a situation is possible, when *Reseller Admin* belongs to the account where another *Admin* is present, thus enabling the exploit of this vulnerability.

One might require further explanation to understand the consequences of the above vulnerability. As was noted in section 4.1.1, in swauth *Reseller Admin* can add new and remove existing accounts to/from OpenStack Object Storage. The idea of adding new account is similar to the notion of adding new customer to the system. A provider of cloud services might serve more than one customer and each of those customers will receive separate account(s) upon registration. Of course, that customer will need to pay provider for the owned accounts. Now imagine that *Admin* of one of the customers discovers the password of *Reseller Admin*. With this information he can add new accounts to the system and use them without the provider's knowledge. This means that such a customer will use provider's services for less costs (if not for free). Besides, a malicious *Admin* will be able to delete existing accounts, thus affecting other customers' businesses and provider's reputation.

We notified OpenStack developers about the found issue and suggested a fix, under which *Super Admin* and *Reseller Admin* can get authentication data for any user, but *Admin* can get information only if the user in question is *not Reseller Admin* [68] (see Appendix A). As a result of our report, a bug with importance 'Critical' was created and fixed afterwards by the OpenStack Object Storage developers [13].

4.4 Authorization and Access Control

4.4.1 Overview

In section 4.1.1 we studied authorization procedures for user management. In this section we study the same topic in regard to accounts, containers, and objects. Accounts are created per user, or per group of users. Objects in OpenStack Object Storage resemble files in traditional operating systems, and containers can be thought of folders (or directories).

First of all, we look at the actions that different types of users can perform with accounts:

- **User** has no permissions relative to account management.
- **Admin** can only get information about an account (URL, users, etc.)
- **Reseller Admin** can add new and remove existing accounts. Has *Admin* permissions to get information about an account.
- **Super Admin** has the same permissions as *Reseller Admin*.

Now let's look at the actions that users in different roles can perform with containers:

- **User** has access only to those containers for which he was explicitly given permissions by means of access control lists (ACL). We will discuss ACLs later in this section.
- **Admin** has permissions for all operations within owned account (add/remove containers, upload/download objects, etc.)
- **Reseller Admin** has *Admin* permissions within all the accounts (see section 4.4.3 for further information).
- **Super Admin** has no permissions relative to container/object management.

In the existing authorization system it is impossible to specify permissions at the object level, which is why customers of OpenStack Object Storage are limited to using container-based permissions. To clarify this restriction we provide an example when user wants to provide public access to object `o1` in container `c1`. Since it is impossible to specify permissions per object, user has two choices: 1) change access to container `c1` and make all the objects within this container public, or 2) create new container `c2`, move object `o1` to container `c2`, and make public access to container `c2`.

As it was noted above, users need to be given access to containers explicitly by administrators. Access is given depending on access control lists (ACL). It is possible to specify separate ACLs for `read` and `write` operations. 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.

4.4.2 Compliance with Industry Standards

Security documents studied in Chapter 3 recommend using eXtensible Access Control Markup Language (XACML) to control access to cloud resources (see [17]). OpenStack Object Storage currently relies on a proprietary solution which does not follow any recognized industry standards.

While searching for any existing Python packages that can be reused in adding XACML functionality to OpenStack, we have found only one available - ndg-xacml [54]. However, the aforementioned package was created for a specific industry project, and we have not found any information in regard to reusing it in any other project. Further investigation is necessary to find out whether ndg-xacml can be reused in OpenStack project.

4.4.3 Inadequate Permissions of Reseller Admins

In section 4.4.1 we have stated that *Reseller Admin*s have *Admin* permissions on all accounts. Unfortunately, the documentation of OpenStack Object Storage does not mention the broad permission of *Reseller Admin*s: only *Admin*s and *User*s are mentioned in [40]. As we have discovered, *Reseller Admin*s can do anything within *any* account as long as they possess the URL to that account.

We conducted the following experiment to verify permissions of *Reseller Admin*s. We added a user in a role of *Reseller Admin* with name `radmina` to `accounta`. Then we created container `filesb` on `accountb` and uploaded some files to it. Afterwards, the following attack was performed. First, we used authentication service to obtain token and account URL for *Reseller Admin*:

```
=== HTTP Request ===
GET /v1.0 HTTP/1.1
X−Storage−User: accounta:radmina
X−Storage−Pass: passradmina
Host: 10.0.0.2:11000

=== HTTP Response ===
HTTP/1.1 204 No Content
X−Storage−Url: http://10.0.0.2:8080/v1/
   AUTH_e6595be640324be4abf5c4faa6cdc524
X−Storage−Token: AUTH_tk00393a73c8664131ac1a4d5144e5b1ec
X−Auth−Token: AUTH_tk00393a73c8664131ac1a4d5144e5b1ec
```

As seen from the HTTP response, it was presumed by the authentication service that the `radmina` should use the URL from `X-Storage-Url` header to get access to account.

However, in the next step we used the received authentication token (`X-Auth-Token`), but *changed* the storage URL to the one pointing to `accountb`. This allowed *Reseller Admin* to successfully receive statistics for *accountb* and the name of containers within this account (`filesb`):

```
=== HTTP Request ===
GET /v1/AUTH_d085e938131a41d4b2386e390dff7f64 HTTP/1.1
X-Auth-Token: AUTH_tk00393a73c8664131ac1a4d5144e5b1ec
Host: 10.0.0.2:8080

=== HTTP Response ===
HTTP/1.1 200 OK
X-Account-Object-Count: 2
X-Account-Bytes-Used: 28
X-Account-Container-Count: 1
Content-Length: 7
Content-Type: text/plain; charset=utf8

filesb
```

Then, using the container name, we were able to retrieve the names of the files from this container. Knowing names of the files we could issue HTTP requests to download these files or even delete them, which confirmed that any user in *Reseller Admin* role possesses permissions to perform any action with the objects within any account, once the URL to this account is known.

One might wonder whether it is possible for *Reseller Admin* to get URLs to other accounts. We claim that it *is* possible. In section 4.3.6 where we discussed portability of authentication data, we showed HTTP requests that can be used in swauth to get information about an account, including its URL. If *Reseller Admin* will issue HTTP request but will not specify account name, then the service will respond with all registered account names. Another request for any of the accounts will reveal its URL. Besides, *Reseller Admin* can use a special tool `swauth-list` which is distributed with OpenStack Object Storage to get this information.

We presume that the permissions *Reseller Admin*s possess are inadequate to their role. Such an approach severely violates privacy of the users who store their files in OpenStack Object Storage by allowing any *Reseller Admin* to read the content of user's files. Of course, one may want to encrypt files with sensitive information *before* uploading them to OpenStack Object Storage, but still the decision to grant *Reseller Admin*s current permissions seems unjustified.

We have notified the developers about the found issue (see Appendix B). The developer's response claimed that *Reseller Admin*s might need such privileges to fix user account, for example by performing data migration for a user [61]. Another point indicated that OpenStack Object Storage can be used with other authorization systems than `devauth` or `swauth`, and these systems might provide other permissions for *Reseller Admin*s.

However, in order to use another authorization system, CSP must invest efforts to plug selected system into OpenStack.

It is up to the provider to make a decision whether changes should be made to reduce the permissions of *Reseller Admin*s. Most probably, the providers will not make changes to existing authorization systems. And most probably, users will stay unaware that their data can be freely accessed by selected representatives from the provider's side (as we have noted before, currently OpenStack documentation does not mention broad permissions of *Reseller Admin*s).

One might look at the above problem as a flavor of *security vs. usability* issue, but instead of security we have a privacy problem. Such a situation might stand for the critique of cloud computing by Richard Stallman, founder of the Free Software Foundation and creator of the computer operating system GNU, who urges users to keep track of their data themselves and not allowing "any Tom, Dick and Harry hold your data" [1].

4.4.4 Protection against Elevation of Privileges

When we discussed user management, we conducted a study to figure out whether it is possible to illegally elevate one's privileges, for example by circumventing role checks (see section 4.1.3). In this section we make the analogous study for account, container, and object management. The approach for account management is rather similar to that of user management, and the same guards against illegal elevation of privileges are in place as described in section 4.1.3. Since access control to objects is done based on access rights to a container where an object is stored (see section 4.4.1), the discussion in this section is devoted to container access.

As it was noted above, 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. Database format is SQLite, and ACL is stored in `metadata` field of `container_stat` table. Here is an example of ACL where a *write* permission is given to `userA` that belongs to `accountA`, and *read* permission is given to *all* the users from `accountA`:

```
{"X–Container–Write": ["accountA:userA", "1303668717.20931"], "X
   –Container–Read": ["accountA", "1303667956.51941"]}
```

One might easily see that ACL is JSON-encoded. We checked the source code that manipulates ACLs (`common/db.py`) and found no possibility for injections: safe methods `loads` and `dumps` from `json` module written in Python were used by the developers.

Even though ACL manipulation seems to have no problems, users of OpenStack Object Storage should be aware that usage of HTTP Referer field to make authorization decisions is insecure. The problem is that it is relatively easy to forge the value of this field. For example, after setting read access to referrer `*.example.com`, we could not get container listing via an ordinary request:

```
=== HTTP Request ===
GET /v1/AUTH_d085e938131a41d4b2386e390dff7f64/filesb HTTP/1.1
X—Auth—Token: AUTH_tk5e2b9aa7dca54ad9a1124152f5016d92
Host: 10.0.0.2:8080

=== HTTP Response ===
HTTP/1.1 403 Forbidden
Content—Length: 55
Content—Type: text/plain; charset=UTF—8

403 Forbidden. Access was denied to this resource.
```

However, by using Fiddler web proxy [20], we added `Referer` field to the request and got the response from the server:

```
=== HTTP Request ===
GET /v1/AUTH_d085e938131a41d4b2386e390dff7f64/filesb HTTP/1.1
X—Auth—Token: AUTH_tk5e2b9aa7dca54ad9a1124152f5016d92
Referer: http://mail.example.com
Host: 10.0.0.2:8080

=== HTTP Response ===
HTTP/1.1 200 OK
X—Container—Object—Count: 1
X—Container—Write: accountb:userb
X—Container—Read: .r:.example.com
X—Container—Bytes—Used: 28
Content—Length: 6
Content—Type: text/plain; charset=utf8

file1
```

4.4.5 Portability of Authorization Data

In section 4.3.6 we studied portability of authentication data. In this section we look at portability of authorization data.

Authorization decisions for account management are made based on user's groups. Groups to which user belongs are stored in the same persistence medium as authentication data - SQLite database in devauth, and text file with JSON-encoded data in swauth. That is why, portability of authorization data for account management is achieved in the same way as that of authentication data (see section 4.3.6 for additional information).

Authorization decisions for container management are made based on user's groups (for users with administrative privileges), or access control lists (for non-admin users). An administrator of an account can retrieve ACLs via an HTTP call. Read and write permissions to the containers will be available in `X-Container-Read` and `X-Container-Write` HTTP headers respectively.

4.5 Recommendations for Identity and Access Management

Based on our study of Identity and Access Management in OpenStack Object Storage, we have collected the following recommendations for the users of OpenStack:

1. Change access right to the `etc/swift/auth.db` file which stores user passwords in devauth. Currently all users have read access to the file. Only the `swift` user needs read access to this file to avoid other users registered in the system from obtaining user passwords.
2. Change default *Super User*'s password upon installation of OpenStack (in devauth it is `/etc/swift/auth-server.conf` file, in swauth - `/etc/swift/proxy-server.conf` file). Make sure that access rights to the file with *Super User*'s password are set to disallow read/write access anyone but the owner.
3. For devauth users, we recommend that a customer specifies authentication data migration possibility in Service Level Agreement (SLA) to avoid lock-in of authentication data.
4. Users should be aware that usage of HTTP Referer field to make authorization decisions is insecure, because it is relatively easy to forge the value of this field.
5. Users have to be aware that existing authorization systems provide *Reseller Admin*s with permissions to view/modify objects in any of the accounts. This means that any file that is stored in the cluster can be read by any user acting in the role of *Reseller Admin*.

