ISSN 2278-3091
Volume 8, No.6, November – December 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse143862019.pdf
https://doi.org/10.30534/ijatcse/2019/143862019

Improving the Open Stack Authentication system through federation with JASON Tokens

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

ABSTRACT

Many cloud computing systems are in place for providing different kinds of services, which include AWS, AZURE, Google Cloud, etc. which are proprietary. These clouds help the users implement their own IT requirements, but the users cannot configure or customize the cloud computing processes as per their needs, especially to handle the issue of security from the perspectives of authentication, access control, and data security.

Open source cloud computing systems, which include Eucalyptus, Open Nebula, open stack, etc. allows the changes carried to the cloud computing systems primarily through configuration, the addition of API, the addition of processes, etc. Users can make changes to affect the system such that it works as per the user requirements, especially to improve the security system built into the cloud computing system, which sometimes found to be vulnerable for attack. Users are concerned about the security of their software data hosted on third-party IT infrastructure. Open Stack cloud computing platform is being used by many for implementing private clouds. Users can customize open stack as per their requirements.

Open stack suffers from many security-related vulnerabilities that can be exploited by the users for attacking the user software and data. A review of the Open stack systems is required to find the gaps that are existing to plug the same.
In this paper, a review of the Open stack presented, bringing out different kinds of vulnerabilities that exist in authenticating the users and a federation method using JSON tokens showed that help eliminating the Vulnerabilities existing in the open stack for enforcing security within the Keystone module of Open stack.

Key words: Open Source cloud computing, Open Stack, JSON Tokens, Federated authentication, Security enforcement within open Stack

1. INTRODUCTION

Users register with a cloud computing system using the username name and password. The registration process sometimes involves the operation of a contract, which includes pricing, services required, response time, throughput, penalties, downtime provisions, limitations on the levels of assistance needed, etc.

Users start communicating with the identity service of a specific cloud computing system by keying in the user name and password. The identity service after validating and verifying the user will send a token, which is a formatted data string containing the details of the services availed, kind of accounts used, etc. The user then uses the authentication token to start directly communicating with the services intended by the user. The general process used for authentication shown in Figure 1. The service component verifies the authenticity of the user after receiving the request from the user, and on getting confirmation, the service component authorizes the user to access the service.

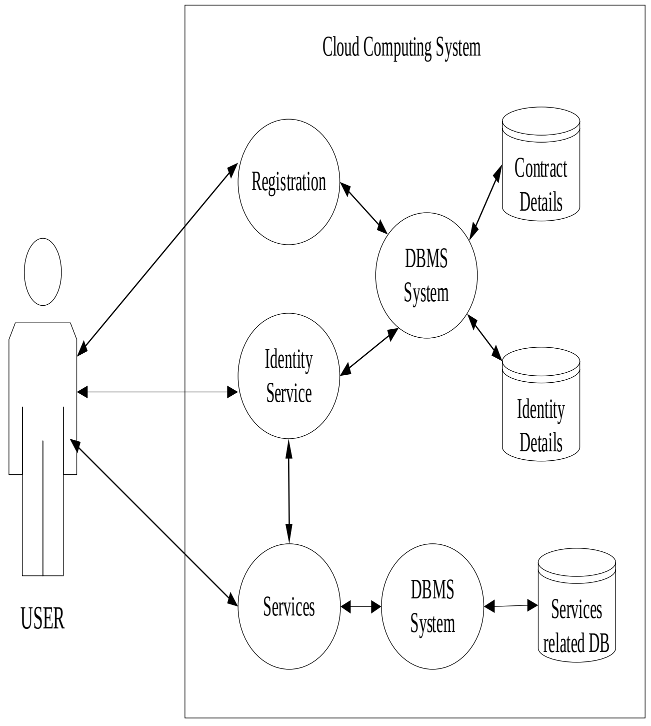

Figure 1: General Process Flow for authentication System

1.1 Multi-factor Authentication

Multi-factor authentication means implementing the process of authentication by using multiple mechanisms. For example, an authentication system in addition to ensuring the implementing authentication within itself but also further check the credentials of the users by checking with other kinds of authentication systems such as LDAP, Active Directory, Red Hat IMS, FreeIPA, etc.

The multi-factor authentication helps to mitigate various kinds of attacks that include brute force, social engineering, spear, and mass phishing attacks, which generally attack the user names and passwords. There is a need to deploy third-party tools and integrate the same with the underlying authentication system built to recognize the users.

Administrative user accounts should authenticate using both native authentication systems and an external authentication service that supports two or more factors for authentication, such as a digital certificate, digital signature, etc. An external authentication service can include Red Hat Identity Management, or Microsoft Active Directory, or any other third party defined authentication system. This approach can help reduce the risk of passwords that might be compromised.

Figure 2 shows the way the multi-factor authentication system works. The identity service usually designed to work with backbends used as plugins, which may use a further extension to the process of authentication. The Identify service configured to use one or more plugins so that the authentication system implemented using multiple factors. The process called a federated authentication system

The token often passed as a specific structure containing the details of the services, resources that can be accessed, etc. The authentication token also provides a catalog of various services that a cloud computing system can offer. Each service listed with its name, access endpoints for internal, admin, and public access.

The token, once distributed, can be revoked by the system that made available the Token. Users can use API of the Identity service to revoke the tokens, get the list of revoked tokens, get the list of various services offered by the cloud computing system to the user who has access to the token, to remove the existing token — all queries related to the tokens initiated by the users or the services supported through API calls. The identify service provides API, which can be used for token management through operations such as token revocation, to list existing tokens, remove tokens, cache tokens, etc.

There are many types of taken management systems used in the literature, which include **UUID, Fernet, PKI, PKIZ, JSON, etc.** which differ from each other in many ways in terms of the content of those tokens and the way content in the token is secured. The token is the most venerable elements within the cloud computing system.

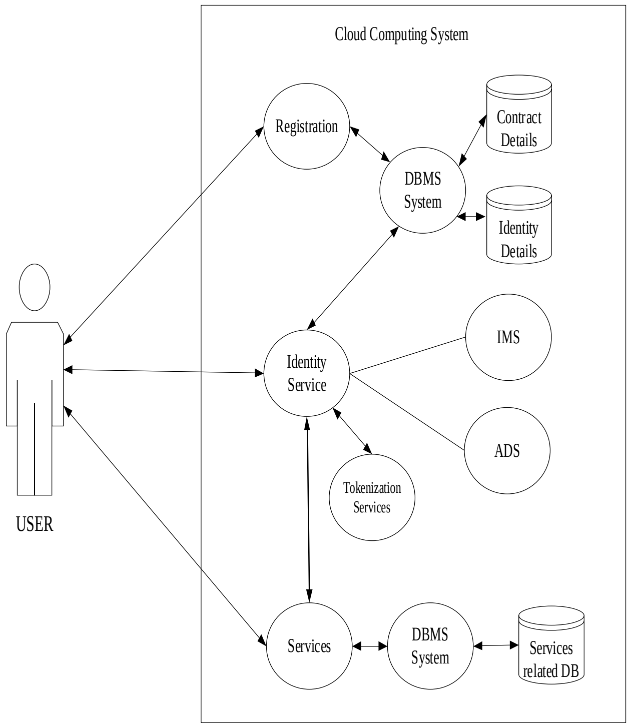

Figure 2: Process Flow for Multi-Factor Authentication

1.2 Use of token for effecting authentication

Once a user authenticated, a token generated for authorization to access cloud computing service. A token can have a variable life span defined by enforcing the system. The recommended expiry value set to a lower value that allows enough time for internal services to complete tasks. If the token expires before tasks are completed, the cloud might become unresponsive or stop providing services. An example of expended time during use would be the time needed by service to transfer a disk image onto the hypervisor for local caching.

