ISSN 2347 - 3983
Volume 8. No. 3, March 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter49832020.pdf
https://doi.org/10.30534/ijeter/2020/49832020

Enhancement of Security within OpenStack – Some measures

Dr.JKR Sastry[1],B. Trinath Basu[2]
[1]Koneru Lakshmaiah Education Foundation, Vaddeswaram, India, drsastry@kluniversity.in
[2]Koneru Lakshmaiah Education Foundation, Vaddeswaram, India, miriiyala68@kluniversity.in

ABSTRACT

Cloud computing is being used rapidly by the businesses around the world. Businesses are migrating the existing IT infrastructure to Cloud based Infrastructure offed third party service providers due to cost and availability of the latest technology at their door front. While that being the Security of the data and software stored on the Cloud has been a real threat and still many issues in relation to the security enforcement within cloud based infrastructure has been the real issue that must be tackled so that the users of the cloud based infrastructure are sure of the confidentiality of their data and software.

OpenStack is open source software that can be used by anybody for building private cloud. Use of OpenStack minimizes the cost required almost to nil for building once own private clouds. Many security related issues crops up when Open stack is used due existence of several vulnerabilities Security enhancements are required to make Open Stack fully reliable in maintaining the secrecy and confidentially of the data stored in Open Stack.

This paper elaborates on the security lapses exposed within OpenStack and proposes various enhancements to the existing provisions so that OpenStack is fully secured without compromising on Service level conditions.
The Existing Architectures of open stack are expanded through addition of more components and the interactions between the new and existing components

Key words: Cloud Commuting, Open Stack, Security provisions, security enhancements, Vulnerabilities existing in Open Stack

1. INTRODUCTION

1.1 Introduction to open stack

Cloud computing implies provision of the entire infrastructure required for the user to implement their business application and access the application from location. Cloud computing implies making available software platform and infrastructure as a service when demanded by the user. Open Stack is cloud computing system that provides infrastructure as service. Users can create their own machine instances on which other open stack and user defined software components run. Cloud computing infrastructure provides a platform using which the users develop their own software and run on the instance which are allocated to the users.

Open Stack can be defined as a set of tools which are used for building private and public clouds. Open Stack is open source which is developed by Open Stack foundation and since the software is open source users can enhance the functionality of the same through addition and Integration user developed components

Users can create their own virtual machines which can run parallel servicing more users. The Virtual machines are created as and when the users make a demand for the same. The computing required is horizontally expanded as more number of users gets added into the system. User has the access to the source code using which the cloud computing system is developed. Users can make changes to the source code and share the modifications made to all the other users who use Open Stack software for building their internal cloud. Thus Open Stack is supported by thousands of the users around the world. Open stack has become robust due to involvement of many users in the world.

However there are inherent disadvantages due to involvement of too many independent developers.Original Open Stack is offered containing 9 Key components which form the core of the software and from there many components have been added by other users existing in the world. The core of Open stack is maintained and distributed by Open Stack Foundation.

The Nine Components of OpenStack include Keystone which provides Identityservice, NOVA that provides Computing Service, Neutron that provides networking services, Glance that Provides Image based data management services, SWIFT that provides object storage services, Cinder that provides Block storage Services, Trove that provides conventional database services, Heat that provides Orchestration services,

Horizon that provides dashboard services and Ceilometer that provides billing Services. Many other type of components have been introduced subsequently some of which include Ironic that provide services to run directly on physical servers known as bare-metal services, Sahara that provides data processing services, Zaquar that provides Messaging services, and Barbican that provides Security Infrastructure. Figure 1 shows the overall architecture of

OpenStack and Table 1 shows the details of the components added till date into Open Stack.

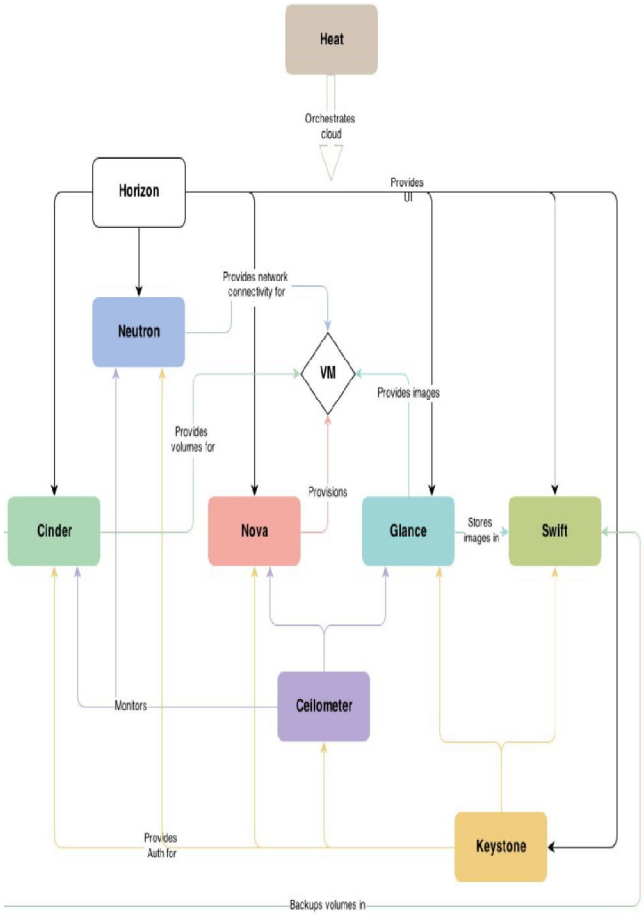

Figure 1: Overall Architecture of the Open Stack.

All OpenStack services expose a RESTful API for communication among them and use HTTP protocol for each data exchange. However, the real communication happens between the internal processes of the services that are dedicated for certain specific execution required. The communication happens using AMQP (advanced Messaging protocol) and using a message bus which is RabbitMQ by default. Every service / module uses SQL based database for storing state information. Multiple SQL databases can be used when more throughputs is required

1.2 Security Issues in Open Stack

Each of the Open Stack components delivers a service meaning a functionality which can be individually attacked. Each of modules must be analyzed to find the extent of security implemented within the Module and the Kind of Vulnerabilities Exposed by each module. Counter Systems required can be analyzed and find the extent to which the mechanism either proposed or implemented as on date. As such the issue of Security is covered by Keystone Module to the extent of Authentication and Authorization and the issue of data security dealt by the modules that deals with data storage services which include GLANCE that manages the images, SWIFT that manages Object storage, CINDER that Manages Block Storage and TROVE that Manages Conventional Databases.

1.2.1 Identity services

Enforcing security requires the Identification of the users and the Groups to which they belong. It is also important to assign roles to the users and the kind of accessing the users can have based on their roles. Keystone provides the Identify services to all the other components supported within Open Stack. Keystone maintains a catalogue of all the services and the association of services to the users having some access rights. Keystone maintains accounts to which the users, projects, user Groups and the services that can be accessed through those accounts. Key Stone also maintains API end points where the services are located. Users can access the services using the API end Points. Users are treated as digital representations of a system, person or a service.

Keystone receives the user requires and then validates that the requests or genuine being initiated by the valid users having a specific roles that has the access to the service that the user is requesting. Keystone assigns a Token to the users on finding that the user is valid. The Token contains all the details that include the roles that the users can play, the kind of services that they can avail.

User roles essentially define the kind of operations that the users can perform. Every service component also find whether the service can be extended to the user based on the service policies maintained for each of the service separately. Service polices are stored in JSON format in a policy. son file. The user credentials are made known to the users as data and the access to the services are provided through the end points only. Figure 2 shows the interaction of the KEYSTONE component with other components.

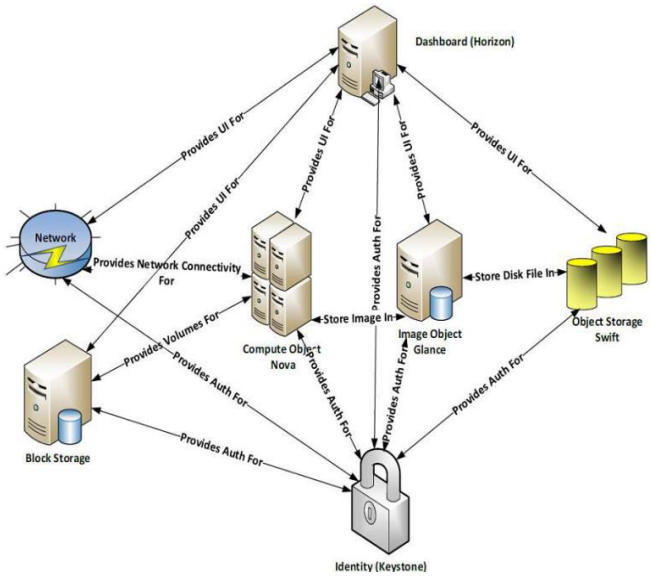

Figure 2: Overall Interactions within Key Modules

Following steps are followed when a user tries to access an OPEN STACK service.

1. Users send their user name and password to the keystone
2. Keystone checks the validity of the same and if found OK, sends a temporary token and list of tenants that the user can opt. Tenant is like an account to which the users, user groups , roles and resources as assigned
3. Users send to the keystone the Temporary token and the desired Tenant
4. Keystone on verifying the identity using the temporary token and the desired tenant sends a Token that has all the details of the roles, services and end points.
5. User selects an end point based on the service that they desire. End point is a kind of URL provided to access the service, something like a WEB interface.
6. User's initiates access to the services through imitation the end points through sending the token sent by Keystone
7. The Individual services verifies the token, acquires the access provided to the token and checks whether the request made is in line with polices maintained separately for the service.
8. If the request is found OK, the service is provided.

Figure 3 shows the architecture of the keystone Module.

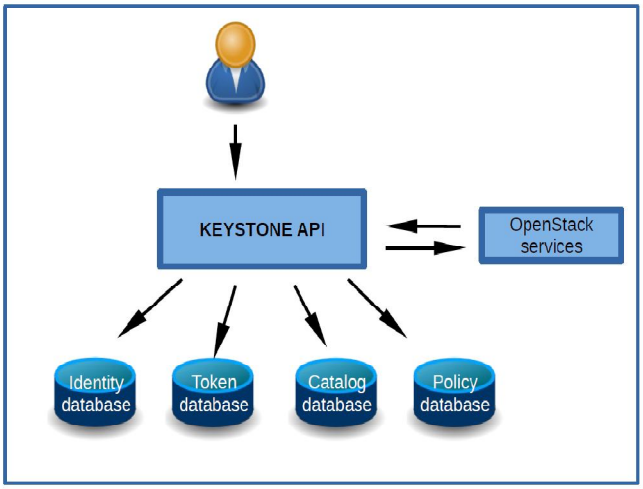

Figure 3: Keystone architecture

The Keystone module is built using 4 sub-components that provide several services that include catalogue services, token services, Identity database services, and overall policy services. The services interact using its internal API. The Identify databases services provide the validation of credentials of the users and also validate the users with reference to the Tenants to which they belong. Tokenisation service related to creating and delivering the tokens to the users who are found to be valid users, Catalogue service maintains an end point registry. The role and rule based authorisation engine is maintained by Policy service.