For the developers of OpenStack we have collected the following recommendations:

1. Implement user provisioning functionality using Service Provisioning Markup Language (SPML) to avoid customers being locked into proprietary solution currently utilized by OpenStack.
2. Implement authentication functionality using Security Assertion Markup Language (SAML) to allow identity federation and avoid locking into proprietary solutions. We indicate that PySAML2 Python library can be used for this purpose by the developers.
3. Implement password strength checks before registering users to disallow weak passwords. We recommend using python-crack library for this purpose because of its ease of use and reach functionality.

4. Implement passwords hashing before storing it in both devauth and swauth authentication systems. Before hashing, passwords have to be concatenated to a salt and/or username. Python module hashlib can be used to hash password with SHA-256 algorithm before storing it. If salt value is used (not a username), it can be stored alongside hashed password.
5. Implement authorization functionality using eXtensible Access Control Markup Language (XACML) to avoid locking into proprietary solution. The ndg-xacml Python library that claims to implement XACML version 2.0 might be reused in OpenStack, however, additional analysis is to be done whether its feasible to use it in the project.

Summary

In this chapter we studied identity and access management in OpenStack Object Storage. For our analysis we took as input the issues that were discussed in security documents which we studied in Chapter 3. From that study we made a conclusion that four sub-topics should be paid attention to when discussion identity and access management: identity provisioning/deprovisioning, identity federation, authentication, authorization and access control. Since analyzed version of OpenStack Object Storage contained two authentication and authorization systems, devauth and swauth, we made an analysis for both of them.

We started with user management (section 4.1). We had to consult source code in order to determine different user roles and their permissions with regard to identity provisioning and deprovisioning, since the documentation lacked necessary information. Then we checked whether existing solution supports Service Provisioning Markup Language (SPML), which was recommended by studied cloud computing documents from Chapter 3. Neither devauth, nor swauth supported SPML. We also analyzed source code to find whether injection attacks can be used to elevate one's privileges during user management and found no weaknesses.

Identity federation was next sub-topic in this Chapter (section 4.2). Cloud computing documents that were studied in Chapter 3 recommended using Security Assertions Markup Language (SAML) or WS-Federation as enabling technologies. Unfortunately, none of the authentication systems provided by OpenStack Object Storage supported any of these technologies.

Authentication discussion followed in our study of identity and access management in OpenStack (section 4.3). We started with an overview and then checked whether provided authentication systems comply with any recognized industry standards. As with user management, no compliance was found. Afterwards, we analyzed whether requirements on password strength were imposed during user registration. No such requirements existed, which made us to investigate which existing Python libraries could be reused to add such functionality to OpenStack. Password storage was next problem we looked into. Our analysis showed that devauth does not utilize appropriate measures to restrict access control to the file with passwords, and passwords are stored in clear text. Even though swauth had access control procedures in place, passwords were still stored in clear form, thus allowing insider with root access to the system getting hold of passwords of all the users in a system. Our recommendation to deal with password storage was based on "Electronic Authentication Guideline" provided by NIST. A thorough evaluation of authentication tokens was performed and showed that the approach taken to generate them used a reliable source of randomness which disallowed attackers to generate new tokens by observing a large number of previously created ones. While analyzing portability of authentication data, we encountered a security vulnerability which could lead to obtaining credentials of Reseller Admin. We submitted our finding to OpenStack developers, and the bug was fixed soon afterwards.

Authorization and access control was next topic that we investigated (section 4.4). Again, we analyzed source code to determine user roles and their respective permissions in regard to account, container, and object management, since the documentation lacked enough details. Our analysis showed that Reseller Admins possess permissions to view/delete any object from any account, which we claim is a serious privacy concern. We found no weaknesses when analyzed source code to find whether injection attacks can be used to elevate one's privileges during account, container and object management. We also showed that access control to objects based on HTTP Referer header can be abused by an attacker.

Finally, we provided a section with collected recommendations that we made during the analysis of identity and access management in OpenStack Object Storage. We provide recommendations both for the users of OpenStack Object Storage, and for the developers who are working on a project.

Chapter 5

Data Management

In this chapter we analyze data management in OpenStack Object Storage. When we studied security issues relevant to cloud computing in Chapter 3, we selected Technical issues to consider for data management in the cloud (see Table 3.4.2 on page 23). Besides, in the list of Policy, organizational, and legal issues (see Table 3.4.1 on page 21), we encountered the legal issue of data location compliance, which was also relevant to data management in OpenStack.

We start this chapter with data location analysis (section 5.1), where we include a discussion on data location compliance. Then we continue with sections on isolation (5.2), backup and recovery (5.3), and deletion (5.4). The study of encryption and key management (5.5), as well as integrity verification (5.6), follows.

While analyzing each of the above issues, we made recommendations for developers and users of OpenStack Object Storage. The summary of these recommendations is given in section 5.7.

5.1 Data Location

5.1.1 Overview

The process of data retrieval is shown in Figure 5.1 on page 50. Users of OpenStack Object Storage access their data via HTTP calls to a Proxy server by providing a logical path to an entity (by entity here and below we mean any of *account*, *container*, or *object*). A logical path starts with storage URL, which the user receives from an authentication server, and may continue with names of container and object separated by slashes ('/'). Storage URL ends with account name for which the user is registered. In the below code we show logical path to user's object `myObject`, which is stored in container `myContainer`:

```
https://<PROXY-IP>:8080/v1/AUTH_e6595be640324be4abf5c4faa6cdc524
   /myContainer/myObject
```

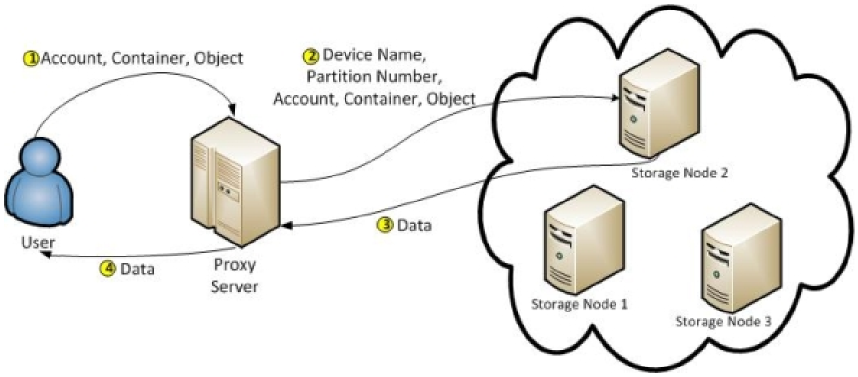


Figure 5.1: Data Retrieval in OpenStack Object Storage

Proxy server is responsible for translating the logical path, specified by a user, to a physical location where data actually resides in the cluster. In order to determine physical data location in the cluster, Proxy server uses the concept of *ring*s [41]. The entire cluster is separated into *partition*s, and each partition is then mapped to a *device* (OpenStack partitions should not be confused with hard disk logical storage units, which are called partitions as well). A device represents a hard disk on one of the nodes in the cluster. Each partition is replicated across the cluster. And ring structure determines on which nodes the partition has to be replicated. Thus, a ring structure is basically a file which maps partitions to nodes (because of the huge number of partitions, the content of the ring file is gzipped).

Once proxy server has found a physical location of an entity, it contacts a dedicated server process on a Storage node. There are separate processes (services) for managing accounts, containers, and objects. On the way from Proxy server to corresponding entity service, the following data is sent: device name, partition number, account, container, object. In the below code we show physical path to user's object `myObject`, which is stored in container `myContainer`:

```
https://<STORAGE-NODE-IP>:6000/sdb1/99721/
   AUTH_e6595be640324be4abf5c4faa6cdc524/myContainer/myObject
```

Server process finds a location for an entity using hash computation, as described in the next section.

5.1.2 Determining Data Location for Accounts, Containers and Objects

In this section we describe the algorithm for computing data location for OpenStack Object Storage entities. The below information was gathered using both the ring documentation [41] and source code analysis. For each entity, whether its account, container, or object, a separate directory structure is created using the same approach:

1. A path to an entity is created using concatenation of account, container, and object (container and/or object may be missing), where slash ('/') is used as separator.
2. Created path is concatenated with `hash_path_suffix`, which is a secret value that is set in configuration file upon installation of OpenStack.
3. MD5 hash of the above value is computed.
4. Based on the hash, partition number for an entity is calculated using bit shift arithmetic.
5. Based on the partition number, a list of nodes (hosts in storage cluster) is determined from the ring file. Each of the nodes will have the same replicated copy of an entity.
6. The location is set as `file://node/entity/partition/last_three_symbols_of_hash/hash`, where entity is one of `accounts`, `containers` or `objects` depending on the entity to be stored.

Based on the above description, we can find an answer to a question what information, other than names of account, container and object, does one need to calculate the location of the file in a storage cluster. It is the value of `hash_path_suffix` and the ring files. `Hash_path_suffix` is stored in `/etc/swift/swift.conf` file. Ring files are stored in `/etc/swift/account.ring.gz` (container and object rings are also stored in dedicated files in the same location).

The protection of `/etc/swift/swift.conf` has to be given enough attention by a provider. Imagine that a malicious insider changes the value of `hash_path_suffix` on proxy node. Since proxy determines the physical location of data upon user requests by making hash calculations using `hash_path_suffix`, previously uploaded files will no longer be accessible (files will still be stored the servers, but there will be no way to calculate the correct path to those files). Thus, affecting `hash_path_suffix` value on the proxy node will cause service outage and have catastrophic consequences for a provider.

5.1.3 Achieving Data Location Compliance

As was emphasized by the security documents studied in Chapter 3, some of the data stored in the cloud has legal requirements to be stored physically within certain geographical boundary (usually within a country). Because of the distributed nature of cloud computing, such a requirement is sometimes hard to guarantee.

When analyzing OpenStack Object Storage, we encountered the concept of *zone*s. Original idea of introducing zones was to achieve greater efficiency from replication. According to the documentation, "zones can be used to group devices based on physical locations, power separations, network separations, or any other attribute that would lessen multiple replicas being unavailable at the same time" [41]. The ring guarantees that no two replicas will be stored in the same zone.