JSON Web Token is a type of non-persistent bearer token similar to the fernet tokens. JWT is an open-standard token that can be maintained using JWT API.

Implementation of JSON Token within a cloud computing system helps in the fully integrated authentication system.

JWT token is a new type of token that is backed by a widely used standard. JWT tokens increase the chances of interoperability between the OpenStack ecosystem and other communities that support JWT.

JWT token designed using open standards recommended by NIST (National Institute of Standards and Technology)

2. AUTHENTICATION SYSTEM AS IMPLEMENTED IN OpenStack

OpenStack is an open-source Infrastructure as a Service (IaaS) cloud computing software, where users can provision virtual machines by using its components such as storage (called “swift”), compute (called “nova”), etc. Figure 3 shows a high-level overview of OpenStack.

OpenStack deployed in standard hardware and its resources like computation, networking, and storage shared in the cloud. These resources controlled using an OpenStack dashboard. Users can avail of these resources by using a client program such as an Internet browser. OpenStack has a modular architecture.

Open Stack is composed of a set of Modules that together deliver the functionality required by the user. Many modules are available, and each module provides a kind of service needed by the user. The functioning of the OPEN stack module individually attacked. To understand the extent to which an open stack is secured, each Module assessed to find the vulnerabilities and the level of security built into each of the Modules. The weaknesses of the OPEN STACK component that can be exploited by attackers must be known to make the Modules secured from attacking through the implementation of counter-attacking mechanisms.

The Security enforcement under open Stack recognized in terms of authorization, authentication access control, and data security.

The authorization and authentication service and the access control archived through KEYSTONE Module. KEYSTONE Module handles all the issues related to the identification of the users.
Virtual Images managed through GLANCE Module, but the security of virtual images handled through SWIFT, which stores all the VM images and security of which is dealt with by it.

Data in OpenStack managed through three distinct modules that include SWIFT (Object Storage), CINDER (Block Storage), Trove (Relational Databases), Regular applications use a database, and therefore, the use of Trove done extensively. Securing when TROVE used is the Major Issue. A certain level of enforcement of security done within these modules

One of the most important issues connected with the security is identifying the users, group of users, and their roles and privileges that the users have in availing the resources. The Module KEYSTONE provides the Identity services to all the other modules in OpenStack. The security issue is very much related to identifying the users and the rights that are granted to those users, such that the users provided with access to the resources for which permissions granted.

Keystone service provides a standard authentication and authorization store for OpenStack services. Keystone is responsible for users, their roles, and to which tenants may belong. Moreover, it provides a catalog of OpenStack services that can be accessed by a user upon request and valid authentication. Essentially, Keystone is responsible for carrying primary functions to control the authentication and authorization of a user:

Keystone Carries user Management; It keeps track of users and what they are allowed to do. It carries the check through verification of the associations between users, roles, and tenants. Keystone provides a catalog of available services and where their API endpoints located.

In OpenStack, the Users are digital representations of a person, system, or service that require the services rendered by the system. Keystone ensures that incoming requests coming from a valid user assigned to a particular tenant with a specific role assigned to resource-access tokens.

Entire user identification and access deigned over certain elements that include user, tenant, role, token, and endpoints. A tenant is a group used to isolate resources that contain a set of users, Customers, and internal processes. A role includes a set of assigned user rights and privileges for performing a specific set of operations. **A user token issued by Keystone contains a list of that user’s roles**. Services then determine how to interpret those roles by their internal policies stored in each **policy.json** file. Credentials are data known only to a specific user who proves his or her identity.

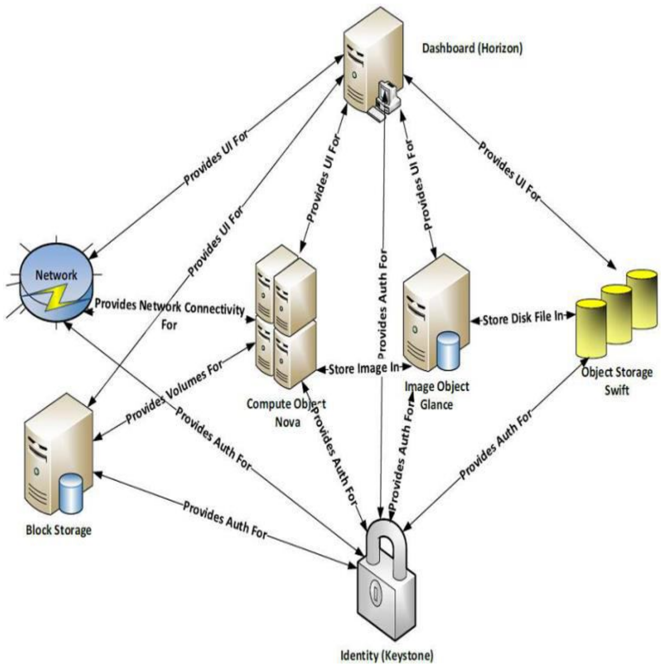

Figure 3 : Overall Interactions within Key Modules

A token is an arbitrary bit of text used to access resources. Each token has a scope describing accessible resources that may be revoked at any time and is valid for a finite duration. An endpoint is a network-accessible address, usually represented by URL, from which services accessed.

The interactions that happen within the OpenStack for accessing the resources shown **in** Figure 3. Keystone provides authentication, authorization and access control of various resources provided to different users

The following steps followed when a user tries to access an OPEN STACK service.

1 The user who is trying to act as a service sends its credentials to Keystone

2 Keystone sends the user a temporary token and a list of tenants if the authentication of the previous credentials succeeds. Here Tenants imply a group that can either refer to a set of users or resources. There can be many tenant groups configured by the administrators. Every user placed in the group. A User Group has access to a group of services.

3 The user sends the credentials to Keystone together with the desired tenant

4 If the authentication of the credentials and the desired tenant is correct, Keystone sends a tenant token and a list of available services for this tenant.

5 The user determines the correct endpoint depending on the action performed and requests to the endpoint along with the tenant token acquired for authentication. An endpoint is a location where the user expects that the service rendered. An endpoint is a network-accessible address, usually described by URL, from which services accessed.