Security issue

Keystone uses two systems tempAuth and swAuth for checking the credentials of the users which is weak. The users can apply brute force attack to know the users names and passwords used by the users. No authentication delegation system is used by KEYSTONE. It does not exploit the exiting highly sophisticated authentication system. The Usernames and passwords are stored in files that can be attacked, Open Stack does not follow NIST rules that include passwords length, use of different characters, use of upper and low case letters etc. for choosing the passwords and as a result the passwords can be easily attacked. Open Stack also do not carry any dictionary check on the passwords used by the users making it easy for attacking. Open stack stores the passwords in plain text. The user name and password of admin and super user also are stored in the file in the plain text making it easy to get hold of the data by the attackers. File based access restrictions must be imposed and the passwords must be also stored in encrypted manner.

The data sent to Users upon authentication through tokens reveals many aspects of the system usage, user, user roles, services, end points etc. The tokens are issues through use of WEB interface making it easy for the user to attack the token. The tokens are not delivered through secured channels. Transport Layer Security (TLS) is not used for delivering the tokens. It is easy for a man-in-the-middle to attack the token.

1.2 2 Virtual Image management services

The essence of any cloud computing is making available virtual machine for running the user defined tasks. Every virtual machine is stored as image using a specific format. The image of Virtual machine contains OS Code, memory maps, catalogues, and file buffers. Every time a VM is provisioned to a user, the VM is loaded into physical server and made to run. For all practical purposes, a VM can be treated as a task that runs in the Physical server.

Entire behaviour of the VM is dependent on the VM image and therefore a tight security is needed to preserve the image so that no attacking on the Image takes place.

The module GLANCE provides the imaging services. The imaging services provided by GLANCE include discovering, registering and retrieving the Virtual machine images. Glance generates VM images based on the user requirements and stores in its database repository. Interaction with Glance can be done using RESTful API that allows querying of VM image metadata as well as retrieval of the actual image.

The heart of cloud computing system is virtual machines. Predefined defined and pre-stored VM images helps in easy provision of the VMs to the users. The GLANCE module

manages the images. VM images are stored in different formats depending on the kind of hypervisor used. Images must be protected. Any attempt to sabotage the image will spoil the functioning of the entire cloud computing system.

The internal architecture of GLANCE is shown in Figure 4. The glance module contained several components for providing different internal services within it. The service "REST **API"** exposes Glance functionalities. The Module has its internal sub-components that include Domain controller (DC), Database Abstraction Service (DAL), Storage service (Glance Store - GS), Registry Service (RL). DAL provides communication interface between Glance databases and Glance services. GDC provides the authentication, notification, database connectivity and maintenance of policies for making available the VM Images. GS provides the database interaction services. RL provide secure communication between DC and the DAL.

Security Issue

GLANCE is dependent on SWIFT for managing the VM Images. The vulnerabilities that exist for SWIFT also are applicable to GLANCE.

The VM mages may be compromised when the Hypervisor that is responsible for dealing with the images is attacked. The attackers can make the Hypervisor in-operable, or exploit to retrieve the personal information of the other. The Hypervisor can be attacked through a DDOS (distributed denial of service) which can be caused by sending too much of a traffic to the VMS which cannot be handled by the Hypervisor and as a results the Hypervisor can be broken out.

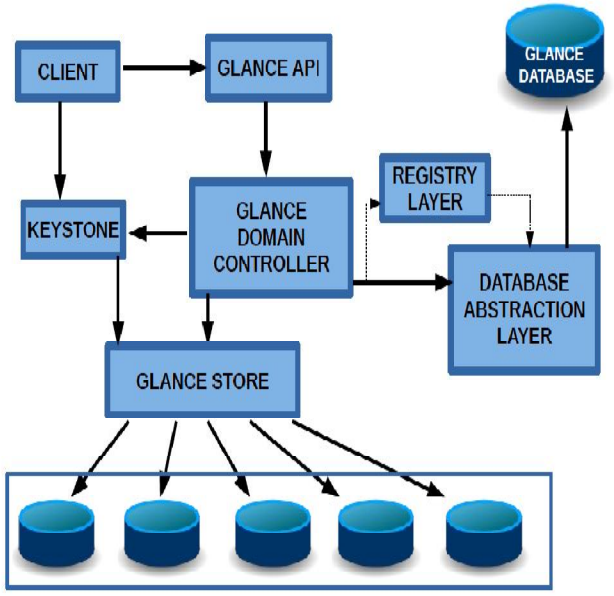

Figure 4 : Internal Architecture of GLANCE modules.

1.2.3 Data Management services

In Open stack data related services are supported through three modules that include SWIFT, CINDER, and TROVE. Swift manages objects storages. Objects handled by SWIFT include Images and files. SWIFT follows a kind of directory hierarchy for containing the data in backend storage devices. Security to access the objects stored by SWIFT is limited to make the user know the location of storage of the object. The vulnerabilities that exist in the way data is manged will be discussed in subsequent sections.

Object Storage Management Services

Users can manage of their objects which can be a file, document, programme anywhere within the Open Stack server either located at one place or distributed across the Open Stack system. The users can carry all kinds of operations that include create, modify, and get the objects and the related Meta data through Use of API which are implemented through REST (Representational State Transfer Services). The data related to objects are stored as shown in Figure 5. The storage is divided into several directories each for holding an Account. Each account is further segmented down to a set of containers and each Container a set of objects along with their access control lists. An account is like a Tenant and refers to a name space for the containers and the containers acts like a name space for the objects contained in it.

ACL (access control lists) related the objects placed in the containers. ACL is the mechanism using which access to the objects contained in it is achieved. Objects which hold data content, images, videos etc. are placed in the container directory. The container can also store a metadata objects that describes several objects contained in the container.

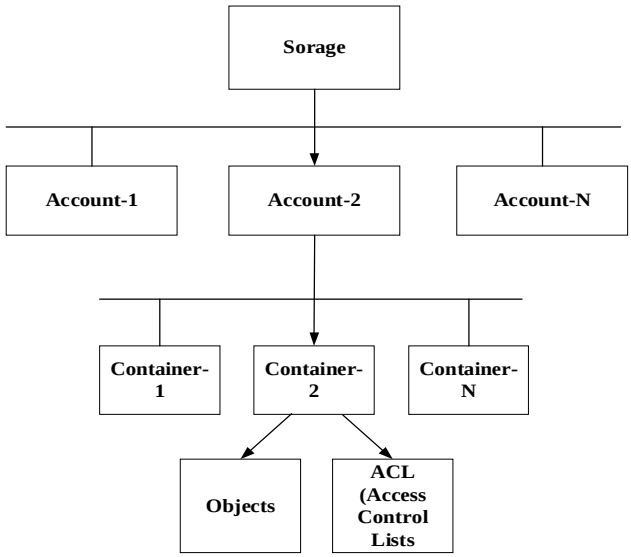

Figure 5 : Data Organisation within SWIFT

OpenStack Swift is implemented using client-server architecture. SWIFT is developed with sub-components in it that include REST API, A Proxy and a Storage Service. API provides the interface required to call the service required for making operations on the objects. All the service requests are received by Proxy and then the most suitable API is made and also communicates with the Storage service for making object related oper within the account and container context. SWIFT module is constructed with a Proxy server, Servers to manage accounts, containers and objects and also data base repositories to store the details of accounts, containers and objects**.**

Figure 6 shows the way data is retrieved using SWIFT Module. Users access the object using HTTP calls by providing logical paths to accounts, containers and objects. The Logical path of the storage required is provided by the authentication server and then the name of the object to be referred by is appended to the Logical path using a ”/” convention.

The physical location of the object that resides on a cluster is translated by Proxy server using the logical path using the “rings” concept. Open stack divides cluster of storage devices into partitions and allocates each partition into devices.

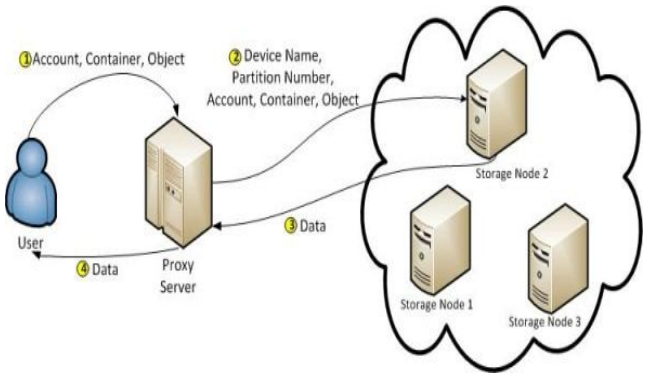

Figure 6: Data Retrieval in OpenStack Object Storage

Rings

SWIFT uses a concept called ring. For each account, container and set of objects a ring is maintained. A ring for a set of objects represents a policy that allows the objects to be accessed provided the user requests confirms to the policy. A ring as such maps to a physical entity stored in a disk and the location where the entity is stored. A ring stores the location of an object in terms of a zone, devices, partitions, and replicas.

Each partition is replicated and stored in three different physical locations in the clusters. Every time a user needs an entity, the ring must be contacted to find its physical location. Ring stores information about the devices to which control must be passed in case some error occurs while accessing the data.

Every account is treated as a device. Weights are assigned to device. The objects contained in a account are distributed across the cluster of servers according to the Weight. The devices with higher weights are more distributed. Device weights are used to move the device partitions across the clusters. The ring will help to move minimum partitions moved at a time.

Storage Policies

SWIFT uses storage policies that can be configured by the users. The storage polices decides the naming of the objects and also defines the service levels to each device. Different service policies are assigned to each device. For each of the service entrusted to a device, a separate service ring is maintained which also maintains the information related to the hardware entrusted to the device. Storage polices can be extended to the container also, which means a set of storage policies are related to a container. The storage policies defined for container are automatically attributed to all the objects stored within the container.