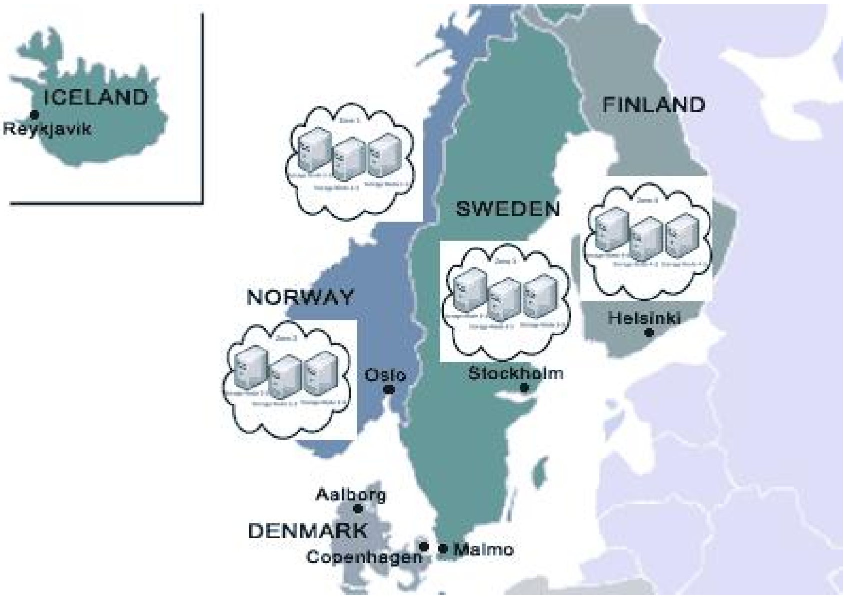


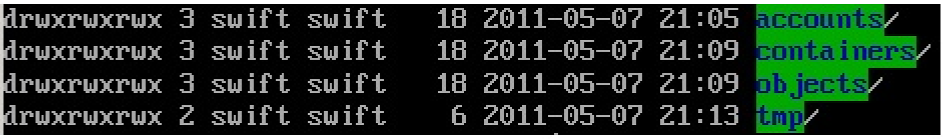
Figure 5.2: Using Zones to Achieve Data Location Compliance

Despite the different original purpose of zones, we state that this concept may be very handy to achieve data location compliance. For example, in Figure 5.2, we show a provider that operates in Nordic countries in Europe and has servers in Norway (zones 1-2), Sweden (zone 3), and Finland (zones 4). If a customer has legal requirements to store its data on the servers in Norway, he might request to restrict data storage to zones 1-2.

We made an inquiry whether it is possible to impose restrictions on data storage location in OpenStack Object Storage [64]. Based on the response from the developers, OpenStack currently does not provide such a functionality. Nevertheless, discussions about modifying ring structure to achieve this functionality exist [45]. We suggested to use zones for the purpose of assuring data location compliance [67]. In our initial suggestion we stated that ring modifications will be necessary to impose restrictions on used zones for specific accounts. However, since we felt that most developers will not feel comfortable with modifying existing ring structure, because it is quite complicated, we submitted another suggestion that does not require code changes for making ring lookups [63] (for the general information about ring structure, please refer to section 5.1.1).

We assume that it is possible to implement data location compliance on top of ring in the way similar to the one, that was used during replication. If an OpenStack replicator detects that a remote drive can not be reached, it will query ring for additional nodes using `get_more_nodes` method of the `Ring` class [46]. Calling `get_more_nodes` subsequently will eventually return all the possible zones that exist in OpenStack installation. So, a wrapper over `Ring` class can maintain the list of banned zones for each of the accounts and use `get_more_nodes` method to find allowed zones [63].

Both of our recommendations were submitted to OpenStack and caused subsequent discussions in mailing list. An extract from the discussion in mailing list is given in Appendix D.



```
drwxrwxrwx 3 swift swift   18 2011-05-07 21:05 accounts/
drwxrwxrwx 3 swift swift   18 2011-05-07 21:09 containers/
drwxrwxrwx 3 swift swift   18 2011-05-07 21:09 objects/
drwxrwxrwx 2 swift swift    6 2011-05-07 21:13 tmp/
```

Figure 5.3: Directory Structure on a Storage Node in OpenStack Object Storage

At the time of writing, it is unsure which way the developers will choose to implement data location compliance; however, in the meantime, cloud service providers, which need the functionality to impose restrictions on data location, can implement it using our suggestions from [63].

5.2 Isolation

5.2.1 Overview

One of the characterizing features of cloud computing is the resource pooling, when provider's resources are shared among different customers. In terms of OpenStack Object Storage, we are concerned about storage sharing. For private cloud deployment (see section 2.1.2 on page 7 for background information), the issue of data isolation is irrelevant, since only a single organization is utilizing cloud storage. However, for community and public deployment, this problem is of high importance.

The issue of separation of data that belong to different customers has been given enough attention in all the security documents studied in Chapter 3. Analyzed documents use words *isolation*, or *compartmentalization* to refer to this issue. No matter what name is used, the basic idea is to separate (logically and/or physically) data that belongs to different customers.

One of the examples that justifies the need for data isolation deals with subpoena processing. Suppose that one service provider serves two customers. If data of one customer is to be disclosed as a result of subpoena issue, data of the other customer has to remain concealed. If data is intermingled on the storage medium, it might be impossible to satisfy the above requirement.

In Figure 5.3 we show the directory structure for OpenStack Object Storage entities that exist within a storage device. As seen from the figure, accounts, containers, and objects have different directory on the storage device. Information about accounts and containers is stored in SQLite database files, while objects are stored as files with extension `.data`. Temporary directory is used for storing file chunks during upload (please refer to section 5.4.1 for more information).

Within each of the directories for OpenStack entities, we have partitions. In OpenStack, partition is a directory which name equals to a number between 1 and $2^{<max-partition-number>}$.

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.

5.2.2 Isolation in OpenStack

When analyzing OpenStack Object Storage, we got an impression that isolation depends only on the hash value calculated for the path to the object. In order to find out whether other isolation measures were applied, we conducted the following experiment:

1. Created a dummy implementation of hash function that returned the same hash value whenever called.
2. Changed OpenStack code to use dummy implementation of hash function when calculating path for files.
3. Uploaded fileA to containerA on accountA using credentials of userA.
4. Uploaded fileB to containerB on accountB using credentials of userB.
5. Downloaded files from containerA on accountA using credentials of userA.

First of all, we created a stub for md5 class in Python that always returned the same hash (see Appendix H for source code). With such an implementation, we were able to test what would happen if two different inputs to the hash function resulted in the same output (we postpone the discussion whether this situation can ever happen until later in this section).

Then, we changed the swift/common/utils.py to include our dummy implementation of md5 hash function, instead of the used one from hashlib Python module. The only function that used md5 routine in source code for OpenStack Object Storage was hash_path function, which was exactly the one that calculated hash of the path before storing objects in OpenStack.

Afterwards, we used st module, a Python module that is built on top of the OpenStack ReST API, to upload and download files to OpenStack Object Storage. The format of the st command is the following:

```
python st –A <URL–to–Authentication–Server> –U <account:user> –K
    <password> <action> <container> <object>
```

The following commands were executed using st module:

```
python st –A http://10.0.0.2:11000/v1.0 –U accountA:adminA –K
   passadmina upload containerA fileA
python st –A http://10.0.0.2:11000/v1.0 –U accountB:adminB –K
   passadminb upload containerB fileB
python st –A http://10.0.0.2:11000/v1.0 –U accountA:adminA –K
   passadmina download containerA
```

As a result of the first command from the above listing (line 1), we uploaded fileA to containerA on accountA using credentials of adminA (a registered user on accountA). The second command (line 2) did the same for the user registered on accountB. When we executed the third command (line 3), we got confirmation for our assumption that OpenStack utilized isolation based only on the results from hash function. As a result of the third command, we successfully downloaded the contents of fileB using credentials of userA.

Below, we provide detailed description of the actions that happened at back-end during the execution of the first command (line 1 from the above code):

1. When Proxy server was requested to store fileA, it first checked credentials for user adminA. Since adminA was a registered user and had right to upload objects to accountA, the permission was given.

2. Afterwards, the hash for /accountA/containerA/fileA path, concatenated with hash_path_suffix, was calculated (hash_path_suffix value was retrieved from the configuration file). Dummy implementation of md5 function returned a hard-coded hash value abcdef.

3. Using bit arithmetic, the partition number was obtained (in our experiment it was equal to 99721), based on which the ring returned Storage nodes where the file had to be stored.

4. As was described in section 5.1.2, the complete path to uploaded file was determined to be objects/99721/def/abcdef.

5. FileA was uploaded to Storage nodes and received a name equal to a timestamp when Proxy server was contacted to store the file (see section 5.4.1 for more details on naming uploaded objects).

Upon execution of the second command, the following actions happened at back-end:

1. Credentials of adminB were confirmed by the Proxy server.

2. Because of the dummy hash function, the same hash value abcdef was returned for the input path /accountB/containerB/fileB.

3. Since the ring used only hash value to determine Storage nodes for the file, the same set of nodes were selected to store fileB, as in the case of fileA.

4. The complete path for storing fileB was determined to be objects/99721/-def/abcdef. Note that this is the same path that was used for storing fileA, even though these two files logically belong to two different containers on two different accounts.

5. FileB was uploaded to the Storage nodes and received a name equal to a timestamp when Proxy server was contacted to store the file. Afterwards, fileA, present in the same directory, was deleted, because its timestamp (i.e. filename) was older, and

the Storage node presumed that `fileA` is an older version of `fileB` and must be removed.

Now it must be clear why upon execution of the third command, `userA` was able to download the contents of `fileB`. Proxy node allowed file download after checking credentials of `userA`, but a Storage node sent back file belonging to `userB`.

5.2.3 Attacks on OpenStack Isolation

Earlier in our experiment we have shown that OpenStack relies only on path hashing to isolate files belonging to different users (see section 5.2.2). In this section we try to analyze whether OpenStack approach to isolation can be abused.

One case to consider is when among all the documents stored at OpenStack Object Storage, two documents are present that have the same hash value. One can use approximations from Birthday paradox to calculate the minimum required number of documents, so that the probability that two of them have the same hash is 0.1%. It turns out that for MD5 hash function this number of documents is $8.3 * 10^{17}$ (for theoretical details how to calculate the value, please refer to [22]).

Now we have to analyze whether such a number of documents can be stored in OpenStack Object Storage to cause a 0.1% probability of a problem for two users. In our analysis we suppose that we use OpenStack as a back-end for storing e-mail messages, which is a real-world use case that has a requirements to store huge number of documents. One of the largest e-mail providers - Yahoo Mail - currently has 284 million users [15]. According to technology market research firm Radicati, the typical corporate user sends and receives about 110 messages daily [58]. If each Yahoo Mail user sends the same number of emails as average corporate user, in one year the number of sent/received messages to be stored on OpenStack will be around 10^{13}. Since the calculated number is orders of magnitude smaller than the number of documents that is required to have 0.1% probability of hash collision, we conclude that currently there is no practical use case of OpenStack Object Storage deployment for which the chosen approach will be insufficient.

The other cases to consider assume existence of an attacker, who tries to exploit OpenStack isolation. Two attacks are possible in this situation. In the *preimage attack*, attacker needs to know path to a file uploaded by some user and then to find names for a container and object that will hash to the same value as the user's file. Currently, there is no publicly available algorithm for finding an input to match the desired output of a hash function. We can not be sure that such an algorithm is unknown to any of the intelligence agencies. Likewise, we can not be sure that such an algorithm will not be found in the nearest future. However, as of now, there are no reasons to state that approach taken by OpenStack is flawed.

Another attack, which an attacker might try to perform, is the *collision attack*, when attacker tries to generate file names that will result in the same hash value by changing "container/object" suffix (it is possible to perform this attack on MD5, see, for example, [73]). Afterwards, an attacker can upload two different files, knowing that the second uploaded file will overwrite the first. One may think that in such a case an attacker can only harm itself, since it is his own files, which will be overwritten. However, once there exists a legal agreement between a provider and a customer that provider will prevent data loss of customer's files, the provider can be sued for data loss by that malicious user, even though such a data loss occurred due to the carefully planned user activities (the cases when cloud service providers were sued by companies for data loss have happened before, see, for example, [23]).

OpenStack prevents the possibility of *collision attack* by using hash_path_suffix, which is added to the file path before passing input to a hash function. In [73] there is a description of an attack, which for fixed P_1 and P_2 finds values S_1 and S_2, so that:

$$md5(P_1||S_1) == md5(P_2||S_2)$$

In our case, P_1 and P_2 are equal and set to account name (or "account/container" pair) for an attacker. S_1 and S_2 will than indicate names for "container/object" pair (or "object"). However, in presence of hash_path_suffix, an attacker needs to generate M_1 and M_2 for fixed P and S, so that:

$$md5(P||M_1||S) == md5(P||M_2||S)$$

The above attack is also possible on MD5, once hash_path_suffix is known. For example, in [74] there are examples of different PDF files which predict different outcomes of US elections. All of the provided files have the same Md5 hash. If one makes byte-to-byte comparison of two such files, he will note that the suffix (i.e. end of the document) is identical. Actually, it is required to be identical to produce syntactically correct PDF documents; however, it is the inner part of the documents that is different.