6 Keystone verifies whether the received token is correct and is allowed to use the service to avoid connections from non-authorized entities.

7 If the keystone authenticates the token and the right to use the service, it checks to find if the access policy attached to the facility conforms to the kind of access requested by the user.

8 If the policy validation is correct, the request executed.

The architecture of the KEYSTONE module shown in Figure 4.

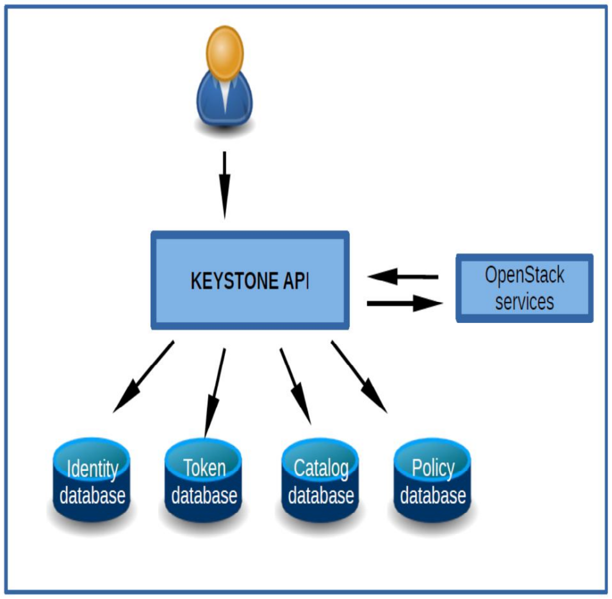

Figure 4: Keystone architecture

Within keystone architecture, four-component services include token service, Catalogue service, and Policy service, and database service included**.** The identity service provides authentication service that provides validation of credentials of the users, and the validation of the users concerning roles and Accounts and also validation regarding metadata data. The token service validates and manages Tokens used for authenticating requests once a user/tenant's credentials have already verified. The Catalogue service provides an endpoint registry used for endpoint discovery. The policy service provides a rule-based authorization engine.

The "Keystone" component of OpenStack provides identity service for authenticating and high-level authorization. A token-based and service-based authorization system implemented by the Keystone component of OpenStack. Keystone is the centralized identity and access management component of OpenStack. The keystone module uses a pluggable data store (SQL, LDAP).

The Identity service can store user credentials in an SQL Database or may use an LDAP-compliant directory server. The Identity database may be separate from databases used by other OpenStack services to reduce the risk of a compromise of the stored credentials.

Organizations may desire to implement external authentication for compatibility with existing authentication services or to enforce stronger authentication policy requirements, although passwords are the most common form of authentication, compromised through numerous methods, including keystroke logging and password compromise. External authentication services can provide alternative ways of authentication that minimize the risk from weak passwords.

These include an internal and an external authentication system. The internal authentication systems require user passwords to conform to minimum standards for length, diversity of characters, expiration, or failed login attempts. In an external authentication scenario, this would be the password policy on the original identity store.

The authentication service requires the user to provide information based on something they have, such as a one-time password token or X.509 certificate, and something they know, such as a password.

A user can manage OpenStack session using its dashboard (Horizon), which can be accessed using a web browser. To be able to get the services from various services, the user has to authenticate to the Keystone server.

First, the user gives identity and credentials (e.g., password) to Keystone. Assuming the user is registered, Keystone authenticates the user, creates a tamper-evident digital token that contains information about the user, the endpoint information of each service (e.g., Nova, Neutron, etc.), and the operations the user is allowed to perform at each of those services.

Keystone authentication performed by using public-key cryptography. It uses a digital signature, and the usage of the digital signature in this system is unconventional. It is well-known that a significant drawback of the digital signature is that it takes a longer time to sign and decrypt the data. For this reason, in the real-world, a digital signature is used for small-sized data (typically hashed data). But the existing system of keystone signs large amounts of data, and this makes the keystone exists system a non- standard and inefficient for high-volume deployments.

The critical point is that a significantly more efficient and standards-based authentication protocol for OpenStack developed. It is feasible to re-designing and re-implements OpenStack's authentication protocol implemented in its Keystone component by employing different approaches. Either the authentication protocol is modified, or sometimes the multi-factor authentication system is implemented.

Keystone is one of the OpenStack components used for providing identification, authentication, and authorization service. This service categorized into two primary functions, which include user Management and Service Catalogue. User Management keeps track of user's necessary data, such as what roles the user has, which project the user belongs to, etc. Service Catalogue keeps track of what services are available and provides the location of the services' endpoints.

Keystone provides identity, token, catalog, and policy services — a public key-based mechanism used in the keystone's authentication system. Public key cryptography allows users to communicate securely over public networks and verify the identity of a user using a digital signature. A digital signature is an electronic signature that can authenticate the user.

In a digital signature, a sender typically uses her private key to sign the data, and the receiver uses the sender's public to the key to verify the signature. A Certificate Authority (CA) plays the trusted role to vouch for the identity of the user with a specific public key.

In OpenStack, the keystone can play the role of a CA using the keystone-manage utility or done by a third party. A Keystone *PKI token* used for authentication. A PKI token is nothing but a token signed by the keystone using its private key. Keystone uses cryptographic message syntax (CMS) within PKI. For this reason, the token referred to as CMS token. Whenever the user authenticates with his/her user name and password, keystone gathers credential data (e.g., user's roles) of the user and creates a token and places them in a file called user metadata. The metadata contains all information of the user like token, service catalog, user role, etc. It also includes an issue and expiration date and the id of the token.

The project information follows next, after which the service catalog information placed. The service catalog has the information on the service(s) and related endpoints the authenticating user can avail. The endpoints are where the services should connect to obtain a specific service (e.g., compute vs. network service). After the endpoints, the information about the user listed. It shows the roles of the user, username, and id of the user. Again, this data called CMS data because the id of the services here written in CMS format, and the signed CMS data is called CMS token.

When a user logs-in with username and password, the keystone gathers all of the information mentioned above and generates a CMS token and sends it to the user's workstation. The user's OpenStack client program in her workstation caches the token locally and uses it for later requests. When the user later requests a service, the client sends the token along with the service request to the Keystone endpoint. The OpenStack service verifies the user's signature and responds with the token.

When the client needs any of the services like nova, glance, cinder, etc., it sends a request along with the CMS token. The target service receives the CMS token and verifies the signature with keystone, and provides the requested service if the token is valid and the user is authorized.

Once a service receives the PKI token, it verifies the correctness of the same. The verification of the taken includes verification of the digital signature, token expiry date and time, and then proceed to handle revoked tokens — a keystone digital certificate required for verification of the digital certificate. The digital certificate can be obtained from KEYSTONE or through a third-party certificate authority. The CMS taken is verified using the certificate. If the signature is found valid, then the metadata is decrypted and then proceed to check the expiry of the token. An error is flagged if the digital certificate does not tally or the token expired.