SWIFT servers

SWIFT uses three servers that include Object server, Container Server and a Account server. Object server provides the services required for storing, deleting and retrieving the objects. Objects are stored as binary files and the metadata of the objects are stored as attributes of the files. Each of the objects is stored using a pathname that is derived through hash on the name of the object and the also using a time stamp attached to the path name. The addition and deletions are treated as version of the files; Thus SWIFT stores several versions of the objects

The container server manages the listings of the objects. The Listings are stored like SQL Database files. Statistics on the objects contained in the container are also stored in the container. Accounts server maintains the listings of the containers.

Operating the Objects within the containers

OpenStack Object Storage management is based on the roles assigned to the users. Various kinds of roles are defined with the Object storage component which includes “Admin”, “Reseller Admin”, “Super Admin”. No user is allowed to have user administration role. The main concept is based on the accounts that are created at the time of entering into SLA agreements.

Users can be added to the accounts by the persons assigned with the role “Admin”. Users can be added or deleted by “Admin” persons interfaced with seAUth () API for addition seAuth ( ) API for deletion.

Security issue

The user makes a request for data resources through a definition of account/container/object, ring name and policy and the proxy server validates the correctness of the association and then when found good is allowed to be streamed to the user through invoking respected server based on the kind of service requested by the user. The information made available about the accounts/containers/objects, rings and polices must be preserved. If this information is leaked, the un-intended users will get the access to the data.

The data stored in SWIFT is not encrypted, thus providing a scope for attacking making it necessary to implement the systems required for securing the data.

The information related to the users, user groups, roles and role based access is critical to making the object management within swift. Any leak on the access information will affect the objects stored in the containers

A hard disk situated on a server that is included in a cluster is regarded as a device. Each device is recognised as a partition which is maintained redundantly in order to take care of disk failures. The ring structure decided the node on which the replication has to be undertaken. The mapping of partitions to the nodes is stored in the File that stores a ring structure in it. The

The physical location of an entity is found by proxy server by contacting a server on the storage node. The proxy server is built using processes that are responsible for managing te containers, accounts and the objects. Details of logical location of the storage are sent to the entity service processor. The processor finds the location of the object by using a hash function.

Ring files and hash_path_suffix are the most important information in locating an object. Ring files are situated at in /etc/swift/account.ring.gz and Hash path suffix is stored at /etc/swift/swift.conffile. These files can be attacked by an attacker by changing hash value

In a storage device different directories exists for 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. In each of the directory more sub-directories exists each representing a partition.

The directory is named using numbers. Data isolation is done using the hash function that uniquely determines the location where the object is stored. While data isolation is possible the confidentiality of the data stored must be maintained by the user who seeks to store the objects in Object storage using SWIFT module.

Block storage Management services

OpenStack has in it a separate module “CINDER” that caters for providing the block storage.

CINDER provides software which can be called through RESTful API for provisioning and managing storage in the form of block of devices called as cinder Volumes. CINDER provides persistent storage to VM instances. User applications can also ask for cinder volumes. User applications can also request for an array of volumes which have differing characteristics. It is possible that the user to define differing characteristics of the volumes. The differing characteristics include mirroring, high performance, clustering etc.

Cinder component deals with Block storage such as dealing with Volumes. The storage devices supported on physical machines can be attached to cinder server nodes. Volumes that are created on third party storage devices can be attached to the cinder servers. Cinders plug-in architecture allows any third party volumes to be attached to the Cinder servers. The connection between the Compute nodes and the storage devices attached to the Cinder Servers can be achieved through use SCSI, NFS or Ethernet. The architecture of CINDER module is shown in **Figure 7**.

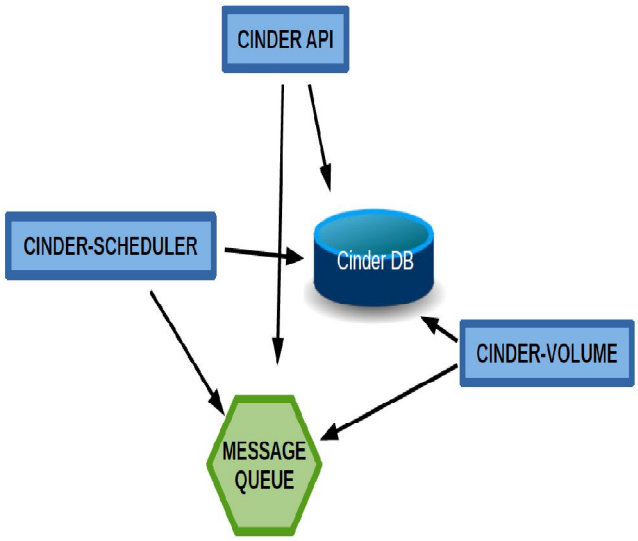

Figure 7 : Block Storage Architecture

Cinder Module is developed with four components that include RESTful API (CINDER-API) that provides interface to manipulate Volumes and Snap-Shots, a component “Cinder Volume” which is responsible for reading and writing the CINDER database to manage the volume state, Cinder-Volume is also responsible to interact with CINDER scheduler and the driver used for storing the data. The Component “CINDER-SCHEDULER” is responsible for selecting the best storing block when a volume is created. The Component “CINDER-DATABASE”is responsible for storing volume state. A

message queue is used to facilitate communication between the user, CINDER volume, and CINDER scheduler.

Security issue

User request for a volume with certain blocks of storage and the user is given with the Volume details. Anybody who has the access to the Volume details can have access to the volume and tamper with it.

Most cloud providers do not encrypt data before saving it to storage. In fact, OpenStack does not provide any data encryption at all; thus, users would need to encrypt their data before uploading it and manage their encryption keys themselves.

Conventional Database services through TROVE Component

OpenStack comes with another important module called TROVE. Any user who wants to store the data as a relational database can use this service; Trove provides relational database. The Internal database used is SQLserver. All the database administration required is done by TROVE relieving the users from such a burden. All the Tasks that can be undertaken using SQL server can also be undertaken using TROVE. Multiple instance of the database can be provisioned by the users and Open stack administrators.

Trove is designed for taking care of data isolation with database while the users take cate of the issues that include Database creation, taking backups, data retrievals and Monitoring. The module TROVE is built using four components include “Trove-API” that provides interface to the VM based applications through RESTful API that supports JSON and XML to provision andmanage Trove instances.

Creation and management of database instances and carrying different database operations are undertaken by Task manager (T-M). The operations on the database arecarried by Guest Agent (GA). GA acquires the database operation related messages through RPC related messages and the required database operation is performed by the agent. The messages are first received by Conductor which is the sub-component of the Trove. The TROVE Architecture is shown in Figure 8.

Security issue

No security is provided to the data stored within the TROVE database. KEYSTONE module provides the access to the trove through the process of identity services. TROVE has no function to encrypt the data so as to secure the same and therefore it is left to the users to secure the same.

1.2.4 Horizon Dash board Service

Horizon provides three interfaces which are dashboards for the users to interact, for administrator to with system related configurations and all the users and administrator to carry application setting. Horizon is distributed with standard set of APIs that the developers can use for interacting with Open Stack components.

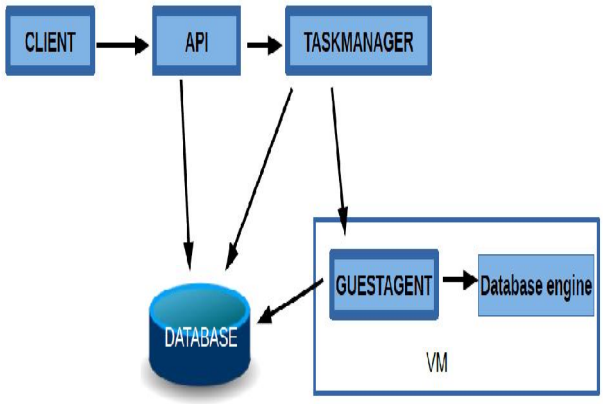

Figure 8: TROVE Architecture

Security issue

Horizon uses signed cookies for storing the session state on the client side. Get hold of the cookie, introspect and access all the details contained in the cookie. Use the information to access the cloud resources OPEN stack stores the authorisation token in the cookie which can be used to access the resources. The cookie can be accessed through use of file system

Some of the initiatives that can be taken to avoid cookie related vulnerability is to terminate the session when user logs out, then in which case the session cookies will also be terminated, Horizon to make a request to Identity service to invalidate the tokens and to implement server side session tracking instead of client side session tracking

1.2.5 Nova – The Computing Module

The actual computing is facilitated through Virtual machines which are handled by Hypervisors installed on physical servers. The management of theses servers is done by NOVA Module which provides major part of IaaS (Infrastructure as service).. This component is most complicated and distributed across several servers. Keystone provides authentication service and Glance provides Image services to NOVA. Users interact with NOVA for computing services using standard API. NOVA is built on several servers and the communication between distributed NOVA components is achieved through using message queues. NOVA uses SQL database for storing system related data. Figure 9 shows the architecture of NOVA component.

The Nova module is built using many components and sub-components. The NOVA API component is responsible for receiving HTTP requests, converting into commands and communicating with other components via the “Oslo

Messaging" queue orHTTP.The sub-component "nova-api" accepts and responds to end user compute APIcalls and the sub-component "nova-API-metadata" accepts metadata requests frominstances.

NOVA establishes communication between the VMs and the Hypervisor. Nova Compute service (NCS) is a daemon that creates and terminates Virtual machine instances. The component selects one of the Hypervisor which include Xen, KVM, or VMware before the VMs are created. The hypervisor is selected as per the user choice. The VMs are scheduled on different servers by the sub-component Nova Schedulers. The interaction between the NCS and Database service is handled by the sub-component nova conductor.

A sub-component Nova network (NW) manages IP forwarding, networking Bridges and also configure and manages Virtual LANS. This is daemon that accepts networking tasks, which are queued upon receiving requests from services and the users. This sub-component performs all the tasks required for setting up bridges, virtual LANS, sometimes making changes to IP tables and rules.

The sub-component "Console" allows end users to access their virtual instance's console through a proxy. It has three sub-components which include "nova-consoleauth daemon" which authorizes tokens for users that console proxies provide; "nova-novncproxydaemon" which provides a proxy for accessing running instances through a VNCconnection, "nova-cert daemon" provides x509certificates.

The sub-component "Image**"** manages interaction with Glance for image use. The sub-components to facilitate interaction with GLANCE module includes "nova-object store daemon" which is an S3 interface for registering images with the OpenStack ImageService and the sub-component "euca2ools client" is a set of command-line interpreter commands for managing cloudresources.