Based on the above, we may emphasize that hash_path_suffix value should be kept secret. Besides, it might be necessary to consider other hashing functions than MD5. As stated in OpenStack Object Storage Administrator Guide, "MD5 was chosen for its general availability, good distribution, and adequate speed." In the light of our analysis of OpenStack isolation, more evidence is gathered towards the decision to select other hash function than MD5.

As a result of our communication with the OpenStack team about the isolation issue (see [69]), a documentation bug was created that aimed to explicitly caution to keep the hash_path_suffix secret (see [8]).

5.3 Backup and Recovery

Cloud Security Alliance indicates that data backup is a mechanism that allows to prevent "data loss, unwanted data overwrite, and destruction", and warns users against assuming that data stored in a cloud is backed up and recoverable [3].

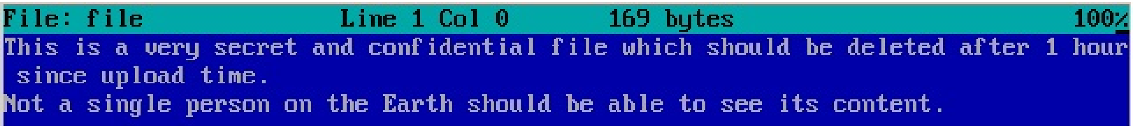
In order to prevent data loss and destruction, as well as increase data availability, OpenStack Object Storage stores data in several location across the cluster. A dedicated server process called *replicator* is running on the Storage node to propagate data copies to different nodes. Separate processes exist for replicating accounts, containers, and objects.

One problem with OpenStack is that data backup/recovery is *not* supported. As we describe in section 5.4.1, upon upload of a new object with the same name all the previous versions of a file are deleted. Even though replication indeed prevents data loss, but in case of unwanted data overwrite, it is impossible to restore file to a previous version.

In our opinion, backup/recovery can be easily added to OpenStack without many changes in the source code. The solution actually lays in the approach that OpenStack uses for data storage. As we note in section 5.4.1, after successful file upload via PUT method, stored file receives name equal to timestamp, and all the previous versions of the file are deleted. If one wants to allow keeping up to N backup versions (can be configurable value), the method which removes previous versions of files 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).

Our suggestion was submitted to OpenStack team via mailing list [66]. Developers replied that distribution of data might have been affected if such an approach was used, since OpenStack Object Storage would store all the versions of the same file in the same storage nodes (ring is unaware about file versions and will select nodes based on "account/container/object" path only). An example which was used by the developers showed a use case when one of the users saved virtual disk image with a size of 5 Gb in OpenStack. If other users stored files of less size, the distribution of occupied space across storage nodes would deviate, thus resulting in a situation when some files could not be saved because of lack of space on some nodes, while there was plenty of free space available on the other nodes. We indicated that if the maximum number of backups was a configurable value, then it would be up to an administrator whether to enable backups or not [62]. For example, in a setup where OpenStack was used to store user pictures, the problem of file distribution would be irrelevant.

Both our recommendations were submitted to OpenStack and caused subsequent discussions in mailing list. As a result, another option was suggested for data backup which would work in the use case for storing files of different size, but required more changes to the source code. An extract from the discussion in mailing list is given in Appendix E. At the time of writing, it is unclear which way the developers will choose to bring backup/recovery functionality to OpenStack Object Storage. In the meantime, cloud service providers which need the backup functionality now can implement it using our suggestions from [66].



```
File: file                   Line 1 Col 0        169 bytes                  100%
This is a very secret and confidential file which should be deleted after 1 hour
 since upload time.
Not a single person on the Earth should be able to see its content.
```

Figure 5.4: Confidential File Uploaded to OpenStack Object Storage

5.4 Deletion

5.4.1 Overview

In the context of cloud computing, issues related to data deletion deal with carefully removing all the copies of the data that existed in a cluster. It might happen that data deleted in all but one storage nodes becomes restored afterwards due to recovery procedures employed. Another issue of data deletion deals with proper storage recycling. Very often files (or some parts of it) that were deleted can be recovered from the hard disk afterwards.

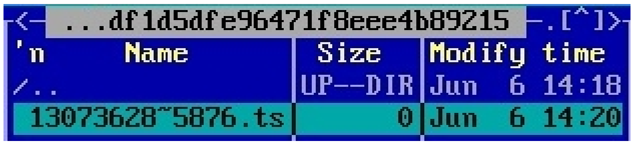
In order to analyze data deletion approach we have to find out how files are uploaded for storage first. Upon studying source code of OpenStack Object Storage, we discovered that at first file is written to a temporary location on a storage device, which by default is set to `/srv/node/sdb1/tmp`. The temporary directory is common for all the customers on the system. Afterwards, when all the chunks of a file are uploaded, this file is moved to a new location using Python `os.rename` function. New location for the file is determined by the same hashing approach that was described in section 5.1.2. The name of the new file will be equal to the timestamp specified in HTTP header `X-Timestamp`.

Using timestamp for a file name allows checking for previous versions of the file. As soon as the file is moved from a temporary to a new location, OpenStack runs an algorithm which compares all the filenames that exist in a new location to the filename of the newly uploaded file. Since filename equals to the creation timestamp, finding a file with an older timestamp means that this file represents an older version and should be deleted.

When user decides to delete his file, OpenStack creates a new zero-size file with extension `*.ts` (*tombstone*) and new timestamp as a name. Afterwards, the same algorithm to delete files with older timestamps is run. By using a tombstone file, OpenStack deals with the problem of removing file from all the nodes: replication process will later propagate the tombstone file to other nodes, at the same time deleting the actual content of the file.

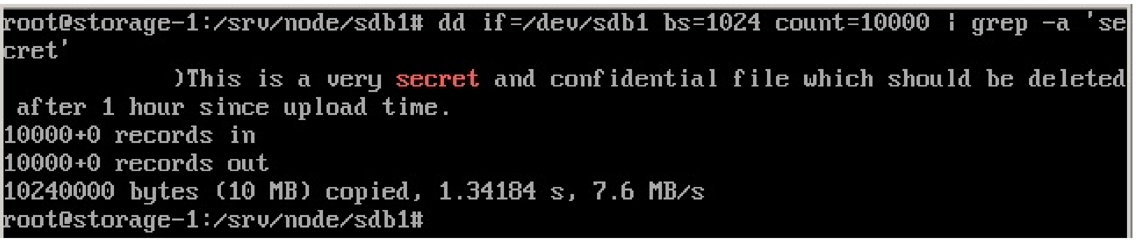
5.4.2 Data Remanence

In order to find out whether all file data was cleared from the storage medium upon deletion, we made an experiment trying to use file recovery to find out whether it was possible to retrieve information stored in a file prior to deletion.



```
<- ...df1d5dfe96471f8eee4b89215 -.[^]>
'n      Name        |  Size   |Modify time
/..                 |UP--DIR|Jun  6 14:18
 13073628~5876.ts  |      0|Jun  6 14:20
```

Figure 5.5: Tombstone of a Confidential File in OpenStack Object Storage



```
root@storage-1:/srv/node/sdb1# dd if=/dev/sdb1 bs=1024 count=10000 | grep -a 'se
cret'
            )This is a very secret and confidential file which should be deleted
 after 1 hour since upload time.
10000+0 records in
10000+0 records out
10240000 bytes (10 MB) copied, 1.34184 s, 7.6 MB/s
root@storage-1:/srv/node/sdb1#
```

Figure 5.6: Part of the Confidential File Retrieved after Deletion

In Figure 5.4 on page 59 we show a file that was uploaded to OpenStack and then deleted afterwards. Upon execution of a deletion procedure, a tombstone file was created, as seen in Figure 5.5. The tombstone file was empty and contained extension `*.ts`.

Afterwards, we used a combination of `dd` and `grep` utilities on Storage node to search within blocks of binary data existed on a hard drive (the credit for the idea to use such an approach goes to [79]). As seen in Figure 5.6, we were able to recover a part of the file, even though it was deleted from the hard disk previously.

We recommend that users include a specific requirement into their SLAs with a provider that obliges the latter to use appropriate sanitization procedures that disallow restoration of deleted files, for example, in a way similar to the one described in this section.

5.5 Encryption and Key Management

OpenStack Object Storage does *not* encrypt files before storing in a cluster [32]. This means that if a user needs to store sensitive information in OpenStack, he will have to encrypt files before sending them to OpenStack and take care of key management for encryption himself.

As we have noted in section 4.4.3 on page 42, users in role of *Reseller Admin* can view any file on any of the accounts, which is why if customers want to prevent provider's personnel from accessing their data, they should encrypt their files before uploading to OpenStack.

5.6 Integrity Verification

OpenStack Object Storage provides the following possibilities for integrity checks:

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

In order to perform integrity verification on the server side, user has to supply MD5 hash of the uploaded object in `ETag` HTTP header [33]. If such a header is provided, OpenStack will calculate MD5 hash on the server and compare both values. In case of differences between user-supplied and calculated value, an HTTP response code 422 is returned, which according to RFC 4918 means that "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" [16]. An example specified in RFC 4918 for this status code with syntactically correct but semantically erroneous XML file, indicate the suitability of the usage of this HTTP status code for integrity errors.

For the client-side integrity checks, user will have to use `ETag` value from the HTTP response and check the integrity of transfer himself. However, in case of integrity errors, the erroneous version of a file will already be stored and replicated across OpenStack cluster, and user will have to upload the same file again to prevent others from downloading potentially incorrect data. That is why using server-side integrity checks during upload seems to be more justified.

In order to analyze integrity monitoring of objects stored in a cluster, we have performed source code analysis of OpenStack Object Storage. On the Storage server, a daemon process which makes integrity checks is called `auditor`. There are separate auditors for accounts, containers, and objects. Audit of accounts and containers is limited to checking whether it is possible to read data from the database with the information about respective account of container. Object auditor checks file size and hash value by comparing actual size and MD5 hash value with those stored in object metadata. Since object metadata is stored as extended attributes on a filesystem, only those filesystems that support such attributes can be used with OpenStack. In case when server process discovers that actual hash value of an object differs from that stored in metadata, a file will be quarantined, and replication process will reload correct version of the file from another replica in a cluster.

5.7 Recommendations for Data Management

Based on our study of Data management in OpenStack Object Storage, we have collected the following recommendations for the users of OpenStack:

1. System administrator should make sure that `/etc/swift/swift.conf` is protected from modifications. A backup copy of this file should be stored in a safe location. If `hash_path_suffix` value that is stored in this file is modified, previously uploaded files to OpenStack Object Storage will no longer be accessible.

2. Users may benefit from including a specific requirement into their SLA with a provider that obliges the latter to use appropriate sanitization procedures before recycling storage medium. This can prevent illegal restoration of deleted files.
3. Since provider's personnel that acts in a role of *Reseller Admin* can view any file on any of the accounts, customers who want to prevent provider's staff from accessing their data should encrypt their files before uploading to OpenStack.

For the developers of OpenStack we have collected the following recommendations:

1. In order to assure data location compliance, we recommend either to make modification to ring structure, or to build a wrapper over the `Ring` class that reuses `get_more_nodes` method to allow restricting data location to specific zones.
2. To allow backup and recovery in OpenStack Object Storage, we recommend to make a number of kept backup copies configurable, and change `unlinkold` method in `DiskFile` class to delete only those previous versions of files that exceed the configured value.