Token verification carried using the token revocation list. The open stack services update the revocation list when the service requested is provided or when the token is either expired. The id of the token is the md5 hashed token, which is revoked by the service. Once a token is revoked, no more the token is valid. The Id, if present in the revoked list, then the token is found to be invalid and if not the token is considered as accurate. Once the token found to be correct, the service is allowed, and a response provided to the end-user. If the request is for a VM, then the VM is created and the IP address of the VM along with the valid port number provided to the user, who in turn uses the IP address for further processing required by the user, such as executing a program.

The whole process of authentication based on the digital certificate and the method has significant drawbacks.

1. A considerable amount of time taken for processing the token through digital certificate affecting the response time
2. Digital signature help securing the data integrity but not the data confidentiality as the token its self has essential information about the open stack system such as the details of the services, endpoints, etc.
3. Tokens once used are revoked, and are not re-used, leading to the signification token processing for revoking, removing, invalidating, etc.

The Identity service supports client authentication through TLS (transmission Layer Security), which might be enabled. TLS client authentication provides an additional authentication factor, in addition to the user name and password that provides more excellent reliability on user identification. It reduces the risk of unauthorized access when user names and passwords might be compromised. However, there is additional administrative overhead and cost to issue certificates to users that might not be feasible in every deployment.

The cloud administrator should protect sensitive configuration files from unauthorized modification achieved through configuring using mandatory access control frameworks such as SELinux, for protecting data that include keystone.conf file and X.509 certificates.

Client authentication with TLS requires certificates issued to services. An external or internal certificate authority can sign these certificates. OpenStack services check the validity of certificate signatures against trusted CAs by default, and connections will fail if the signature is not valid or the CA is not trusted. Cloud deplorers might use self-signed certificates; in this case, the validity check must be disabled, or the certificate should be marked as trusted.

Fernet tokens are now the default token provider in most of the cloud computing system. Fernet is a secure messaging format explicitly designed for use in API tokens. Fernet tokens are non-persistent (no need to be persisted to a database), lightweight (within 180 to 240 bytes), and reduce the operational overhead required to run a cloud. Authentication and authorization metadata bundled into a message packet payload, which is then encrypted and signed in as a Fernet token (within 180 to 240 bytes).

Unlike UUID, PKI, and PKIZ tokens, Fernet tokens do not require persistence. Pruning expired tokens from the token database is no longer needed when using Fernet tokens. Since Fernet tokens are non-persistent, they do not have replicated as long as each identity shares the same repository, the Fernet tokens used across the services offered by cloud computing systems instantly across nodes.

Compared to PKI and PKIZ tokens, Fernet tokens are smaller in size, usually kept under a 250-byte limit. For PKI and PKIZ tokens, more significant service catalogs will result in longer token lengths. This pattern does not exist with Fernet tokens because the contents of the encrypted payload kept to a minimum.

Use of fernet Tokens

Fernet tokens are message packed tokens that contain authentication and authorization data. Fernet tokens are signed and encrypted before being handed out to users. Most importantly, however, Fernet tokens are temporary, which means no persistence across clustered systems is required. The need for the user to make multiple calls to several cloud computing components for getting access to a service or resource gets minimized when fernet tokens used.

However, the fernet format has problems that make it non-ideal. The fernet specification abandoned, making it hard to get changes into it and thereby into the cryptography implementation of it. Moreover, the fernet specification not recognized by any standards body and therefore not as carefully audited as an IETF standard, making it more susceptible to zero-day vulnerabilities. Addressing these vulnerabilities falls solely on the cloud computing service providers.

Some of the requirements of the users that are not supported by the fernet tokens

1. Need for a non-persistent token that does not depend on symmetric encryption or signing implementation. An implementation built on asymmetric signing or encryption allows the distribution of public keys from one node to another instead of synchronizing a repository of symmetric keys, which makes it easier for the cloud components with read-only capabilities strictly used for token validation. The asymmetric encryption method helps to deploy keys in read-only regions where the token validation undertook, while the tokens issued from a central identity management system in a separate area.
2. Need for a fallback mechanism in the event of noticing a security vulnerability in the fernet-spec or the implementation of the cryptography
3. As an operator, I want to be having a token provider to fall back on in the event there is a security vulnerability in the fernet spec or the cryptography implementation consumed by keystone.
4. The need for a token used not be used within a cloud computing system but also with other prices of the software, which is outside the scope of cloud computing systems.

Thus there is a need for implementing proper authentication system within cloud computing systems to make it more secured from the point of Authenticating the users

3. SECURITY ISSUES RELATING TO KEYSTONE MODULE

The critical study of the keystone module carried to find security lapses contained within the keystone module. A detailed presentation on the vulnerabilities existing in the keystone module provided in the following sections

Invalid LoginAttempts

The Identity Service (keystone) does not provide a method to limit access to accounts after repeated unsuccessful login attempts. A pattern of repetitive failed login attempts is generally an indicator of brute-force attacks. This type of attack is more prevalent in public cloud deployments. One can mitigate this by using an external authentication system that blocks out an account after a configured number of failed login attempts. The account then might only be unlocked with further administrative intervention.

Detection techniques used to mitigate damage. Detection involves a periodic review of access control logs to identify unauthorized attempts to access accounts. Possible remediation would include reviewing the strength of the user password or blocking the network source of the attack through firewall rules. You can add firewall rules on the keystone server that restricts the number of connections; this can help reduce the attack's effectiveness. Besides, it is useful to examine account activity for unusual login times and suspicious actions and take corrective actions such as disabling the account.

User authentication Issue

In keystone, two authentication functions, namely tempAuth() and "swAuth()used which use username and password for authentication. When authentication successfully performed, the user receives a token used to identify him to the system later to access the services. The provided token has a configurable expiration time, and its default value set to 24 hours.

Most of the security systems implement the concept called "Authentication delegation" through the process of issuing a confirmation of delegation through SAML format. In OpenStack, the authentication delegation system, as such, is not implemented.

Password strength issue

Since all OpenStack projects provide username and password combination to authenticate users, it is crucial to take a closer look and study the strength built into the password.

The "Electronic Authentication Guideline" created by NIST provides some rules to prevent users from choosing bad passwords, including checking a password against a dictionary of commonly used passwords, specifying minimal password length, and requiring the use of different characters (lower-case, upper-case, non-alphabetic).

Unfortunately, no such requirements (password length and special characters) exist in OpenStack. There are also no dictionary checks so that users can register with a password as short as one character.

Password storage Issue

Storing passwords is a well-known problem in all information systems that use password authentication. The general practice for information security requires that the administrator ensure passwords not stored in clear-text, but somewhat encrypted. It is also essential to provide limited access to stored passwords.

The authentication system "tempAuth" stores username and password in a configuration file in which all passwords stored in plain text format. The location of the superuser credential stored in the same folder. By default, each user in the system possesses read access to this file. Such an approach enables system users to obtain the password of other users and gain access to their accounts quickly. The main reason why "tempAuth" was never considered by developers to be an option for production deployment as it takes the passwords of others.