The sub- component "Database" is responsible for storing most build-time and run-time states for a cloud infrastructure such as available instance types, instances in use,available networks and projects.

Nova supports multiple different implementations for compute via plugins or drivers: NOVA can be configured to deal with Virtual servers, Containers and Bare Metal services**.** Virtualservers, in most cases, provide access to virtual servers from a single hypervisor; however, it is possible to have a Nova deployment include multiple different types of hypervisors. Containers allow using containers based on LXCorDocker in a similar way virtual machines are handled. A bare Metal server allows using physical machines in a similar way the virtual machines are requested by the user. Figure 9 shows the architecture of NOVA component

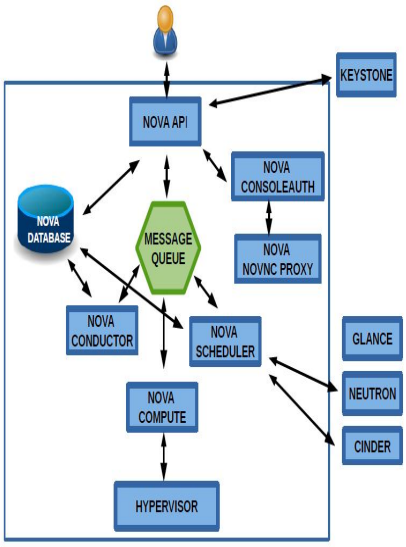

Figure 9: NOVA Architecture

Filterscheduler

The NOVA scheduler carries the filtering and weighing to decide on the server where the new instances must be created or can be created. The NOVA scheduler takes into account only the Physical servers on which COMPUTE component is installed. NOVA gets details about the physical servers from a dictionary and then filters the servers on which VMs can be created based on the filter properties set by the users.

Conductor

The conductor service component of NOVA manages the execution of workflows which involve the scheduler. This component is responsible for building, rebuilding, resizing the virtual machine and migrating the VM from one Physical server to the other. This Component provides separation of responsibilities between what compute nodes should handle and what the scheduler should handle, and to clean up the path of execution.

As soon as NOVA API receives a request from the user to build aninstance, API sends an RPC to conductor for building an instance. Conductor in turn sends an RPC to scheduler to select an existing VM and waits for a response. If a response is received, the VM stands selected. If no response is received from the scheduler, Conductor sends an RPC to "compute" component for building an instance and then return the address of the VM back to the conductor.
Security issue

No vulnerability exists when it comes to NOVA. The only issues are that the VM image is stored in the object storage through GLANCE Module. An attacker at best will be able to attack the image itself as it is stored as a resource file within object storage.

1.3 Other attacks on Open stack

In OpenStack user / application interface is provided through **RESTfull API. Using this interface a** Distributed Denial of Service attack (DDS) can be executed by sending to much traffic on a server. The servers however can handle only few numbers of requests in a go. A DDS attack results into a complete failure or slow down the services. DDOS is one of the most dangerous forms of attacks to the modern world as it cannot be traced easily. At each of the user Interface provisions are required for sensing the DDS attacks and take countering action such that entire system is not jeopardised.

The users generally interact through HORIZON module as it provides the dashboard required to interact with Open Stack system. Attackers can design their own web interface similar to HORIZON dash board and try to attack the open stack system by sending spurious requests and random initial user identifications

2. PROBLEM DEFINITION

The measures provided within open stack for providing authentication, access control and Data security are inadequate as they vulnerable and therefore the security system provided in the open stack needs to be extended.

3. RELATED WORK

[Qihong Shao 2011[[1] presented in his thesis different models required for implementing software as a service SaaS under Multi-Tenancy concept. He has presented several models to enable implementation of effective SaaS. The models presented by him include service request prioritisation model, Multi-layered customisation framework, a hybrid database management model that immensely support customisation and an ontology based resource access control O-RBAC (object based role based access control) system to secure the Multi-Tenant based model.

[RostyslavSlipetskyy 2011] [2] presented various aspects related object storage management by Open stack module "SWIFT", They have discussed the vulnerabilities of processing the objects stored in the database managed by SWIFT. They have focussed on the issues related to identify, access control and data security cellar bringing out the vulnerabilities and the counter measures that make the storage system secured.

When it comes to identity and access management, of object storage they have identified a security vulnerability relating to administrators with lower permissions could obtain credentials of administrators with higher permissions, they have also identified that administrators with higher credentials could read and delete the files and data of all the users. They have recommended that the user encrypt the data before uploading to the Object storage.

They have found vulnerability in storing the passwords in plain text and acceptance of weak passwords

The location of existence of a file is the confidentially maintained by OpenStack. Users are provided with the location of the files based on authentication token provided by identity services. A malicious user can claim damages to an existence file to open stack administrator who may commit a rollback.

[SamanZarandioon 2012] [3]has addressed the security challenges that one must meet when cloud computing services are to be offered to the customers.

A new protocol K2C which uses AB-HKU key updating scheme has been presented. The protocol is focussed at access control based on Cryptographic principles. A framework called OMOOS which can be implemented on client side helps seamless integration with cloud computing services and resources. A user centric identity management solution named Web2ID protocol was also presented that leverages client side cryptography and implements mechanisms that support Authentication, Exchange of attributes related user Identity and access delegation.

[Tanisha, 2013] [4] has proposed a new methodology that implements a security scheme using which the files uploaded to cloud can be secured. She has proposed a two stage encryption and decryption algorithm. No implementation on a live cloud computing system has been presented by this author.

[Jisha et al., 2013] [5] have presented a comparative analysis of different open source frameworks that include Nimbus, OpenStack, Eucalyptus, C-meter, Hadoop, and Open Nebula. They have not done any comparison with reference to security issues.

[Sasko Ristov et al., 2013] [6] have used scanners within OpenStack System for finding security threats and vulnerabilities using 4 different Operating systems. They have developed a methodology to determine the security vulnerabilities existing in compute and controller nodes of Openstack and using different configuration of virtual machines especially built on different operating systems. They have used Nessus 5 vulnerability and configuration assessment scanner using External Network Scan policy. Nessus scans all TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) ports, as well as the vulnerabilities of the services that work on certain opened port. Each vulnerability is rated as derived from the associated Common Vulnerability Scoring System (CVSS) score

[Ericson 2013] [7] Made a quick study on Identity module of open stack. They have compared the features supported within KEYSTONE with a conceptual model that states the requirements of an ideal Identity and access management system. They have defined what a generic management

system is. A generic management unit (IAM) which is primarily meant for access management that controls Authorisation and Authentication has been presented. IAM is designed for manging the accounts and services and propagating privileges and also for maintaining the user access policies..

In OpenStack, each service is responsible for authorization of any request authenticated by the Keystone server. Each service makes this decision based on the defined policy (defined rules in policy file) and context information (e.g., usernames, tenants, roles) it receives from a validated request. Although, it supports centralized role (role information is provided by Keystone service) based authorization, the current role is not fully compliant with RBAC model. Some of the missing features include

1. Separation of duty amongroles
2. Role hierarchy
3. Role to permission assignment.
4. Centralized policy management
5. Attribute based access control

The authors have identified different vulnerabilities existing in the keystone module of Openstack relating to authorisation, authentication and access control.

[Doudou Fall et al., 2014] [8] Presented that OpenStack has a logical architecture in which, the degree of interconnectedness within and between the components is a source of many security concerns. To prevent the damages that can be caused by the combination of these security issues, they have proposed a vulnerability tree security analysis of OpenStack's logical architecture that allowed us to generate ready-to-use vulnerability trees of the major services or components of the architecture. They have proposed an amendment of OpenStack's vulnerability naming, because the current naming does not cope well with our proposal.

[Girish L S et al., 2014] [9] Presented the way open stack can be implemented duly citing the operational procedures. They have detailed the architectures considered for building open stack.

[Sasko Ristov 2014] [10] Presented security assessment methodology including identification of assessment domain, assessment tools and test targets. They have focussed on determining security risk among the OpenStack cloud server nodes which are the physical servers of the cloud provider,

virtual machine instances that are provisioned on the physical machines, and OpenStack Dashboard. All security assessments are conducted from inside the cloud as inside vulnerabilities are reflection of the vulnerabilities that exists outside the cloud.

[Maria JorbaBrosa 2014] [11]Presented a thesis that documents the way the Open stack modules installed, configured, tested and trouble shouted and specifically the way VM instances can be created. The installation process to be followed for each of the module explained in detail.

[HalaAlbaroodi et al., 2014] [12] have presented different security issues that must be addressed in each deployment model which include SaaS, PaaS and IaaS. They have identified the following flaws within open stack system,

1. Users have no right to change their passwords, once set by the administrator
2. The administrator of open stack has access rights for all the projects and accounts. The administrator can tamper with the privileges of other users, change the projects and some time they have the privilege to delete the projects.
3. Clear text is used in in the API and also the URLS referring to the end points, making it easy to attack. Np support for SSL/TLS is provided making it possible for man-in-the-middle attacks and also sniffing the passwords on transit.
4. Any HOST that has the access to the DB and AMQP messaging system can operate on compute node for managing the VMs.
5. The user names and passwords are stored in clear text.

[MeryemeAyache et al., 2015[[13] have explained the security policies as defined by the cloud provider are used to effect access control of the cloud computing systems. There is no provision for high level user defined access control. Only fine grained access polices are implemented within open stack to execute a specific tack on a specific defined object. The authors have implemented middleware to provide high level security polices which are reduced into fine grained security polices implemented by swift.

[Ishan Gidwani et al., 2015] [14] have presented a security architecture built into cloud computing systems in general and then traced the security architecture built into OpenStack. They have presented all the known vulnerabilities in the OpenStack and the way they are exploited and can be mitigated. They have done the security assessment of OpenStack using nMaps scan and the have also presented the OpenStack can be used using tools that include Metasploit, THC Hydra, and Acunetix. They have provided some of the mitigation approaches to counter attack the OpenStack system.

[Doudou Fall, 2015] [15] Have proposed security risk quantification method that will allow the users and administrators to measure the level risk that one has to face when cloud computing systems are opted for use as in-house infrastructure. The author had used fault tree analysis approach. He has replaced the faults through probable vulnerabilities in a cloud system and they have used CVSS (common validation scoring system) to compute the risk formula. They have also provided an architecture which can be used to rank the cloud computing systems

Cloud computing systems are dynamic which means meets the demands of the user as and when they happen which means the security enforcement mechanisms must also be dynamic, but it is surprising the static security mechanisms are being used for enforcing security of the cloud computing systems. Doudou Fall has adapted risk-adaptive authorisation (RAdAM) model for simple cloud environment. He has used fuzzy inference system to prove the effectiveness of RAdAM. He has further extended the RAdAM to include vulnerability based authorisation mechanism VBAM which is a real-time authorisation model based on average vulnerabilities scores of the objects present in the cloud. They have applied this model to Open stack and implemented a new authorisation system.