Summary

In this chapter we studied Data management in OpenStack Object Storage. An input for our analysis were the security issues identified during the study of cloud security documents from Chapter 3. Issues that were considered included the following: data location, isolation, backup and recovery, deletion, encryption and key management, and integrity verification.

We started with data location (section 5.1). We studied source code and documentation to find out how OpenStack determined data location for user-provided files. Since the issue of data location compliance was relevant to this section, we checked whether it was possible to restrict users' data to specific locations, and found that such a functionality was not supported. We suggested to use OpenStack concept of zoning to allow restrictions on users' data and submitted our ideas about required modifications to the source code to OpenStack mailing list.

The issue of isolating data that belongs to different users was studied next (section 5.2). We determined that OpenStack Object Storage uses MD5 hashing algorithm to separate users' data. In order to determine whether additional guards were used, we created a dummy implementation of MD5 hash function and changed OpenStack source code to use our implementation. Our experiments showed that no other guards were in place to separate users' data, which could make cloud service provider that uses OpenStack vulnerable to legal claims from dishonest customers who exploited OpenStack isolation. Our findings on isolation were submitted to OpenStack team.

In section 5.3 we discovered that backup and recovery were not supported in OpenStack Object Storage. Again, we suggested how to bring this functionality to OpenStack and started a discussion in the mailing list about the issue. Next section (5.4) was devoted to secure data deletion, where we showed that it is possible to recover files deleted by OpenStack server process, which is why we advised customers to include into their SLAs with providers requirement to securely erase data from storage media before recycling.

Encryption (section 5.5) was not supported in OpenStack. Study of Integrity Verification (section 5.6), where we came to the conclusion that existent procedures were sufficient, concluded our study of Data management issues in OpenStack.

At the end of the chapter we presented a summary of collected recommendations both for the users of OpenStack Object Storage and developers working on a project.

Chapter 6

Conclusion

In this work we analyzed security issues in open-source cloud computing project - OpenStack Object Storage. We started with finding out which security issues should be taken care of when using cloud services. In doing this, we looked into documents that were created to facilitate adoption of cloud computing by a Cloud Security Alliance, an organization consisting of industry representatives, and two governmental institutions: European Network and Information Security Agency, and National Institute of Standards and Technology. By examining security-related documents for cloud computing, we were able to compile a list of security issues to be used when evaluating security of OpenStack cloud solution.

Next, we performed an analysis of Identity and access management in OpenStack Object Storage. The following areas were covered: identity provisioning/deprovisioning, identity federation, authentication, authorization and access control. As a result of our study, we found a number of security issues, including but not limiting to the following: security vulnerability, which allowed administrators with lower permissions to obtain credentials of administrators with higher permissions; inadequately high permissions of one type of administrators, which allowed to read/delete all the files of all the users; poor password management procedures for both authentication systems provided by OpenStack.

Afterwards, we studied Data management procedures in OpenStack Object Storage. The following areas were covered: data location, isolation, backup and recovery, deletion, encryption and key management, integrity verification. As a result of our analysis, we reported a possibility to compromise isolation of files belonging to the same user with subsequent overwrite of one file by another, which could make cloud service provider that uses OpenStack vulnerable to legal claims from dishonest customers who exploited OpenStack isolation. Besides, we submitted our suggestions for implementing data location compliance and backup/recovery procedures in OpenStack Object Storage.

Our findings show that there is a need for security evaluation of cloud computing offerings in general, and OpenStack in particular. Since OpenStack is a relatively new project which evolves rapidly (for example, our analysis was based on Bexar release, which appeared on February 2011, but in April 2011 new release of OpenStack became available), we suggest that further study of security issues in OpenStack is necessary. Besides, the other projects from OpenStack family, such as OpenStack Compute or OpenStack Image Service, can benefit from security evaluation.

Glossary

Birthday Paradox

A problem from probability theory to find a minimum number of randomly chosen people so that the probability of any pair of them having birthday on the same day is 50%. The answer is 23, which is a suprisingly low value, hence the *paradox*.

Collision Attack

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

Community cloud

Cloud deployment model under which cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns.

Hybrid cloud

Cloud deployment model under which cloud infrastructure is a composition of two or more clouds that remain unique entities but are bound together.

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.

OS Hardening

The process of addressing security issues in OS by appropriate configuration, applying latest patches, etc.

Pay as you go

A payment model usually utilized in cloud computing when customers pay only for the actually consumed resources/services.

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.
Social Engineering Attack	An attack where human interaction (e.g. phone call to a service desk with a request to tell forgotten password) is used to obtain sensitive data.
Software as a Service (SaaS)	Cloud service delivery model under which customer can use provider's applications which are running on a cloud infrastructure.
User Deprovisioning	A process of unregistering 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.
Web Server Gateway Interface	Interface between web servers and web applications for Python programming language.

Bibliography

[1] C. Arthur. Google's ChromeOS means losing control of data, warns GNU founder Richard Stallman. http://www.guardian.co.uk/technology/blog/2010/dec/14/chrome-os-richard-stallman-warning, December 2010. Retrieved May 2011.

[2] E. Brown. Cloud Computing at NIST: Two New Draft Documents and a Wiki. http://www.nist.gov/itl/csd/cloud-020111.cfm, February 2011. Retrieved February 2011.

[3] 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. Available at http://www.cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf.

[4] W. E. Burr, D. F. Dodson, and W. T. Polk. Electronic Authentication Guideline. Technical report, National Institute of Standards and Technology, April 2006. Publication 800-63. Version 1.0.2. Available at http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf.

[5] D. Catteddu and G. Hogben. Cloud Computing Security Risk Assessment. Technical report, European Network and Information Security Agency, November 2009. Available at http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment/at_download/fullReport.

[6] Cloud Security Alliance. About Cloud Security Alliance. http://cloudsecurityalliance.org/About.html. Accessed February 2011.

[7] J. Curry. OpenStack Blog: Introducing OpenStack. http://www.openstack.org/blog/2010/07/introducing-openstack/, July 2010. Retrieved February 2011.

[8] J. Dickinson. Ensure that the docs explicitly caution to keep the hash path suffix secret. https://bugs.launchpad.net/swift/+bug/791620, June 2011. Retrieved June 2011.

[9] European Network and Information Security Agency. Activities - ENISA. http://www.enisa.europa.eu/about-enisa/activities. Accessed February 2011.

[10] Gartner Inc. Gartner Executive Programs Worldwide Survey of More Than 2,000 CIOs Identifies Cloud Computing as Top Technology Priority for CIOs in 2011. http://www.gartner.com/it/page.jsp?id=1526414, January 2011. Retrieved February 2011.

[11] F. Gens. IT Cloud Services User Survey, pt.2: Top Benefits & Challenges. http://blogs.idc.com/ie/?p=210, October 2008. Retrieved February 2011.

[12] F. Gens. New IDC IT Cloud Services Survey: Top Benefits and Challenges. http://blogs.idc.com/ie/?p=730, December 2009. Retrieved February 2011.

[13] gholt. Fix .admin get user privileges. https://bugs.launchpad.net/swift/+bug/747618, April 2011. Retrieved April 2011.

[14] B. Golden. Cloud Computing: 2011 Predictions. http://www.cio.com/article/645763/Cloud_Computing_2011_Predictions?page=2&taxonomyId=3112, December 2010. Retrieved February 2011.

[15] J. Greene. Yahoo rolls out new e-mail service. http://news.cnet.com/8301-1023_3-20065506-93.html, May 2011. Retrieved May 2011.

[16] 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.

[17] 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.

[18] M. Kan. IBM, China-based firm set to build Asia's largest cloud computing center. http://www.computerworld.com/s/article/9206461/IBM_China_based_firm_set_to_build_Asia_s_largest_cloud_computing_center, January 2011. Retrieved February 2011.

[19] Launchpad Project. Python implementation of SAML2. https://launchpad.net/pysaml2. Accessed April 2011.

[20] E. Lawrence. Fiddler Web Debugger - A free web debugging tool. http://www.fiddler2.com/fiddler2/. Accessed April 2011.

[21] P. Leach, M. Mealling, and R. Salz. RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace. http://www.ietf.org/rfc/rfc4122.txt, July 2005. Retrieved March 2011.

[22] F. H. Mathis. A generalized birthday problem. *SIAM Review*, 33(2):pp. 265–270, 1991.

[23] R. McMillan. Lawsuit: Fired data center worker wiped out TV show. http://www.itworld.com/security/142270/lawsuit-fired-data-center-worker-wiped-out-tv-show, April 2011. Retrieved May 2011.

[24] P. Mell and T. Grance. The NIST Definition of Cloud Computing, Version 15. Technical report, National Institute of Standards and Technology, 2009.

[25] C. Metz. Microsoft backs NASA's open source cloud kit. http://www.theregister.co.uk/2010/10/23/microsoft_vows_hyperv_support_for_openstack/, October 2010. Retrieved February 2011.

[26] C. Metz. The New Linux: OpenStack aims for the heavens. http://www.theregister.co.uk/2011/01/08/openstack/, January 2011. Retrieved February 2011.

[27] A. Michael, F. Armando, G. Rean, D. Anthony, K. Randy, K. Andy, L. Gunho, P. David, R. Ariel, S. Ion, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[28] T. Morgan. NASA and Rackspace open source cloud fluffer. http://www.theregister.co.uk/2010/07/19/nasa_rackspace_openstack/, July 2010. Retrieved February 2011.

[29] Security Requirements for Cryptographic Modules. Technical report, National Institute of Standards and Technology, January 1994. FIPS Pub 140-1. Available at http://csrc.nist.gov/publications/fips/fips1401.htm.

[30] N. Neulinger. Cracklib on SourceForge.net. http://sourceforge.net/projects/cracklib/. Accessed April 2011.

[31] NIST. NIST Cloud Computing Collaboration Site. http://collaborate.nist.gov/twiki-cloud-computing/bin/view/CloudComputing/, 2011. Accessed February 2011.

[32] OpenStack. Containers and Objects. http://docs.openstack.org/cactus/openstack-object-storage/admin/content/containers-and-objects.html. Accessed May 2011.

[33] OpenStack. Create/Update Object. http://docs.openstack.org/cactus/openstack-object-storage/developer/content/create-update-object.html. Accessed May 2011.

[34] OpenStack. Language-Specific API Bindings. http://docs.openstack.org/cactus/openstack-object-storage/admin/content/language-specific-api-bindings.html. Accessed May 2011.

[35] OpenStack. OpenStack Community. http://openstack.org/community/. Accessed February 2011.

[36] OpenStack. OpenStack Compute. http://openstack.org/projects/compute/. Accessed May 2011.

[37] OpenStack. OpenStack Image Service. http://openstack.org/projects/image-service/. Accessed May 2011.

[38] OpenStack. OpenStack Object Storage. http://openstack.org/projects/storage/. Accessed May 2011.

[39] OpenStack. OpenStack Open Source Cloud Computing Software. http://openstack.org/. Accessed February 2011.

[40] OpenStack. The Auth System - Swift v1.2.0 documentation. http://swift.openstack.org/1.2/overview_auth.html. Accessed March 2011.

[41] OpenStack. The Rings - Swift v1.2.0 documentation. http://swift.openstack.org/1.2/overview_ring.html. Accessed April 2011.

[42] OpenStack. Welcome to Glance's documentation! http://glance.openstack.org/. Accessed April 2011.

[43] OpenStack. Welcome to Nova's documentation! http://nova.openstack.org/. Accessed May 2011.

[44] OpenStack. IRC Log for April 26, 2011. http://eavesdrop.openstack.org/irclogs/%23openstack.2011-04-26.log, April 2011. Retrieved April 2011.