The authentication system "SwAuth" uses a configuration file where the super admin password is stored. Unlike the "tempAuth," "swauth" possesses properly configured access rights to secure password data. The only security concerns that arise with "swAuth" are clear-text stored passwords within this file. Because of this issue, and an inside attacker would gain a superuser account on the system, thus being able to find out user passwords. OpenStack should consider hashing passwords before storing them in the password file.

Both "tempAuth" and "swAuth" lack the appropriate protection of passwords. A recommendation for both authentication systems taken from NIST's "Electronic Authentication Guideline" would be to store passwords "concatenated and hashed with an approved algorithm, so that the computations used to conduct a dictionary or exhaustion attack on a stolen password file would not be useful to attack other similar password files."

Tokens of authentication

Authentication tokens play similar roles as identifiers for web applications. An API, such as an OpenStack service, is used to authenticate a user. Successful authentication generates a token used to authorize service requests. The password and username gave as input to the API interface. When authentication succeeds, the resulting feedback includes an authentication token and service catalog. The tokens remain valid for 12 hours. Issued tokens become invalid when the token is expired or when then token canceled.

The authentication executed over a secure channel, such as Transport Layer Security (TLS); otherwise, an attacker could obtain a user token by performing a man-in-the-middle-attack and remove the user who received the token from the authentication system.

DDOS Attack

The users can also attack using A Distributed Denial of Service attack, which executed by sending too much traffic on a server. This server, however, can handle only a certain number of requests, which results in it either a complete failure or the slow down of the services.

4. PROBLEM DEFINITION

The Identity service provided within the keystone module leaves several vulnerabilities making any cloud computing system vulnerable for attacks. A sound security mechanism is needed to be built within the open stack cloud computing system so that the OpenStack system can be used effectively for creating either public or private clouds. A secure authentication system implemented for establishing the user credentials so that the desired access to the services provided. The authentication system secured in addition to making the authentication system faster. The authentication system must be implemented in an open platform so that the operation easily integrated with other internationally proven and available authentication systems.

5. RELATED WORKS

Platform Specific digital signatures and tokens used for authentication within Openstack. An independent, decentralized, and flexible Mechanism that serve the purpose of authentication presented by Razib Hassan Khan et al., [1]. They have used OpenID, which is an open-source for the development and implementation of authentication within OpenStack. They have developed and offered the authentication system as a service. The platforms prosed by then are built web services and have implemented a single-sign-on for accessing multiple services. The users signing into an operating system is also used as login into OpenStack. They have shown how the users who registered into OpenID can log into Dashboard/Django GUI.

R. T. Fielding [2], in his thesis, has presented REST architecture for designing web applications. REST is stateless and works on the principle of cloud computing. The API uses a secured https protocol for proving communication between the users and a server. The users through API call logs into the server and get connected.

Jamie Bodley-Scott [3] presented that identity management is now being made user-centric from organization-centric approaches. Access to multiple service points implemented through user-centric approaches that are scalable and flexible. The user-centric methods based on single sign-on for making available different kinds of services. Through a single sign-on, a federation of login systems used which improves the usability of service-oriented system extensively

EC2API client [4] and the *python-nova* OSAPI client [5], which are API tools used for authentication of the users into cloud computing systems. A GUI hides the use of API and makes accessing a service much simpler for the users. WEB-GUI became the widely acceptable front end for making available the cloud services to the end-user and also the administrators. The Horizon component of OpenStack provides the GUI through which the user can access the services without the need for accessing the system through the use of API. There are, however, many shortcomings in the way the dashboard provides authentication of the end-user.

API generally used within OpenStack EC2API [6] or OSAPI [7]) for implementing front-end GUI services. Through API, the processes that are related to Access control, authentication and cryptographic algorithms, and generation systems handled within OpenStack. Many weaknesses found when the authentication systems implemented using API calls. The user names and passwords when used within a different framework on which the service provider has no control for authenticating the user. Once the user verified, administrator credentials used for retrieving the credentials of the user. The server that provides the service in the open stack does not participate in the authentication process but rather depends on the credentials provided by the administrator.

OpenStack does not support federated identity management. Many federation based authentication systems such as OpenID [8][9[, SAML [11] [12], Shibboleth[10] used within the open Stack for implementing the authentication system

Administering the policies and taking policy-related decisions are situated at specific policy decision points (PDP), and there can be many policy enforcement points that communicate with PDP for effecting the authentication and access control to the resources. Policy enforcement points situated within a cloud computing system that delivers with PDP for providing access to resources [13][14]. In OpenStack, the front-end GUI server within the client is a separate area within OpenStack. The user credentials have to be stored within the GUI server as OpenStack has no support for federation with the different authentication servers. The implementation of Multiple PAPs and PDPs is not possible when a centralized authentication server was not in place. There should be trust between the Client (Front-end GUI) and the cloud computing system, which is possible when a tightly coupled GUI and Cloud computing system implemented. The WEB server tightly coupled to the Back-end cloud computing servers which provide the services required by the users.

OpenID is an authentication system available as open sources extended to implement user preferences. The system provides a centralized Identity system that is user-centric, which means the users can opt for the kind of system that needs performing for enforcing the authentication to access the cloud computing resources.

OpenStack is a cloud computing software that is available as open-source. OpenStack initially designed to of Infrastructure as a service (IaaS). Users can ask for the kind of machine in terms of CPU power, extent memory and storage required, and the nature of the operating system that must run on the Machine. Users install an IDE and develop their application. Users can also connect their System software like database management software etc. Users can deploy their claims on the machines and also run the application.

In Open Stack, Security is built through the process of authentication and access control and also providing security infrastructure that can be used by the users to enforce security on their own. The Keystone Module within open stack takes care of security enforcement through primarily utilizing a process based on Tokenisation. The Keystone module provides tokens to the users, and the users access the services offered by OpenStack with the help of tokens

Open Stack has not used any standard for implementing security enforcement within the cloud. The security implemented within the open Stack system is nonstandard. Kerberos is an authentication standard. Sazzad Masud et al. [15] have studied the Kerberos system and have shown the wat the system implemented within the open stack as an independent component that federates with a keystone.

OpenStack [18] is an open-source system that offers Infrastructure as a Service (IaaS)[17]. Open stack allows the users to provision the virtual machine with storage, computing resources, which are provisioned by the OpenStack components such as SWIFT, NOVA, etc. RACKSPACE and NASA together have developed OpenStack in python language [16].

C. Kaufman et al. [19] have presented the way the authentication protocol system used by OpenStack implemented within the Kerberos protocol. They have also introduced a prototype authentication system based on Kerberos standard

Keystone uses Digital signatures for implementing the authentication system. Authenticating a massive amount of text would be cumbersome [18] when massive traffic between the clients and the cloud computing system expected. The authors have proposed to hash the data so that the size of the text reduced and then they have applied a digital signature

The Keystone service developed using many interlinked and structured internal services [20]. The services offered by keystone can be services provided by keystone include policy services, catalog services, token services, and Identity services. The authentication system implemented within the keystone based on critical public infrastructure that helps the users to communicate in a secured manner over public networks — the identity of the users established through a digital signature. A user uses a private key to sign the message digitally and the receiver uses the public key of the user to decrypt the message and find the identity of the user who sent the message. A certificate authority, either internal or external, will verify the trustworthiness of the public key used by the user.