[Yang Ou, 2015] [16] has described different security related issues that must be addressed by the cloud computing service provider. He has deliberated on the requirement of including privacy, data security and confidentiality, data audit, authentication and access control policy, security of virtual machine and automated management. He has brought of some insights into theses areas.

Cloud computing suffers due to lack standardisation, lack of customisation and privacy of the user data. Heavy demand exists for migrating traditional databases to the cloud meaning support of database as a service. Each database service requires that a specific standard is used for accessing the database where as some different kinds of standards are to be used for making a database as service to be offered within cloud. Therefore it becomes necessary for the cloud vendor adapt the database required standards with the service orientation standard of the cloud computing system.[Mehdi Bahrami et al., 2016] [17] Proposed a cloud computing based on service oriented architecture called DCCSOA (Distributed Cloud Computing based on service oriented architecture) that enhance the ability to adapt to different standardisations and customisation within the cloud. A single layer called Dynamic Template Service Layer (DTS) that interacts with all native cloud services and any cloud based services. This layer also provides standardisation for existing services and future services. The layer also provides customised services based on the requests of user Groups.

P. Ravi Kumar et al., [18] have presented a detailed survey on various threats, attacks that can be effected within a cloud computing system and also various mechanisms that can be used for counter attacking the attacks that can be made exploiting the Venerable areas of cloud computing. They have also detailed different stages of data processing and the related data security issues that crop up in a Multi-Tenant environment. They have identified 10 different services that can be offered by cloud computing where security is required which include applications, data, runtime, middleware, operating system, visualisation, server, storage and networking. The kind of security issues that crops up when cloud computing is used differ from service model to another service model. They have described clearly the kind of attacks that can happen in each layer of cloud computing and also suggested some mechanisms through which some of the attacks can be mitigated thoroughly.

[Tianwei Zhang 2017] [19] in his thesis, traced out different security issues tackled across the cloud computing system. He has also shown the way the security threats can be mitigated. He has presented new architectures and methods to detect and mitigate the security attacks that happen within a cloud computing system. He has classified all kind of attacks into 7 different types of attacking vectors which include services interface, network, cloud managers, virtualised system, share infrastructure, cloud services He has presented an architecture that helps monitoring the health of a Virtual machines. He also dealt three types of vulnerabilities that needs to be protected that include availability, confidently and integrity related vulnerabilities,

Bashir Mohammed et al., 2017] [20] have developed security infrastructure that can be built on top of OpenStack so that Vulnerabilities existing in OpenStack can be protected. The have named the security infrastructure called BradStack. They have explained the way BradStack can be deployed on OpenStack and the Penetration testing is carried to show the extent of security protection could be implemented within OpenStack.

[Marco Anisetti et al., 2017] [21] have proposed a benchmark which can be used to evaluate the extent to which security built into a cloud computing system. A security benchmark is a set of (standard) recommendations against which the security strength of different systems can be compared. The benchmark defines the parameters and the way the scores are computed. The scores are used to figure out the extent to which security is provisioned within cloud computing syste. The have experimented the benchmark using OpenStack. They have therefore defined a security benchmark for OpenStack as an instantiation and refinement of the generic CIS benchmark for IaaS systems on the basis of the OpenStack security guidelines.

[Bruce Benjamin et al. 2017] [22] Have considered three major security features in Open- Stack which include, key management, block storage encryption which provides confidentiality for data at rest and in transit and image integrity that ensure that VM images are not modified prior to boot. The authors have contributed to using Key Management Interoperability Protocol (KMIP) for storing keys, encrypting the block storage through which confidentiality of the data is maintained, and methods for maintaining the integrity of VM images.

[Darshan Tank et al., 2017] [23] Systematically have analysed the security aspects of the OpenStack keystone and explore the threat model against, and security requirements of, OpenStack keystone. They have proposed a new authentication model using the RESTful API to satisfy the security needs of OpenStack Keystone. The proposed

authentication model can accommodate a diverse set of security services.

[Kyle Hogan et al., 2018] [24] have presented RAFT the novel authorisation and Authentication technique of developing tokens which can be used for authorisation and authentication that helps that identity is not disclosed even when the services of OpenStack are corrupted.

[Oliver Schluga et al., 2018] [25] have verified the extent to which security issues have been addressed in cloud computing systems in comparison with ISO 27107 regulations. The have assessed the extent to which the security issues documented within Open stack documentation meet the ISO 27107 regulations.

[rhos-docs@redhat.com, 2019] [26] has published a guide on the good practice advice and conceptual information about hardening the security of a Red Hat OpenStack Platform environment. This document is applicable when Red Hat is used as an operating system used as the governing operating system installed on the HOST servers or made part of an image of a virtual machine. They have presented several methods to tie up the loose ends and hardening the security of open stack cloud computing system.

A number of Articles have been published by Sastry et al., [27][28][29][30][31][32][33][34][35][36][37][38][39][40] related to cloud computing systems, in particular Open Stack detailing various issues related to securing the Open Stack Cloud computing systems.

4. GAP

While many articles have been published in literature regarding securing Open Stack, none have proposed complete solutions especially considering the Multi-Factor Authentication, attribute based and User Privacy policy enforced access control, and enforcement of user data security. Open Stack considers three different type data storage which include VM image related database, Object based data storage, and Conventional Database oriented storage. The Module Keystone takes care of Authentication and Access control, SWIFT module takes care of Object storage, CINDER is related to Image storage and the Module Trove is related conventional database storage.This GAP is bridged to the extent of authentication, access control, and convention data storage implemented through TROVE Modules

5. INVESTIGATIONS AND FINDINGS

5.1 Authentication related Security Enhancements

The Major enhancement required is in the area of authentication. The most important implementation in these areas is Multi Factor Authentication through use of JACKSON tokens, ADS system Integration, and IMS system Integrations

Multifactor authentication through JSON Implementation

The way the multifactor authentication implemented using JSON server is shown in Figure 10.

JSON tokens are non-persistent, which are based on the JSON Web Token standard and implement the same as another component with the Open Stack. This backend will work the same way as fernet tokens works. The JSON token developed and signed using JWT(Java Web technologies) and JWS (java web services) standard, and the token will contain the authentication payload. Signed tokens are web safe and integrity verified, but the token payload is not opaque to its holder. It is possible to decode a token and inspect the payload with JWS tokens. The JSON Web Tokens are equivalent to Fernet tokens as they are encrypted and signed.

JWS tokens will be integrity verified with a private key and validated using a corresponding public key. Since the ES256 implementation only uses signing (as opposed to signed, encrypted payloads), this adheres to slightly better security practices over fernet because private keys never have to be synced across keystone API nodes. Only public keys need to be transferred to other keystone API servers to validate tokens across a cluster.Since JSON implementation is an independent application, the administrators of the open stack system will be able to change, modify, or remove items in the payload at any point in time and for any reason.

The token provider can undertake changes to the payload. The payload as such, is developed using the formats and structures decided by the token providers. The interpretation of the payloads based on parameters that are to be decoded by the users is risky as the users may miss-interpret the contents of the payloads. It is always non-risky if the formal API is used by users to request information from the Authentication service provider. The process will help to provide the payload information to the users which are not sensitive. Similar to the Fernet, JWTs will require a principal repository to set up to use for signing tokens. There is a need to add new command “keystone-manage” to handle generation and rotation of keys, implemented through the use of fernet commands “fernet_setup” and “fernet_rotate” commands. ES256, ES284, ES512 are the recommended algorithms to be used for signing the authentication message.

JWS tokens will be integrity verified with a private key and validated using a corresponding public key. Since the ES256 implementation only uses signing (as opposed to signed, encrypted payloads), this adheres to slightly better security practices over fernet because private keys never have to be synced across keystone API nodes. Only public keys need to be transferred to other keystone API servers to validate tokens across a cluster.The Authentication service (KEYSTONE) should not expose the algorithms used

internally to the end-user. End-user, as such, should not be allowed to request a specific JWS algorithm used for the creation of authentication Tokens. Only trusted algorithms used for token development.

The enhancement in this case is add a JSON component and Integrate with Ferret provide by making suitable changes to the Configuration file.

Multifactor authentication through Keystone integration with ADDS system

Another option available for implementing multifactor authentication is integrating the ADDS (Active Directory domain service) system with Keystone Module of Open Stack.

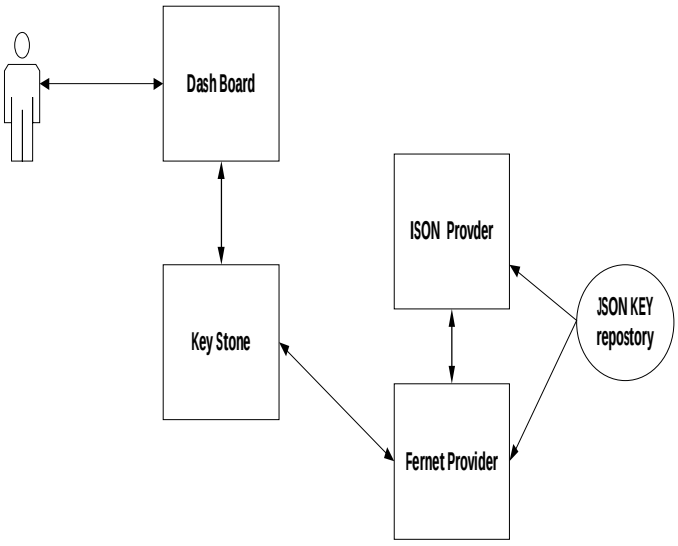

Figure 10: Enhancement of Open Stack authentication system through JSON tokens

The Vulnerabilities existing in the Keystone module investigated from different perspectives, especially the issue of tokenization and the use of multi-factor authentication. Several mechanisms can be introduced into the Open Stack so that the identity of the users can be made more secure. The measures added into Open Stack include the introduction of more secured Tokens, implementation of multi-factor authentication through federation approaches, etc. Every path leads to some complexity.

The more security built into the cloud computing system, the more will be the cost, especially in terms of loss of response time, which also requirement of sophisticated security models to be added into the system. The security models chosen must match the risk involved in providing a specific service required by the customers. The risk mitigation based security model is the most ideal.