[45] OpenStack. IRC Log for May 4, 2011. http://eavesdrop.openstack.org/irclogs/%23openstack.2011-05-04.log, May 2011. Retrieved May 2011.

[46] OpenStack. OpenStack Object Storage. Administrator Guide. Bexar release (Feb. 3, 2011). http://docs.openstack.org/openstack-object-storage/admin/os-objectstorage-admin-book.pdf, February 2011. Retrieved February 2011.

[47] OWASP. How to test session identifier strength with WebScarab. http://www.owasp.org/index.php/How_to_test_session_identifier_strength_with_WebScarab. Accessed March 2011.

[48] OWASP. WebScarab Project. http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project. Accessed March 2011.

[49] PortSwigger Web Security. Burp Sequencer. http://www.portswigger.net/burp/sequencer.html. Accessed March 2011.

[50] J. Purrier. OpenStack Announces Cactus Release. http://www.openstack.org/blog/2011/04/openstack-announces-cactus-release/, April 2011. Retrieved April 2011.

[51] J. Purrier. The OpenStack Bexar Release. http://www.openstack.org/blog/2011/02/the-openstack-bexar-release/, February 2011. Retrieved April 2011.

[52] Python-Crack Documentation. What is crack? http://www.nongnu.org/python-crack/doc/what.html, April 2006. Retrieved April 2011.

[53] Python Software Foundation. hashlib - Secure hashes and message digests. http://docs.python.org/library/hashlib.html. Accessed April 2011.

[54] Python Software Foundation. ndg-xacml. http://pypi.python.org/pypi/ndg-xacml/. Accessed April 2011.

[55] Python Software Foundation. Python-crack. http://pypi.python.org/pypi/python-crack. Accessed April 2011.

[56] Python Software Foundation. sqlite3 - DB-API 2.0 interface for SQLite databases. http://docs.python.org/library/sqlite3.html. Retrieved March 2011.

[57] Rackspace Open Sources Cloud Platform; Announces Plans to Collaborate with NASA and Other Industry Leaders on OpenStack Project. http://www.

openstack.org/press/rackspace-openstack-7-19-2010/, July 2010. Retrieved February 2011.

[58] Radicati Group, Inc. Email Statistics Report, 2010-2014. http://www.radicati.com/wp/wp-content/uploads/2010/04/Email-Statistics-Report-2010-2014-Executive-Summary2.pdf, April 2010. Retrieved May 2011.

[59] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technical report, National Institute of Standards and Technology, April 2010. Publication 820-22. Revision 1a. Available at http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf.

[60] SecurityFocus. SecurityFocus Vulnerability Database. http://www.securityfocus.com/bid. Accessed March 2011.

[61] R. Slipetskyy. Privacy concern: Reseller admins being able to manipulate files from every account. https://answers.launchpad.net/swift/+question/154305, April 2011. Retrieved April 2011.

[62] R. Slipetskyy. Re: Suggestion for data backup/recovery in swift. https://lists.launchpad.net/openstack/msg02664.html, May 2011. Retrieved June 2011.

[63] R. Slipetskyy. Re: suggestion for data location compliance in swift. https://lists.launchpad.net/openstack/msg02642.html, May 2011. Retrieved June 2011.

[64] R. Slipetskyy. Restricting data location to specific zones. https://answers.launchpad.net/swift/+question/155889, May 2011. Retrieved May 2011.

[65] R. Slipetskyy. Some of the libraries that can be reused for OpenStack Auth. https://lists.launchpad.net/openstack/msg02008.html, April 2011. Retrieved May 2011.

[66] R. Slipetskyy. Suggestion for data backup/recovery in swift. https://lists.launchpad.net/openstack/msg02632.html, May 2011. Retrieved June 2011.

[67] R. Slipetskyy. Suggestion for data location compliance in swift. https://lists.launchpad.net/openstack/msg02633.html, May 2011. Retrieved June 2011.

[68] R. Slipetskyy. swauth: illegally obtaining reseller admin credentials via GET v2/account/user call.

https://answers.launchpad.net/swift/+question/150824, March 2011. Retrieved April 2011.

[69] R. Slipetskyy. Two paths to the files hashing to the same value. https://answers.launchpad.net/swift/+question/156307, April 2011. Retrieved April 2011.

[70] S. Spector. OpenStack Austin Release is Out. http://www.openstack.org/blog/2010/10/openstack-austin-release-is-out/, October 2010. Retrieved April 2011.

[71] S. Spector. Announcing Project RedDwarf - Database as a Service. http://www.openstack.org/blog/2011/04/announcing-project-reddwarf-database-as-a-service/, April 2011. Retrieved April 2011.

[72] SQLite. Features. http://www.sqlite.org/features.html. Accessed March 2011.

[73] M. Stevens, A. Lenstra, and B. Weger. Chosen-prefix collisions. http://www.win.tue.nl/hashclash/ChosenPrefixCollisions/, February 2007. Retrieved June 2011.

[74] 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. Retrieved June 2011.

[75] D. Talbot. Security in the Ether. *Technology Review*, pages 36–42, February 2010.

[76] L. Tucker. Cisco joins OpenStack Community. http://blogs.cisco.com/news/cisco-joins-openstack-community/, February 2011. Retrieved February 2011.

[77] Ubuntu Manpage Repository. Ubuntu Manpage: random, urandom - kernel random number source devices. http://manpages.ubuntu.com/manpages/lucid/man4/random.4.html. Accessed April 2011.

[78] U.S. Federal Cloud Computing Market Forecast 2010-2015. http://www.marketresearchmedia.com/2009/05/20/us-federal-cloud-computing-market-forecast-2010-2015/, May 2009. Retrieved February 2011.

[79] D. Watson. XFS undelete HOWTO: How to undelete a file in a linux XFS filesystem. http://linuxwebdev.blogspot.com/2005/06/xfs-undelete-howto-how-to-undelete.html, June 2005. Retrieved June 2011.

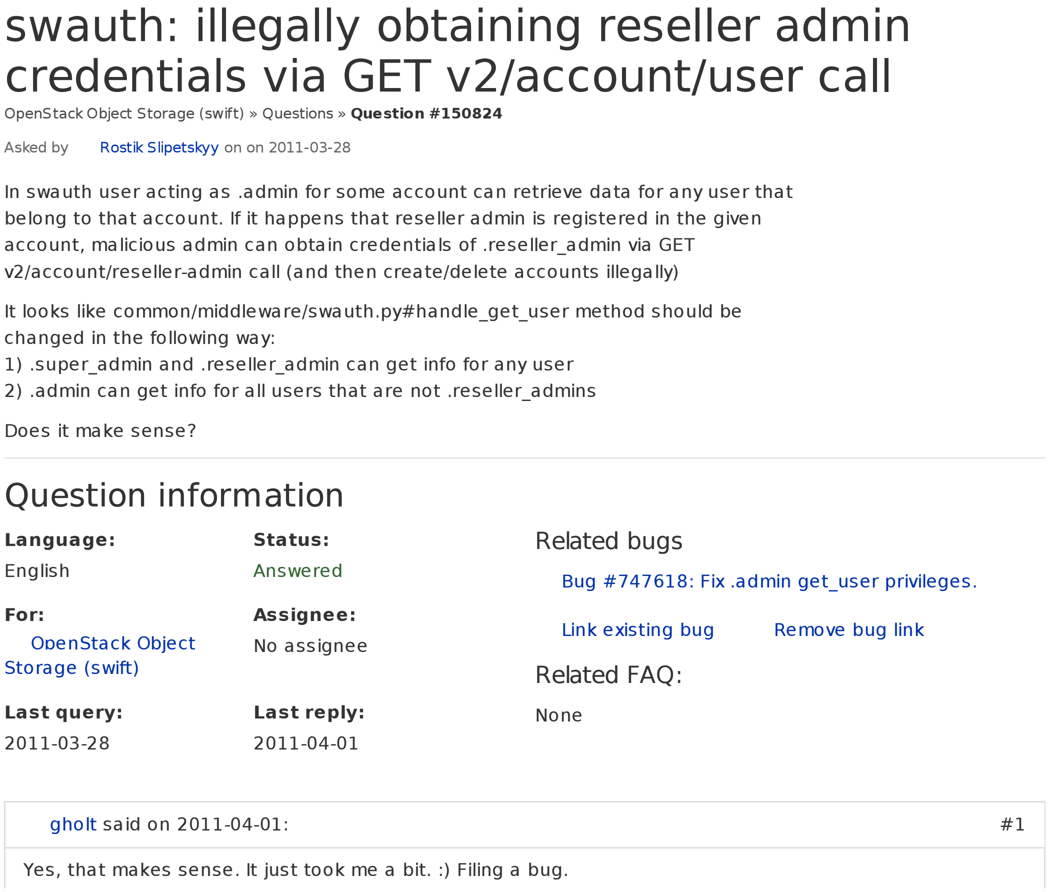
[80] J. Williams. State of OpenStack Auth. https://lists.launchpad.net/openstack/msg01254.html, March 2011. Retrieved March 2011.

[81] Wireshark Foundation. Wireshark. Go Deep. http://www.wireshark.org/. Accessed April 2011.

Appendix A

Notification about Vulnerability that Allows to Obtain Credentials of Reseller Admin

Vulnerability was discovered while analyzing portability of authentication data in OpenStack (section 4.3.6 on page 38). The original of the notification can be found in [68].



swauth: illegally obtaining reseller admin credentials via GET v2/account/user call

OpenStack Object Storage (swift) » Questions » **Question #150824**

Asked by Rostik Slipetskyy on on 2011-03-28

In swauth user acting as .admin for some account can retrieve data for any user that belong to that account. If it happens that reseller admin is registered in the given account, malicious admin can obtain credentials of .reseller_admin via GET v2/account/reseller-admin call (and then create/delete accounts illegally)

It looks like common/middleware/swauth.py#handle_get_user method should be changed in the following way:
1) .super_admin and .reseller_admin can get info for any user
2) .admin can get info for all users that are not .reseller_admins

Does it make sense?

Question information

Language:
English

Status:
Answered

For:
OpenStack Object Storage (swift)

Assignee:
No assignee

Last query:
2011-03-28

Last reply:
2011-04-01

Related bugs

Bug #747618: Fix .admin get_user privileges.