Marek Denis et al., [21] have explored the implementation of identity federation within the OpenStack system through the use of local identity called “Domain Accounts”

Darshan et al. [22] have presented that keystone plays a significant role in binding all cloud computing projects together, each project implementing a service. There is a need to protect the resources used by the keystone such token repository, the identity of the users and resources, the endpoints, etc. The security of the open stack system is enormous as all the possible attackers have the source code of the OpenStack in their hands. The authors have analyzed the security requirements open stack and formed a threat model. A RESTFul API based authentication system that offers various security services needs implementation within the OpenStack system.

Most of the organizations around the world are shifting to cloud computing infrastructure for supporting their IT requirements due to cost, reliability, and availability of the required resources as and when needed. Many cloud service providers have already come into the market, playing a significant role. Some of the service providers that have come into play include SalesForce, Amazon, Google, Microsoft, Rackspace, Oracle, Verizon, etc. [23].

Security concerns are many when one wants to use cloud computing systems. Security is a real barrier to using cloud computing systems [24]. A survey conducted in 2016 revealed that security risks are the primary concerns/obstacles in using cloud computing systems.

Several frameworks developed in the past related to cloud computing systems. Some of the frameworks are open source based. The customers use the frameworks for building private clouds that offer different types of services. Some of the notable frameworks include Cloud Stack, Eucalyptus, and Open Nebula. Off late Open stack has become the most sought out Open sources based framework for developing users their private networks [25].

OpenStack is vulnerable to attack. Experimentation using a porotype specification revealed that the dashboard component of Open Stack is sensitive for attacking. Most of the researchers have offered different kinds of solutions used for making the OpenStack framework secured. The keystone module of open stack provides another type of identity services that include identification, cataloguing, management of policies, and dealing with tokens for authentication. The identity services offered by the keystone module provided as a set of services situated at more than one endpoint. The services initiated through frontend calls. An authenticate call initiated from the user frontend will validate either project or user credentials, and on finding the eligibility, an authentication token issued using which the users access the service [26].

A study conducted, and an analysis of security issues relating to open stack carried especially considering object storage service. Security requirements, as stated in two different standards released by NIST(National institute of standards and technology) and ENISA (European Network and Information security agency) and came out with a set of security requirements implemented within OpenStack [27].

GidwaniIshan et al. made another study that focuses on the security issues and threats existing in OpenStack system., [28]. The authors argued that OpenStack did not implement any complexity within the password system and also that the passwords stored in plain text. They have conducted a penetration test using some of the existing tools and have come out that Open Stack is prone to future attacks as many vulnerabilities still existing in the system

A new Enhanced authentication system that works in conjunction with original authentication system implemented within the keystone presented by B. Cui and T. Xi in [29]. They have shown details and the way the new model performed. They have compared the features of the new model with the features supported by Keystone and also have introduced the way the new model provides a high level of security by subjecting the open stack system with the attacks that cannot be mitigated by the keystone system.

A study of the features supported in the keystone module [30] has found many weaknesses and a lack of support for access control, authentication based on attributed provisioning, audit mechanisms, and policy-based security enforcement. They have carried a threat and identified threats that exist concerning interfaces, components, internal processes of the elements within OpenStack.

Series of versions of the OpenStack released, leading to the improvement of security enforcement that mitigates many of the vulnerabilities existing in the open stack. The open stack being open source exposes many threats and vulnerabilities. Open source-based Testbeds used for testing cloud computing systems, and these testbeds used to test new methods included in the OpenStack system for testing resource provisioning and management of services deployed under Multi-data centers [31].

The architecture of most of the open sources is similar [32]. Most of the cloud computing architectures consider including a cloud controller and a set of nodes on which several services implemented. The controller controls the instances, network, administrative interfaces, and schedules the interfaces. The nodes run instances of VMs through the use of available resources.

Authenticating the users for providing secured storage and access to the information is required, when it comes to service-oriented information exchanges — identity management systems needed for ensuring confidentiality and security considering both sides of the client and provider. Many drawbacks exist within many of the cloud computing systems, including open stack, which causes data violations, unauthorized access. A single point of failure happens due to the use of centralized access. Security of cloud computing systems enhanced through Federated Identity management, which leads to structured, adaptable and systematic implementing of the security systems within cloud computing systems [33].

Performance analysis of a two factor authenticated system carried by J. M. Alve, T. G. Rodrigues [34] using the different hypervisors, which include VMware, Xen, KVM, etc. They have used the user name password in the first instance and then followed by OTP based authorization. They have used OpenID protocol, so that single sign-on access to the services provided. They have shown that KVM hypervisor performance extensively well using a two-factor approach [34].

A cloud computing adaption framework is proposed by Vicor Chang et al., [35], which can be customized by the user for implementing the organization-specific security requirements. They have presented that security enforcements applied in real-time through a Multi-Layer approach. They have used three layers for implementing security through a firewall, intrusion, and access control, which all are implemented in three layers.

Kryszt Benedyczak et al. have presented the use of middleware for implementing federated computing [36]. They have proposed a method that does not require either certificates or delegation mechanisms. They have used a component called "Unity" for serving the identity management services. The method proposed by them allows many federation integration approaches that include integrating with OpenID and SAML.

Benjamin Ertl [37] presented that Authentication for every kind of service has to be rendered based on cross-domain identification. They have introduced a protocol that considers the issue of linking different accounts associated with a client. The protocol proposed by them supports verification of authentication for each of the services requested by the clients. They have put their recommendation in terms of the existing federated infrastructures.

Controlling access to different resources considering Fine-grained access control, Scalability, data utilization, privacy preservation, and revocation of privilege is most complicated. A scheme covering these aspects was proposed by Rohit Ahuja [38] based on encryption carried using a set of attributes. Encryption of data undertaken through consideration of a set of attributes. They have considered that the users hierarchically organized, with each user carrying specific attributes. The characteristics are selected based on the path to be used for moving data from one user location to another, keeping because of the scalability. The authors have presented the method of hybridization of re-encryption and attribute-based encryption to realize the flexible revocation of system privileges.

Security and privacy are the two major concerns of the users who store their data in the clouds. Several security concerns arise, which include access control, Data integrity control, access logging, access auditing, and managing the identity of the users when data transmitted between the user and the cloud. Many complex issues lead to multiple open problems requiring in-depth research carried Bhale Pradeep Kumar [39].