Microsoft introduced Active Directory (AD) in which information about the domains and its related IP addresses, valid users and their passwords, details of the devices such as printers, disks, files, telephone numbers, etc. The directory queried using a standard API. The directory can be moved into different clients so that checks/decoding required carried in client location. The directory, as such moves over a network. AD is a shared infrastructure for managing and administering various network resources. Sometimes the directory is stored in a server, and the server queried for want of information especially the domain names and decoding of which to the IP addresses. AD considers every resource as an object and maintains different attributes of those objects.

A set of rules used to name the resources the details stored in AD. The user names used by AD are unique, and there will not be a name collision. The ADDS (Active Directory Domain system) implements the access control mechanisms so that only eligible users and the processes could access/Query the AD. ADDS is called the domain controller as it provides unique resource IDs given the name of the resources. ADDS system controls access to the information related to the resources - the details of which stored in AD. It implements the authentication and authorization system that regulates access by the users .to the resources contained in AD based on access policies. ADDS system provides that include directory services, federation services, certificate service, rights management services, and Lightweight Directory Services using LDAP (Lightweight directory access protocol). Users can access the services through a standard API supported ADDS.

ADDS server can be stand-alone or installed as a cluster. AD is distributed among several servers when clustered architecture used primarily when high availability is required. To implement a fault-free environment, ADDS performed as a Primary domain (PD), and a secondary domain (SD) with SD replicated using PD from time to time. In the case failure of PD, SD accessed until the time PD repaired and replicated again.

Each time a user makes a request for a resource, ADDS logs in the request, accesses a network resource, or runs an application, and the AD domain controller either authenticates the request are rejects the request if permissions for the resource access does not exist. Corruption in the ADDS database or the failure of the domain controller server can devastate an enterprise, so administrators often set up ADDS on a server cluster for automatic replication and synchronization for resiliency and added performance.

Smaller organizations can use Active Directory Lightweight Directory Services, which functions almost identically to ADDS but does not need domains or separate domain controllers.

Active Directory Certificate Services creates, validates, and revokes public key certificates used to encrypt files, emails, virtual private network traffic, and Transport Layer Security/IPsec network traffic. Active Directory Federation Services provides a single sign-on service to give users

access to resources or services -- typically outside of the enterprise -- using one set of credentials. Active Directory Rights Management Services controls encryption and access control for email, documents, and web content.

ADDS system used for authenticating and authorizing the users to have access to different resources contained in the computer system. Every user needs to be authenticated by the operating system before the user is allowed to have access to the application. The Application intern depended on the access provided by the operating system or enforces an additional security system built within the application. The app uses the ADDS services for verifying the access rights of the user. The access mechanism as supported by the ADDS systems, shown in Figure 11.

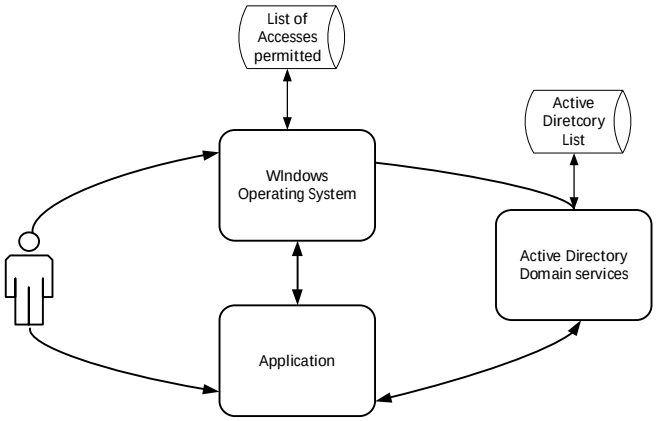

Figure 11: Authentication and authorizations ADS system

As explained earlier, Keystone uses fernet services for authenticating and authorizing to have access to different resources, and the bottlenecks of using such a tokenization system need consideration. Active services proved to be a versatile system of enforcing security, and ADDS system provides extensive services used to enforce security. Federation of ADDS system with fernet gives high-security provisions, and also existing users need not have any additional registrations with Open Stack. The way the ADDS system federated with Fernet shown in Figure 12.

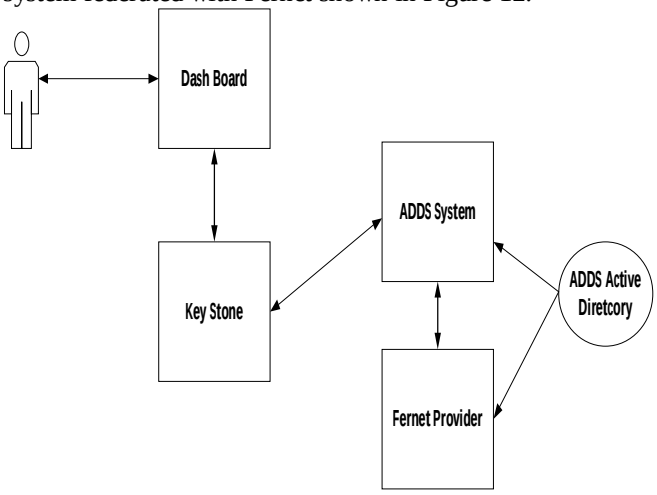

Figure 12 :ADDS federation with Fernet Tokenization system

It is necessary to complete prior tasks that include installation, configuration, and operationalization of ADDS, Open Stack, and DNS Systems and also that ADDS configured to use LADAP using port number 636, so that ADDS properly integrated with Keystone services All components of Open Stack that include NOVA, COMPUTE, KEYSTONE, HORIZON restarted for effecting the ADDS system for authentication purposes. Users account created in the ADDS system for making the users interact through DASHBOARD.

Multifactor authentication through Keystone integration with IMS system

The Vulnerabilities existing in the Keystone module are investigated from different perspectives, especially the issue of tokenization and the use of multi-factor authentication. Several mechanisms can be introduced into the OpenStack so that the identity of the users can be made more secure. The measures added into OpenStack include the introduction of more secured Tokens, implementation of multi-factor authentication through federation approaches, etc. Every path leads to some complexity. The more security built into the cloud computing system, the more will be the cost, especially in terms of loss of response time, which also requirement of sophisticated security models to be added into the system. The security models chosen must match the risk involved in providing a specific service required by the customers. The risk mitigation based security model is the most ideal.

RED HAT introduced the identity Management system (IMS) for assuring security. The IMS system used for authenticating and authorizing the users to have access to different resources contained in the computer system. Every user needs to be authenticated by the operating system before the user is allowed to have access to the application. The Application intern depended on the access provided by the operating system or enforces an additional security system built within the app. Every application running under the Red Hat operating system uses the IMS services for verifying the access rights of the user.

As explained earlier, Keystone uses Fernet services for authenticating and authorizing to have access to different resources, and the bottlenecks of using such a tokenization system need consideration. The IMS system provides several services through standard API for enforcing security based on international standards such as NIST. Federation of IMS system with Fernet gives high-security provisions, and also existing users need not have any additional registrations with OpenStack. The way the IMS system federated with Fernet shown in Figure 13.

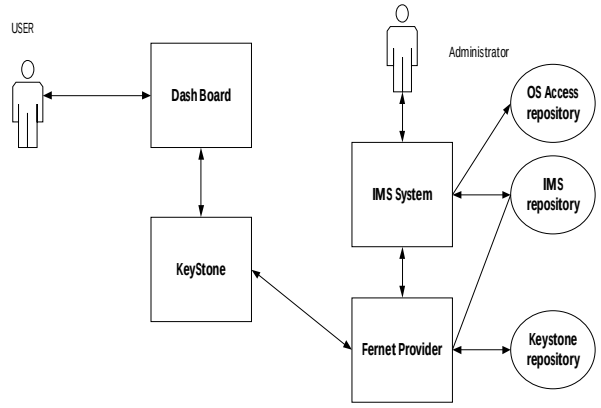

Figure 13 :Multifactor Authentication with IMS Integrated

It is necessary to complete prior tasks that include installation, configuration, and operationalization of IMS, Open Stack, and DNS Systems and also that IMS configured to communicate with Keystone using port number 636, so that IMS properly integrated with Keystone services All components of Open Stack that include NOVA, COMPUTE, KEYSTONE, HORIZON restarted for effecting the IMS system for authentication purposes. Users' accounts created in the IMS system for making the users interact through DASHBOARD. If a firewall installed to filter the communication between OpenStack and other services, then a configuration of the firewall is needed to allow the traffic between Keystone and IMS systems. An Open Stack controller node should be able to communicate with IMS and port number TCP636.

IMS (Identity Management System) is the Authentication system implemented by the Red Hat operating system which is known as the "Red Hat Identity Management" System and in short, it is referred to as IMS. IMS is used extensively for enforcing security when RED HAT used as an operating system. IMS proved to be a system that provides extensive protection. The keystone identity system needs strengthening so that the open-source software used for the development of public and private cloud to the utmost satisfaction of the clients. Multi-factor authentication by integrating the IMS with keystone gives a higher level of security

Identity Service (IMS) authenticates certain Red Hat Identity Management (IdM) users while retaining authorization settings and critical service accounts in the Identity Service database. As a result, Identity Service has read-only access to IdM for user account authentication, while retaining management over the privileges assigned to authenticated accounts. Before configuring and integrating IMS with OpenStack, the modules that include Red Hat Identity Management, Red Hat OpenStack Platform, and DNS name resolution installed, configured, and made operational. These steps allow IdM users to authenticate to OpenStack and access resources. OpenStack service accounts (such as keystone and glance), and authorization management (permissions and roles) will remain in the Identity Service database. Permissions and roles are assigned to the IdM accounts using Identity Service management tools.

For the IDM to work appropriately with Keystone, the Keystone for adding the IdM backend and Compute services on all nodes needs restarting for switching over to Keystone V3. Users will be unable to access the dashboard until their accounts created in IdM.

5.2 Authorization Related to Security Enhancements

While the Auutheication is implemented for providing the identity service, Access control is implemented through Authorizations Systems. The Identity service of open stack supports the notion of groups and roles. Users belong to groups while a group has a list of roles. OpenStack services reference the roles of the user attempting to access the service.The OpenStack policy enforcer middleware takes into consideration the policy rule associated with each resource then the user'sgroup/roles and association to determine if access is allowed to the requested resource.