Link existing bug Remove bug link

Related FAQ:

None

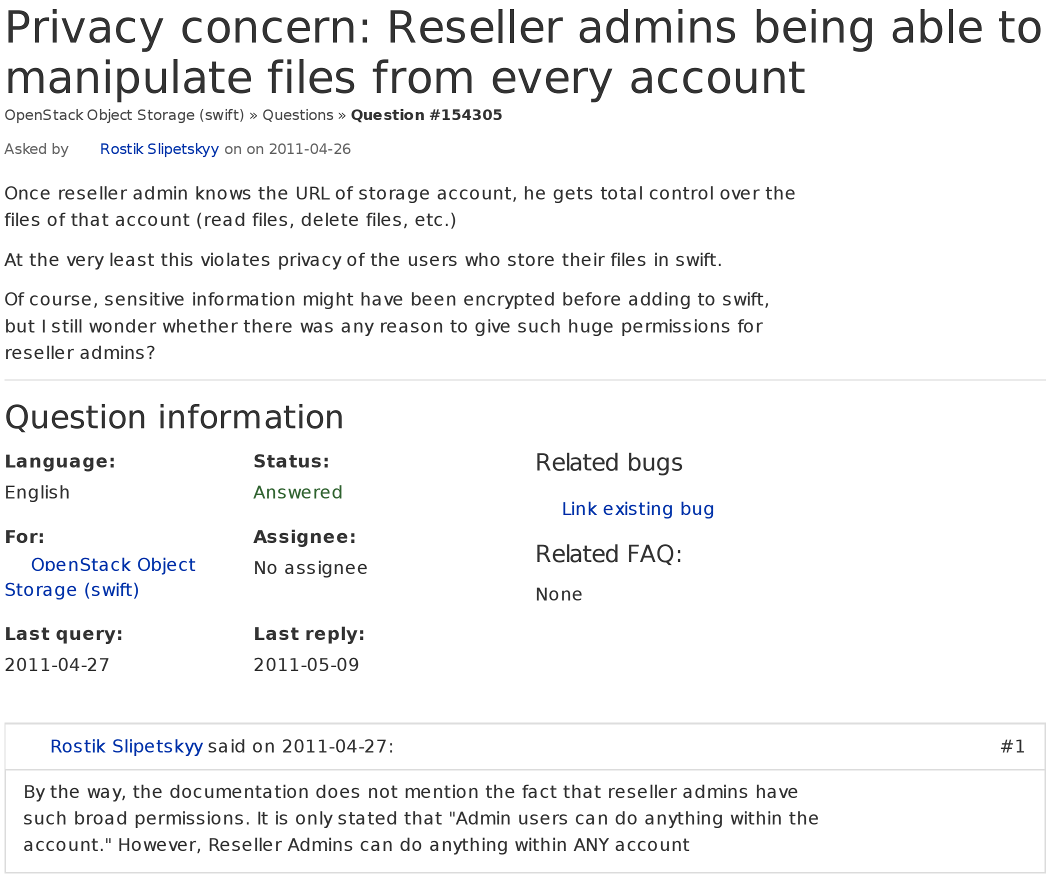
gholt said on 2011-04-01: #1

Yes, that makes sense. It just took me a bit. :) Filing a bug.

Appendix B

Notification about Privacy Concern Arising from Inadequate Permissions of Reseller Admins

Inadequate permissions of *Reseller Admins* were found while analyzing Authorization in OpenStack (section 4.4.3 on page 42). The original of the notification can be found in [61].



Privacy concern: Reseller admins being able to manipulate files from every account

OpenStack Object Storage (swift) » Questions » **Question #154305**

Asked by Rostik Slipetskyy on on 2011-04-26

Once reseller admin knows the URL of storage account, he gets total control over the files of that account (read files, delete files, etc.)

At the very least this violates privacy of the users who store their files in swift.

Of course, sensitive information might have been encrypted before adding to swift, but I still wonder whether there was any reason to give such huge permissions for reseller admins?

Question information

Language:
English

Status:
Answered

For:
OpenStack Object Storage (swift)

Assignee:
No assignee

Last query:
2011-04-27

Last reply:
2011-05-09

Related bugs

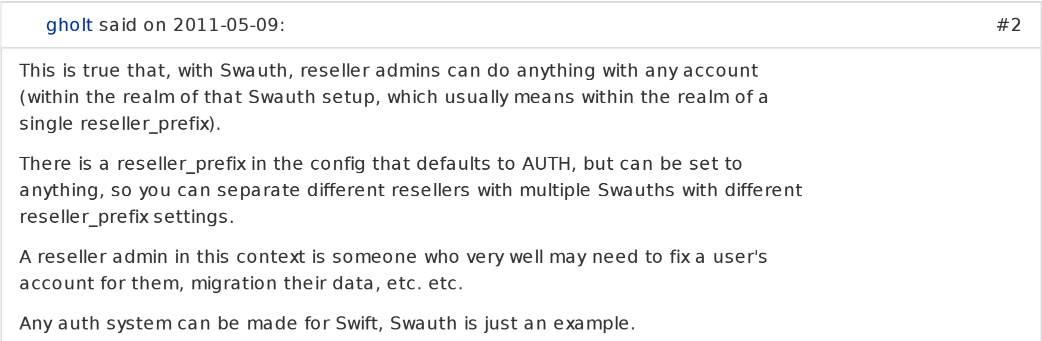
Link existing bug

Related FAQ:

None

Rostik Slipetskyy said on 2011-04-27: #1

By the way, the documentation does not mention the fact that reseller admins have such broad permissions. It is only stated that "Admin users can do anything within the account." However, Reseller Admins can do anything within ANY account



ghoIt said on 2011-05-09: #2

This is true that, with Swauth, reseller admins can do anything with any account (within the realm of that Swauth setup, which usually means within the realm of a single reseller_prefix).

There is a reseller_prefix in the config that defaults to AUTH, but can be set to anything, so you can separate different resellers with multiple Swauths with different reseller_prefix settings.

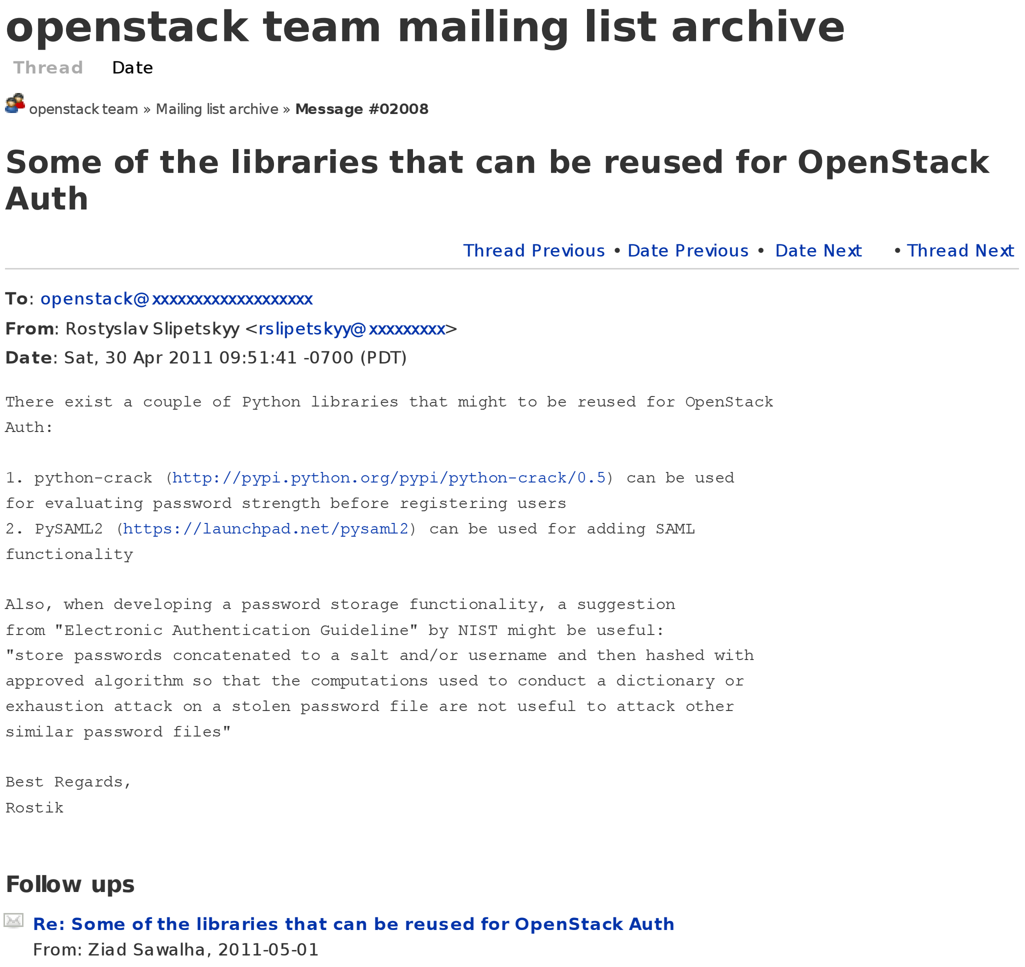
A reseller admin in this context is someone who very well may need to fix a user's account for them, migration their data, etc. etc.

Any auth system can be made for Swift, Swauth is just an example.

Appendix C

Suggestion for Using Third-Party Libraries for OpenStack Authentication

Suggestions presented here were gathered as a result of analyzing Authentication in OpenStack (section 4.3 on page 30). The original of the message can be found in [65].



openstack team mailing list archive

Thread Date

openstack team » Mailing list archive » **Message #02008**

Some of the libraries that can be reused for OpenStack Auth

Thread Previous • Date Previous • Date Next • Thread Next

To: openstack@xxxxxxxxxxxxxxxxxxx
From: Rostyslav Slipetskyy <rslipetskyy@xxxxxxxxx>
Date: Sat, 30 Apr 2011 09:51:41 -0700 (PDT)

```
There exist a couple of Python libraries that might to be reused for OpenStack
Auth:

1. python-crack (http://pypi.python.org/pypi/python-crack/0.5) can be used
for evaluating password strength before registering users
2. PySAML2 (https://launchpad.net/pysaml2) can be used for adding SAML
functionality

Also, when developing a password storage functionality, a suggestion
from "Electronic Authentication Guideline" by NIST might be useful:
"store passwords concatenated to a salt and/or username and then hashed with
approved algorithm so that the computations used to conduct a dictionary or
exhaustion attack on a stolen password file are not useful to attack other
similar password files"

Best Regards,
Rostik
```

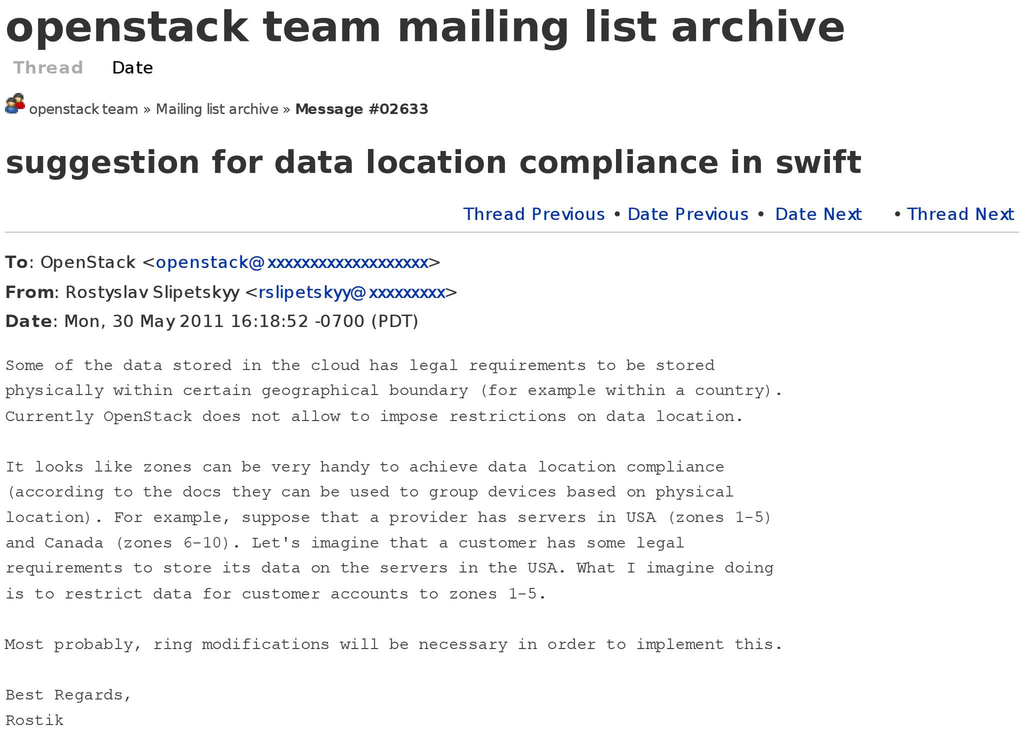
Follow ups

Re: Some of the libraries that can be reused for OpenStack Auth
From: Ziad Sawalha, 2011-05-01

Appendix D

Suggestion for Using Zones for Data Location Compliance in OpenStack

Suggestions presented here were gathered as a result of analyzing data location compliance in OpenStack (section 5.1.3 on page 51). The information below is only an extract from the mailing list discussion [67, 63].