A single sign-on is sufficient to access the services proposed by multiple service providers recommended by Jaweher Zouari [40]. They have proposed identity as a service framework in which an Identity Finder system incorporated. The identity finder system, associates service providers with identity providers systems after taking the consent of the users. They have proposed additional functionality that helps to transform between different standards and mapping semantics relating to varying attributes so that the same identity context preserved over the entire system

Policy-based methods and mechanisms used for effecting access control have been used by Georgios Katsikogiannis [41] for implementing multilevel identity integration, authentication, and authorization for providing secured access to cloud computing resources. They have used SOA for implementing a policy-based security system. They have analyzed Identity integration, user roles, authentication, authorized access control and used for validating the rules.

X Darth protocol has been used by Quratulain Alam [42] for implementing identity management that spread across several domains. The follow of information that takes place when XDAuth used is modeled using Petri nets at a high level. The authors have used the Z language for analyzing the rules of information flow. The model verified by using a Z3 solver.

A cryptographic primitive, which is key-homomorphic embedded into RDIC (identity-based remote data integrity checking) protocol, which considers the user identity for reducing the complexity of a system. The modified RDIC helps in minimizing the cost of implementing PKI within RDIC, Yong Yu [43].

A scheme that helps to implement RIBS (Revocable Identity-based signature) proposed by Xiaoping Jia [44] uses an external cloud revocation server used for carrying all critical updates. Xiaoping has proved that a new framework that incorporates RIBS is highly resistant to foraged messages and different identity attacks. They have convinced that RIBS is highly efficient when compared IBS scheme.

A cloud computing platform suffers from both external and internal attacks. Nit many Mechanisms/methods proposed to deal with internal attacks. Carlos Eduardo et al. [45] described self-adoption schemes to handle insider threats. The authors have presented the way the self-adaption systems introduced into the Open Stack cloud computing platform

The self-adaption mechanism found to deal with the uncertainty that exists in a wide range of applications. The component "Keystone" contained in open stack can be included with self-adaption mechanisms so that internal threats counter-attacked

Carlos Eduardo et al., have presented that adds self-adaption components to Open Stack architecture to handle insider threats. They have analyzed several threats occurrences that can happen within the open stack and have evaluated the impact of the treats that occur in several scenarios. The self-adaption system explained considering several threat scenarios.

Mell PM et al. [46] explained that the distribution of data among different servers is primarily dependent on the type of cloud (private, public, and Hybrid). The data distribution is also dependent on the users who are either internal, external, or both. The data distribution aspects considered while making available infrastructure as service through an open stack cloud computing system.

The self-adoption techniques did not lead to complete protection considering security and privacy, especially when insider threats are involved. Many contributions made to deal with securing the cloud computing systems [47, 48, 49], but none of these methods could solve the issues of insider attacks.

Cappelli DM et al. [50] have explained that an insider threat is either a user or process that has authorized access to the internal resources and can attack the integrity, availability, and confidentiality of the data.

Cole DE et al. [51] have explained that insider threats are not the same as those connected with the cloud computing components, which are either hypervisors or brokers. Insider threats can be catastrophic resulting in considerable losses to the organizations as explained by Duncan A et al., [52]

Self-adaption systems proved to be effective in dealing with insider threats as the mechanism deal with un-certainty considering a wide range of application and especially with the apps that are related to effecting access control mechanisms [53, 54, 55, 56]. De Lemos R et al. [57] have explained that the self-adaption systems could modify/update their behavior or data structure at run-time, thus dealing with the dynamic management of insider attacks.

An insider threat generally caused by authorized users of the system [58]. The internal user regarded as the inside attacker. Cert et al. [59] defined an employee, business partner, and contractor who has access to the internal information resources as the inside attackers. The users have intensions to take advantage of the company's data for unlawful activity affecting availability, integrity, and confidentiality regarded as inside attackers [59] [60]. Cappelli DM et al. [59] have considered the issue of security from the point of likely abuse of the data by the users, which can lead to some threat [61]. Insider risks classified as unintentional and intentional. Only intentional threats are classified into insider attacks [60]

Various models presented in the literature relating to authentication, authorization, and access control helps to implement different security measures that help to ensure integrity, confidentiality, and availability of the data as per the user requirements. Many models were presented in the literature which includes RBAC (Role-based Access control) [64], ABAC (Attribute-Based Access Control)[65]. These models, based on the assignment of attributes to roles either through relationships or rules, assign permissions to access the resources, find Rules that express relationships between the users and roles. An identity federation management system includes identity providers, assigned attributes to the users and uses the Authentication related infrastructure provided by the service providers for effecting the security enforcement [66].

The system that implements ABAC/RBAC relies on the software components that protect access to several resources. The self-adaptive systems use the inputs provided by the elements meant for controlling the access for changing their behavior [68]. MAPE-K framework implements the self-adaptive model combined with the components that protects access to the resources [69].

An enhanced scheme has been presented by Yapping Chi et al., [70] that strengthen the authentication system implemented within the keystone module of the OpenStack System. The scheme uses FreeIPA for including a sentinel that performs authentication, service management, and access control. The effectiveness of the sentinel tested by exposing the open stack to the external users.

Clouds provide resources that are shared by several users/tenants through availing of different services that are made available by cloud computing providers — the accessibility to the resources controlled so that one user does not get into another user jurisdiction. Several security mechanisms must put in place, which includes authentication and access control to provide non-conflicting access to the resources [71][72].

The authentication systems implemented within the cloud computing systems must be flexible such that the authentication system implemented varies based on the kind of resource requested by the user. Many existing cloud computing systems implement proprietary authentication systems through uses of signatures and tokens. Khan [73] has designed and implemented a model based on the OpenID framework so that limitations existing in proprietary protocols removed.

A new authentication framework is proposed by Anisetti [74], which is deployed on a single open stack node and proved the effectiveness of the framework in implementing security within OpenStack. Cui [75] et al. have analyzed the security implementation within an OpenStack, considering each component separately. They have proposed a new model based on symmetric and asymmetric encryption the feasibility verified by deploying the same within OpenStack.

Chi Yaping et al. [76] have used the FreeIPA framework and developed a sentinel which has been introduced into an open stack and proved the effectiveness of the same. Several papers published relating to securing various aspects within cloud computing systems some of which have not been included in Open Stack **[77][78][79][80][81][82][83][84][85][86]**.

6. INVESTIGATIONS AND FINDINGS

The Vulnerabilities were 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 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 same, the more 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.

6.1 Implementing JSON Tokens within Open Stack

Overview on JSON Tokens

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.

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.

.