Establish Formal Access Control Policies

Prior to configuring roles, groups, and users, one should document required access control policies for the OpenStack installation. The policies should be consistent with any regulatory or legal requirements for the organization. Future modifications to the access control configuration should be done consistently with the formal policies. The policies should include the conditions and processes for creating, deleting, disabling, and enabling accounts, and for assigning privileges to the accounts. Periodically review the policies and ensure that the configuration is in compliance with approved policies. Open Stack provides access control only for fine grained access to the services. They do not support user defined polices especially for accessing the data

Each OpenStack service defines the access policies for its resources in an associated policy file. A resource, for example, could be APIaccess,the ability to attach to a volume, or to fire up instances.The default policy rules can be modified by creating a JSON format file called policy.json.

For example, for the Computeservice, create a file called policy.json in the nova directory.Note that the exact file path might vary for containerized services.. These policies can be modified or updated to control access to various resources. Ensure that any changes to the access control policies do not unintentionally weaken the security of any resource.

Open stack offers Access control based on the roles assigned to the users. The access control is enforced through allocation of a system defined roles to the users. Users have their own functional roles which can be

defined based on the attributes of a user. The access to the users must be provided based on their functional roles. Thus there is a requirement of converting the functional roles to the fine grained user roles as defined by the Open stack and also user defined access policies must also be implemented which are to be converted to system defined access polices.

Figure 14 shows the extension of the access control system implemented by Open Stack, After the user is authenticated by keystone, the user has the identity token which is exchanged with all the servers in the Open stack. The user then sends the token to a Middleware server where the user attributes are found and the same are converted to the system defined User Access. Using User attributes user defined polices are accessed which are then converted to System defined Polices for accessing a specific service, Both the System define Access right and the System policies are fed to the services server which uses rights and polices to decide dynamically the extent to which the service can be give at fine Grained Level.

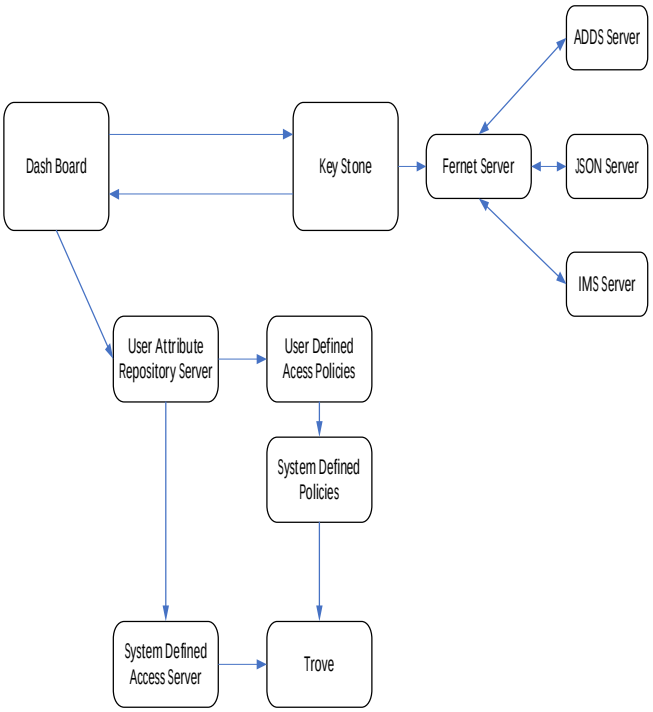

Figure 14: Enhancement of Access control system implemented in Open Stack

5.3 Data Related Security Enhancements

Open Stack supports three types of storage to access Images, Objects and Conventional Databased. The kid of security built within each service to provide the access to the data resources varies. In this section an enhancement to the security provision made within Trove Module is proposed.Any user who wants to store the data as a relational database can use this service; Trove provides Database as a Service for OpenStack. TROVE designed to run entirely on OpenStack allowing users to quickly and efficiently utilize the features of a relational database without the burden of handling complex administrative tasks. Cloud users and database administrators can provision and manage multiple database instances as needed.

Initially, the service will focus on providing resource isolation at high performance while automating complex administrative tasks, including deployment, configuration, patching, backups, restores, and monitoring.

The module TROVE built using four components includes "Trove-API" that provides the interface to the VM based applications through RESTful API that supports JSON and XML to provision andmanage Trove instances. The component "Trove-Task Manager" helps in providing database instances, managing the database instances, and also carries the required database operation. The SQL server used as a database engine in the backend. The component "Trove-guest agent" as service runs within the guest instance, responsible for managing and performing operations within the Database itself. The Guest agent listens for RPC messages through the message bus and completes the requested transaction. 'Trove-conductor" is a service that runs on the host, responsible for receiving messages from guest instances to update information on thehost. The Architecture Trove is shown in Figure 15

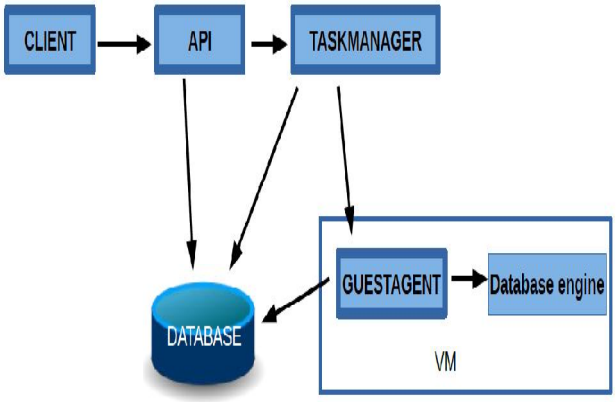

Figure 15: Trove Architecture

TROVE is not designed with built-in security features to protect the data.KEYSTONE module provides access to the trove through the process of identity services. Most cloud providers do not encrypt data before saving it to storage. OpenStack does not provide any data encryption at all; thus, users would need to encrypt their data before uploading it and manage their encryption keys themselves. The way the Users can access the database is shown in figure16.

To the standard OpenStack Architecture, an additional component added for providing the data security to protect the data when several applications placed in a VM, and several users are given access to the applications. Addition of

an additional component "Data Security" needed for achieving data Isolation to be effected among the users accessing several applications running on the same Virtual Machine. Figure 17shows theExtended Architecture. The data security components implement three different sub-components, each responsible for performing data security using one of the approaches that include Multi-Instance, Multi-User Spaces, or Multiple database segments. The sub-components interact with TROVE for carrying database operations through calling Restful-API.

Security under the implementation of Multi-Tenancy The way data security is implemented when Multiple Database segments, Multiple User spaces and Multi instances are used could designed developed as per user requirements

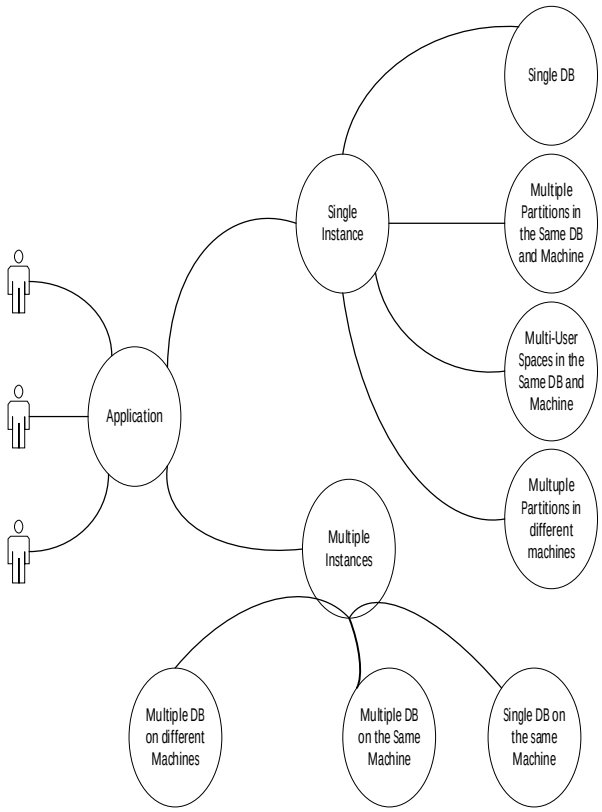

Figure 16: Data organization mechanisms under cloud

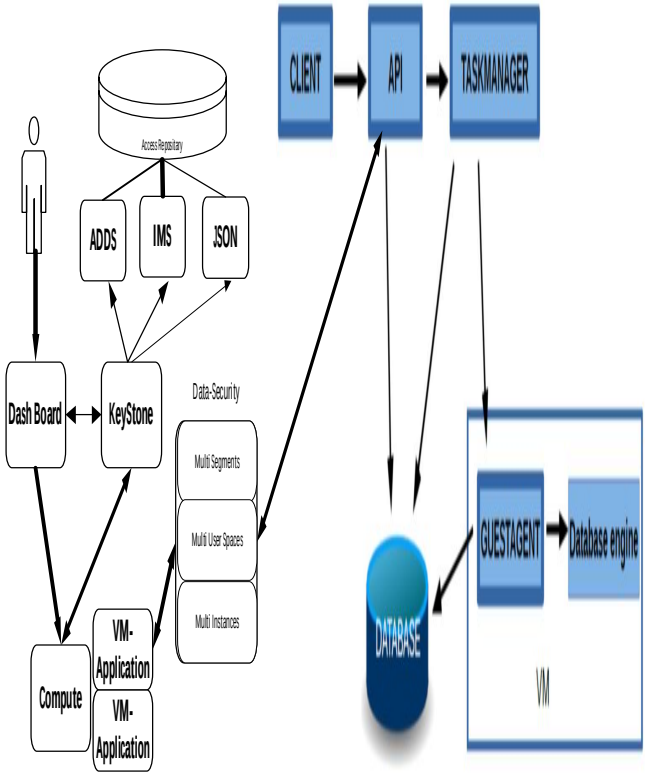

Figure 17: Extending OpenStack for implementing Data

6. CONCLUSION

Open Stack is not fully secured as many vulnerabilities exists in many of the process that include Authentication, access control and data security especially when data have to be stored and managed in the conventional database using the TROVE Module

Multi factor authentication provides full proff tokens when JSON tokens are used or by integrating ADDS or IMS system

For proper access control to be implemented, Attribute based and user defined policy based Access control is to be implemented which needs proper integration with the Open Stack defines user access systems.

When it comes to storage using conventional databases, TROVE offers least protection to the user data and therefore required additional security is to be Built over and above TROVE. Users have to build a security system considering the uses of databases in terms of multi instances, Multi user spaces and Multi Partitions.