Thread
Date
openstack team » Mailing list archive » **Message #02633**
Thread Previous • Date Previous • Date Next • Thread Next

openstack team mailing list archive

suggestion for data location compliance in swift

To: OpenStack <openstack@xxxxxxxxxxxxxxxxxxx>
From: Rostyslav Slipetskyy <rslipetskyy@xxxxxxxxx>
Date: Mon, 30 May 2011 16:18:52 -0700 (PDT)

```
Some of the data stored in the cloud has legal requirements to be stored
physically within certain geographical boundary (for example within a country).
Currently OpenStack does not allow to impose restrictions on data location.

It looks like zones can be very handy to achieve data location compliance
(according to the docs they can be used to group devices based on physical
location). For example, suppose that a provider has servers in USA (zones 1-5)
and Canada (zones 6-10). Let's imagine that a customer has some legal
requirements to store its data on the servers in the USA. What I imagine doing
is to restrict data for customer accounts to zones 1-5.

Most probably, ring modifications will be necessary in order to implement this.

Best Regards,
Rostik
```

Re: suggestion for data location compliance in swift

Thread Previous • Date Previous • Date Next • Thread Next

To: Joshua McKenty <josh@xxxxxxxxx>
From: Rostyslav Slipetskyy <rslipetskyy@xxxxxxxxx>
Date: Tue, 31 May 2011 06:37:53 -0700 (PDT)
Cc: OpenStack <openstack@xxxxxxxxxxxxxxxxxxx>
In-reply-to: <92BA24A5-9448-4E35-A421-E8B221C07DD1@piston.cc>

The idea to make ring implementation pluggable seems nice.

At the same time I am thinking that many developers might not will feel comfortable with modifying existing ring structure, since it *works* :) Probably, the other viable option for allowing data location compliance is to implement it outside of ring file structure (but maybe inside the future Ring component/service).

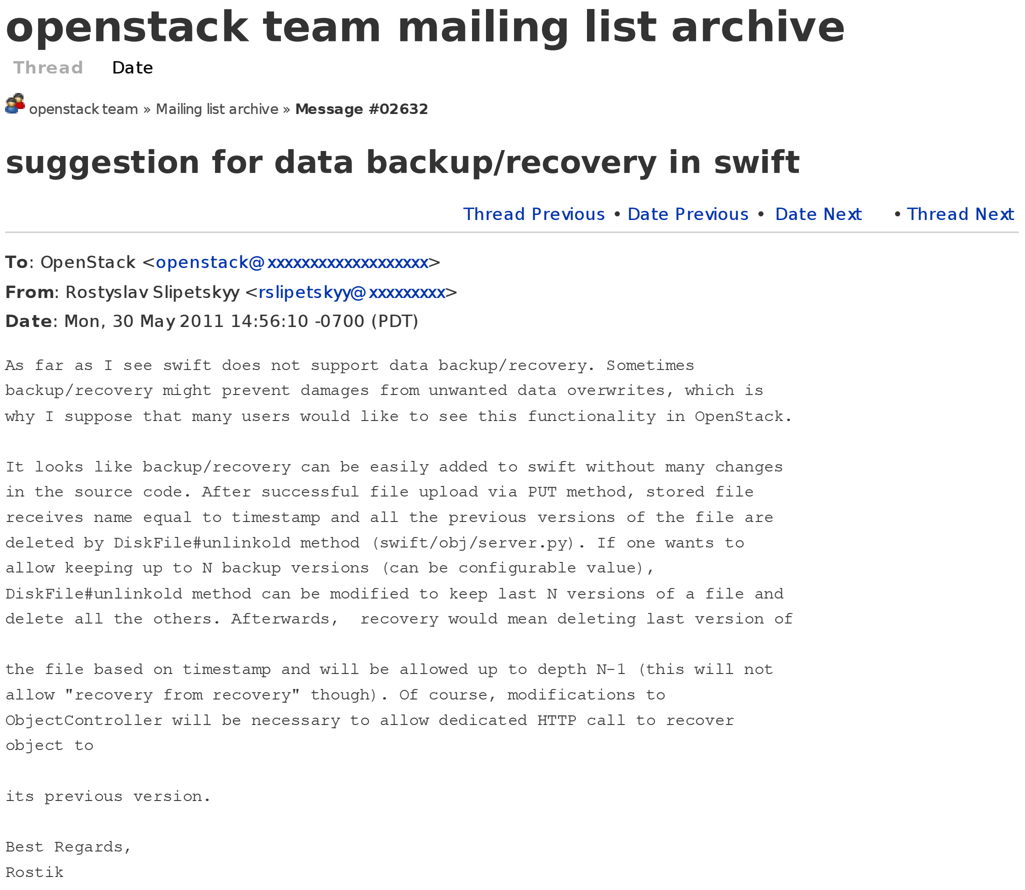
As an example I look at how replication works, "If a replicator detects that a remote drive is has failed, it will use the ring's “get_more_nodes” interface to choose an alternate node to synchronize with." It seems that "Ring#get_more_nodes" can be used in a similar manner to select alternative nodes in other zones for storing objects once some of the zones are banned for specific accounts.

- Rostik

Appendix E

Suggestion for Implementing Backup/Recovery in OpenStack

Suggestions presented here were gathered as a result of analyzing backup and recovery in OpenStack (section 5.3 on page 57). The information below is only an extract from the mailing list discussion [66, 62].



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@xxxxxxxxxxxxxxxxxxx>
From: Rostyslav Slipetskyy <rslipetskyy@xxxxxxxxx>
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 Slipetskyy <rslipetskyy@xxxxxxxxx>
Date: Tue, 31 May 2011 14:49:21 -0700 (PDT)
Cc: OpenStack <openstack@xxxxxxxxxxxxxxxxxxx>
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 F

Configuration Files Used with Test Deployment of OpenStack Object Storage

Base configuration file for each node in OpenStack Object Storage:

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

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

```
[DEFAULT]
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
bind_port = 8080
workers = 4
user = swift

[pipeline:main]
pipeline = healthcheck cache auth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
set log_name = swift-proxy
set log_level = DEBUG

[filter:auth]
use = egg:swift#auth
ssl = true

[filter:healthcheck]
```

```
use = egg:swift#healthcheck

[filter:cache]
use = egg:swift#memcache
memcache_servers = 10.0.0.2:11211
```

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

```
[DEFAULT]
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
bind_port = 8080
workers = 4
user = swift

[pipeline:main]
pipeline = healthcheck cache swauth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
set log_name = swift-proxy
set log_level = DEBUG
set log_headers = True

[filter:swauth]
use = egg:swift#swauth
set log_name = swift-swauth
set log_level = DEBUG
set log_headers = True
default_swift_cluster = local#https://10.0.0.2:8080/v1
super_admin_key = swauth

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:cache]
use = egg:swift#memcache
memcache_servers = 10.0.0.2:11211
```

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

```
[DEFAULT]
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
user = swift

[pipeline:main]
```

```
pipeline = auth-server

[app:auth-server]
use = egg:swift#auth
default_cluster_url = https://10.0.0.2:8080/v1
super_admin_key = devauth
set log_name = swift-auth
set log_level = DEBUG
```

Configuration files for Account service on Storage node:

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

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

Configuration files for Container service on Storage node:

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

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

Configuration files for Object service on Storage node:

```
[DEFAULT]
bind_ip = 10.0.0.5
```

```
workers = 2

[pipeline:main]
pipeline = object−server

[app:object−server]
use = egg:swift#object

[object−replicator]

[object−updater]

[object−auditor]
```

Appendix G

Modified Source Code of WebScarab Session ID Analysis Plugin

We have used this modified code of WebScarab Session ID Analysis Plugin from OWASP to test randomness of OpenStack authentication tokens (see section 4.3.5).

The below source code includes only the changed method. For copyright information and the remainder of the code, please consult the original source code of WebScarab.

```
public class SessionIDAnalysis implements Plugin, ConversationHandler {

    /* ... */

    public Map getIDsFromResponse(Response response, String name, String regex) {
  Map ids = new TreeMap();
        Request request = response.getRequest();
        if (request == null) {
            System.out.println("Request was null?");
            return ids;
        }
        HttpUrl url = request.getURL();
        Date date = new Date();
        NamedValue[] headers = response.getHeaders();
        if (name != null && !name.equals("") && regex != null) {
            // check if session should be found in a header
   String sessionInHeader = response.getHeader(name);
   if (sessionInHeader != null && !sessionInHeader.equals("")) {
    System.out.println("Session ID found in header: " + name);
    if (regex == null || regex.equals("")) {
     SessionID id = new SessionID(date, sessionInHeader);
     ids.put(name, id);
```

```
 } else {
  // regex should be used to extract session value
  Pattern pattern = Pattern.compile(regex);
  Matcher matcher = pattern.matcher(sessionInHeader);
  if (matcher.matches() && matcher.groupCount() > 0) {
   for (int j=1; j<=matcher.groupCount(); j++) {
    SessionID id = new SessionID(date, matcher.group(j));
    ids.put(name + " " + j, id);
   }
  }
 }
}
//
String location = response.getHeader("Location");
         if (location != null) {
             Pattern pattern = Pattern.compile(regex);
             Matcher matcher = pattern.matcher(location);
             if (matcher.matches() && matcher.groupCount() >
                0) {
                 for (int j=1; j<=matcher.groupCount(); j++)
                    {
                     SessionID id = new SessionID(date,
                        matcher.group(j));
                     ids.put(name + " " + j, id);
                 }
             }
         }
         String type = response.getHeader("Content-Type");
         if (type != null && type.startsWith("text/")) {
             String charset = "UTF-8";
             String body = null;
             try {
                 body = new String(response.getContent(),
                    charset);
             } catch (UnsupportedEncodingException uee) {
                 body = new String(response.getContent());
             }

             Pattern pattern = Pattern.compile(regex, Pattern
                .MULTILINE | Pattern.DOTALL);
             Matcher matcher = pattern.matcher(body);
             if (matcher.matches() && matcher.groupCount() >
                0) {
                 for (int j=1; j<=matcher.groupCount(); j++)
                    {
                     SessionID id = new SessionID(date,
                        matcher.group(j));
```

```java
                        ids.put(name + " " + j, id);
                    }
                }
            }
        } else {
            Pattern pattern = Pattern.compile("(.*)");
            if (regex != null && !regex.equals("")) pattern = Pattern.compile(regex);
            for (int i=0; i<headers.length; i++) {
                if (headers[i].getName().equalsIgnoreCase("Set-Cookie") || headers[i].getName().equalsIgnoreCase("Set-Cookie2")) {
                    Cookie cookie = new Cookie(date, url, headers[i].getValue());
                    Matcher matcher = pattern.matcher(cookie.getValue());
                    name = cookie.getKey();
                    if (matcher.matches()) {
                        SessionID id = new SessionID(date, matcher.group(0));
                        ids.put(name, id);
                        if (matcher.groupCount() > 0) {
                            for (int j=1; j<=matcher.groupCount(); j++) {
                                if (!matcher.group(j).equals(matcher.group(0))) {
                                    id = new SessionID(date, matcher.group(j));
                                    ids.put(name + " " + j, id);
                                }
                            }
                        }
                    }
                }
            }
        }
        return ids;
    }
}
```

Appendix H

Dummy Implementation of md5 Hash Function for Testing OpenStack Isolation

We have used this dummy implementation of md5 hash function in order to test OpenStack isolation (see section 5.2.3):

```
class md5():
 def __init__(self, str):
  pass

 def digest(self):
  return 'abcdef'

 def hexdigest(self):
  return 'abcdef'
```