REFERENCES

1. Qihong Shao, Towards Effective and Intelligent Multi-tenancy SaaS, Thesis submitted to Arizona State University, 2011
2. RostyslavSlipetskyy, Security issues in Open Stack, Thesis submitted to University of Norwegian University of Science and Technology, 2011
3. SamanZarandioon, Improving the security and usability of Cloud services with user-centric Security models, Dissertations submitted to State University of New Jersey, 2012
4. Tanisha, Ensuring File Security on cloud using two tire encryption and decryption, thesis submitted to school of mathematics and computer science, Thaper University, 2013
5. Jisha S. Manjaly, Jisha S., A comparative study on open source cloud computing frameworks, International Journal Of Engineering And Computer Science, Volume 2, Issue 6 June, 2013 Page No. 2026-2029
6. SaskoRistov, MarjanGusev and AleksandarDonevski, OpenStack Cloud Security Vulnerabilities from Inside and Outside, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, 2013
7. https://www.ericsson.com/open stack keystone analysis.pdf
8. Doudou Fall, Takeshi Okuda, YoukiKadobayashi, and Suguru Yamaguchi, Towards a Vulnerability Tree Security Evaluation of OpenStack's LogicalArchitecture, T. Holz and S. Ioannidis (Eds.): TRUST 2014, LNCS 8564, pp. 127–142, 2014.
9. Girish L S, H. S. Guru Prasad, Building Private Cloud using OpenStack, International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), Volume 3, Issue 3, May – June 2014
10. SaskoRistov, MarjanGusev, and AleksandarDonevski, Security Vulnerability Assessment of OpenStack Cloud, Sixth International Conference on Computational Intelligence, communication Systems and Networks, 2014
11. Maria JorbaBrosa, Study and Development of an OpenStack solution, Thesis submitted to University of at Politècnica de Catalunya, 2014
12. HalaAlbaroodi, SelvakumarManickam and Parminder Singh, Critical review of openstack security: issues and weaknesses, Journal of Computer Science 10 (1): 23-33, 2014
13. MeryemeAyache, Mohammed Erradi and Bernd Freisleben, Access Control Policies Enforcement in a Cloud Environment: Openstack, 11th International Conference on Information Assurance and Security (IAS), 2015
14. Ishan Gidwani, Mane, Security issues in openstack, International Journal of Computer Science and Information Technology Research, **Vol. 3, Issue 2, pp: (1147-1158), 2015**
15. Doudou Fall, Security quantification and Risk-Adaptive authorisation mechanism in cloud computing, Thesis submitted to Nara Institute of Science and technology, Japan, 2015
16. Yang Ou, The concept of cloud computing and the main security issues in it, Thesis submitted to Turku University of Applied Sciences, 2015
17. Mehdi Bahrami, A Dynamic Cloud with Data Privacy Preservation, Thesis submitted to University of California, Merced, 2016
18. P. Ravi Kumar, P. Herbert Raj, P. Jelciana' Exploring Security Issues and Solutions in Cloud Computing Services – A Survey, Cybernetics and information technologies volume **17**, No 4, 2017
19. Tianwei Zhang, Detection and Mitigation of Security Threats in Cloud Computing, thesis submitted to Princeton University
20. Bashir Mohammed, SibusisoMoyo, K. M Maiyama, SulaymanKinteh, Al NoamanM.K. Al-Shaidy, M. A. Kamala and M. Kiran, Technical Report on Deploying a highly secured openStack Cloud Infrastructure using BradStack as a Case Study.Cloud Computing Modelling and Simulation Research Group School of Electrical Engineering and Computer Science University of Bradford.UK
21. Marco Anisetti, Claudio A. Ardagna, Filippo Gaudenzi, Ernesto Damiani, A Security Benchmark for OpenStack, IEEE 10th International Conference on Cloud Computing, 2017
22. Bruce Benjamin, Joel Coffman, HadiEsiely-Barrera, Kaitlin Farr, Dane Fichter, Daniel Genin, Laura Glendenning, Peter Hamilton, Shaku Harshavardhana, Rosalind Hom, Brianna Poulos, Nathan Reller, Data Protection in OpenStack, IEEE 10th International Conference on Cloud Computing, 2017
23. Darshan Tank, Akshai Aggarwal, and NirbhayChaubey, Security Analysis of OpenStack Keystone, International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS) Volume VI, Issue VI, June 2017 | ISSN 2278-2540
24. Kyle Hogan, HodaMaleki, Reza Rahaeimehr, Ran Canetti, Martenvan Dijk, On the Universally Composable Security of OpenStack, National Science Foundation as part of the MACS Frontier project,
25. Oliver Schluga, Elisabeth Bauer, Ani Bicaku, SiliaMaksuti, Operations Security Evaluation of IaaS-Cloud Backend for Industry 4.0, Proceedings of the 8th International Conference on Cloud Computing and Services Science, *pages 392-399, 2018*
26. rhos-docs@redhat.com, Red Hat OpenStack Platform 12 Security and Hardening Guide- Good Practices, Compliance, and Security Hardening, 2019, (Information Science), 2015 (11): 700-706.
27. JKRSastry, M TrinathBasu, Securing SAAS service under cloud computing-based multi-tenancy systems, Indonesian Journal of Electrical Engineering and Computer Science, Volume 13, Issue 1, Page 65-71, 2019
28. JKRSastry, M TrinathBasu, Securing Multi-tenancy systems through multi DB instances and multiple databases on different physical servers, International

Journal of Electrical and Computer Engineering (IJECE), Volume 9, Issue 2, Pages 1385-1392, 2019
29. M. TrinathBasu, Dr.JKRSastry, A full security included Cloud Computing architecture, International Journal of Engineering & Technology, Volume 7, Issue 2.7, Page 807-812, 2018
30. JKRSastry, M TrinathBasu, Securing Multi-tenancy systems through user spaces defined within the database level, Jour of Adv Research in Dynamical & Control Systems, Volume 10, issue 7, Page 405-412, 2018
31. J. K. R. Sastry, K. Sai Abhigna, R. Samuel and D. B.K. Kamesh, Architectural models for fault tolerance within clouds at the infrastructure level, ARPN Journal of Engineering and Applied Sciences, VOL. 12, NO. 11, 2017, Pages 3463-3469,
32. DBK Kamesh, JKRSastry, Ch. Devi Anusha, P. Padmini, G. Siva Anjaneyulu, Building Fault Tolerance within Clouds at Network Level, International Journal of Electrical and Computer Engineering (IJECE), Vol. 6, No. 4, pp. 1560~1569, 2016
33. S. L. SUSHMITHA, Dr. D. B. K. J.K. R. SASTRY, V. V. N. SRI RAVALI, Y.SAI KRISHNA REDDY, building fault tolerance within clouds for providing uninterrupted software as service, Journal of Theoretical and Applied Information Technology, Vol.88. No.1, Pages 65-76, 2016
34. NVSPavan Kumar, Dr.JKRSastry, Dr. K Raja Sekhara Rao, Mining Distributed Databases for Negative Associations from Regular and Frequent Patterns, International Journal of Advanced Trends, Volume 8, Issue 4, Pages 1440-1463, 2019
35. NVSPavan Kumar, Dr.JKRSastry, Dr. K Raja Sekhara Rao, On Incremental mining Databases for Regular and Frequent Patterns, International Journal of Emerging Trends and engineering research, Volume 7, Issue 9, Pages 291-305, 2019 https://doi.org/10.30534/ijeter/2019/12792019
36. NVSPavan Kumar, Dr.JKRSastry, Dr. K Raja Sekhara Rao, Mining Negative Frequent regular Itemsets from Data Streams, International Journal of Emerging Trends and engineering research, Volume 7, Issue 8, Pages 85-98, 2019 https://doi.org/10.30534/ijeter/2019/02782019
37. M. TrinathBasu, JKRSastry, Improving the Open Stack Authentication system through federation with JASON Tokens, International Journal of Advanced Trends n Computer Science and Engineering, 3596-3614,2019. https://doi.org/10.30534/ijatcse/2019/143862019
38. JKRSastry, M TrinathBasu, Multi-Factor Authentication through Integration with IMS System, International Journal of Emerging Trends in Engineering Research, Volume 8, Issue 1, 2020, PP. 87-113
39. JKRSastry, M TrinathBasu, Strengthening Authentication within OpenStack Cloud Computing System through Federation with ADDS System International Journal of Emerging Trends in Engineering Research, Volume 8, Issue 1, 2020, PP. 213-238
40. JKRSastry, M TrinathBasu, Enhancing Data Security under Multi-Tenancy within Open Stac, International Journal of Advanced Trends in Computer Science and Engineering, Volume 8, Issue 1, 2020, PP. 533-544

Table 1: Major Modules of the Open Stack - Functionality

Serial Number	Name of the Module	Description of the Module
1.	**NOVA**	Nova is the primary computing engine behind OpenStack. It is a "fabric controller," which is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks.
2.	**SWIFT**	Swift is a storage system for **objects and files**. Rather than the traditional idea of a referring to files by their location on a disk drive, developers can instead refer to a unique identifier referring to the file or piece of information and let OpenStack decide where to store this information. This makes scaling easy, as developers don't have the worry about the capacity on a single system behind the software. It also allows the system, rather than the developer, to worry about how best to make sure that data is backed up in case of the failure of a machine or network connection
3.	**CINDER**	Cinder is a block storage component, which is more analogous to the traditional notion of a computer being able to access specific locations on a disk drive. This is more traditional way of accessing files might be important in scenarios in which data access speed is the most important consideration.
4.	**NEUTRON**	Neutron provides the networking capability for OpenStack. It helps to ensure that each of the components of an OpenStack deployment can communicate with one another quickly and efficiently.
5.	**HARIZON**	Horizon is the dashboard behind OpenStack. It is the only graphical interface to OpenStack, so for users wanting to give OpenStack a try, this may be the first component they actually see. Developers can access all of the components of OpenStack individually through an application programming interface (API), but the dashboard provides system administrators a look at what is going on in the cloud, and to manage it as needed
6.	**KEYSTONE**	Keystone provides identity services for OpenStack. It is essentially a central list of all of the users of the OpenStack cloud, mapped against all of the services provided by the cloud which they have permission to use. It provides multiple means of access, meaning developers can easily map their existing user access methods against Keystone
7.	**GLANCE**	Glance provides image services to OpenStack. In this case, images refers to images (or virtual copies) of hard disks. Glance allows these images to be used as templates when deploying new virtual machine instances.
8.	**CEILOMETER**	Ceilometer provides telemetry services, which allow the cloud to provide billing services to individual users of the cloud. It also keeps a verifiable count of each user's system usage of each of the various components of an OpenStack cloud.
9.	**HEAT**	Heat is the orchestration component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application. In this way, it helps to manage the infrastructure needed for a cloud service to run.