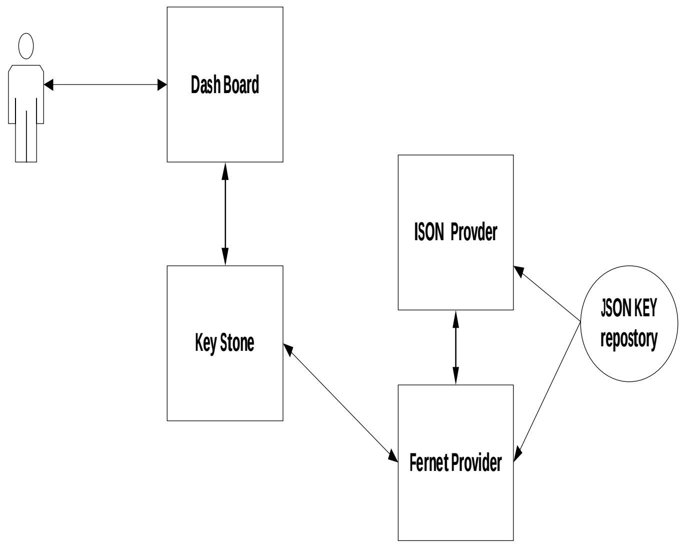

Figure 5: Implementing JSON Tokens within OpenStack Cloud Computing System

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.

The architecture of JSON Token implementation within OpenStack shown in figure 5

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.

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 configuration file related to Fernet will be modified to redirect the

6.2 JOSN Token Payload

The payload of the JWS will have the following components:

1. A string containing the ID of the user who authenticated for the token
2. A numeric value for token expiration
3. numeric value relating to the time a token was issued
4. The following components will be included in the token along with payload to keep information about the elements used for developing the token.
5. Authentication methods used to obtain the token
6. Audit Ids required for claiming audit information associated with a Token
7. Open Stack system ID in case of system-scoped tokens
8. OpenStack domain ID in case of domain-specific Tokens
9. Open Stack Project ID in case of project-scoped tokens
10. Open Stack trust ID in case of trust-scoped tokens
11. Open Stack Application Credential ID in case of credential tokens
12. Open Stack Group Ids in case of federated tokens to carry a temporary user's group assignments
13. Open stack idpId in case of federated tokens to take the ID of a user's identity provider
14. Open Stack protocol ID present in federated tokens to denote the protocol used by a federated user to authenticate
15. Open stack access token present in OAuth tokens

The JSON application is developed in python language and using PyJWT and JWCrypto library. ES256 algorithm is used for digitally signing the tokens, is dynamically selected so that it becomes impossible for anybody to attack the tokens. Users will request and present tokens in the same way they currently do with Fernet tokens. There is no need to add or change any APIs.

6.3 Implementing Key Setup and Rotation Strategy

Much like the Fernet implementation, a JWT provider will require a key rotation strategy. Since ES256 relies on asymmetric signing, the suggested rotation strategy will be slightly different, known with Fernet.

The Fernet implementation requires the usage of a staged key, which is just a key with a unique name, to ensure tokens validated during the rotation process. This kind of usage of key names not required in the case of JSON tokens. The following steps should be sufficient to perform key rotation without token invalidation due to missing signing keys.

1. For every server, A key pair created in terms of Private and Public Key.
2. The public keys exchanged between the servers
3. Tokens every server ca be validated anywhere as the public keys of every sever contained in every other server.
4. Any server can be designated to rotate the keys. The key rotation by the servers in a round-robin fashion based on time slicing.
5. A server responsible for key rotation creates a key pair and sends its public key to all the other servers. Once an acknowledgment received from other servers that the key loaded into the key repositories of the respective servers, the key rotating server will start singing the keys. The server rotating the keys can wait for some time and begin singing the tokens if in case there is a need to avoid the process of acknowledgment.

The Modified Authentication process meant for implementing multiple signatures to make the system much more secured as it becomes quite challenging to attack numerous signatures.

The PyJWT library does not have functions to sign the tokens using multiple signatures. A feature added that is capable of singing with numerous signing algorithms. Changing the key pairs would not affect the system as the servers have keys used to sign or design with one of the digital signature algorithms.

A separate function is implemented within the JSON application to revoke the Key-pairs. A count maintained for each of the key-pair stored in the repository, The key-pair that has achieved a threshold value is automatically invalidated and deleted from the repository subsequently

6.4 Crypto-Agility through JSON Library

An expansion to the JWS specification implemented to deal with more algorithms that can be selected randomly and dynamically for singing the tokens digitally and also include more algorithms that can also be chosen dynamically for validating the tokens once received by a server. To achieve crypto agility, the Tokens generated by different algorithms converted to Fernet Tokens as a standard format for exchanging the tokens.

In the JWT, the users can change the kind of algorithm used for affecting the digital signature or validating the tokens by the servers by specifying the same by making changes to the JOSE header. A validation weakness or wrongly digitally signing the tokens can thus be affected. The source of the algorithms in the Python Library must be validated before the same used for securing the tokens.

Since JWT is a widely used web standard, this will have a net positive impact on security. The implementation will use asymmetric signing, reducing the risk of having to replicate or transfer private keys from one host to another. Since the token payloads are signed, data within the token will be readable to anyone who has the token. The token can only be validated using the corresponding public key of the private key used to sign the token initially.

In most cases, JSON Web Tokens will have a header, payload, and signature where each section is delimited by a period (.). The header contains the name of the algorithm used to verify the integrity of the token — the name of the algorithm stored as the "alg" attribute of the header. The library validating the token uses the algorithm specified in the header to perform an integrity check and compares its results to the signature portion of the token.

Security concerns documented and raised that describe the issues with allowing clients to dictate algorithms used for token verification. The problem is a concern specifically with applications that support asymmetric and symmetric signing. An attacker could effectively bypass the verification check of a token by using a published, or known, the public key to generate a JWT with a symmetric signing algorithm.

The issue would be applicable if keystone supported signed tokens and encrypted tokens with the same token provider implementation. This vulnerability addressed across various libraries after its discovery, but the keystone should be aware of the overall technique that leads to it in the first place. We can mitigate this type of vulnerability in keystone by ensuring keystone doesn't blindly allow end-users to specify which algorithm used to verify the integrity of a token (e.g., only implementing support for ES256) and also ensure that the "alg" supplied in the token header-only populated by keystone and also to ensure that keystone only issues tokens of a single encryption or signing strategy (e.g., not allowing users to get signed token and encrypted tokens from the same server, thus mixing asymmetric and crucial symmetric usage at runtime)

Implementing JSON Tokens within Open stack Systems

The steps to be undertaken for implementing the JSON tokens shown in Table 1

Table 1 : Steps for implementing JSON token system in OPEN STACK

Serial Number	Step Undertaken
1	Add a new JSON class into Keystone Module
2	Modify FernetUtility Class such that JWT API library called for either creation of a token, or digitally signing the token or validating the token or revoking the existing token
3	Add a function into JSON Class which called from fernetUtility class for rotating JWT signing keys
4	Add a function into JSON class which called from FernetUtility class for verifying the Algorithms used for digitally signing the tokens
5	Add a function into JSON class which called from FernetUtility class for validating the tokens
6	Generalize the Token Formatter class and derive JSON formatter class used for formatting the tokens

Different Python Libraries required for implementing the JSON Tokens, the details of which provided in Table 2.

Table 2: Use of Python Library

Serial Number	Library Name	Description of the Library
1	PyJWT	This library only supports token signing or JWS. It does not support JWE, or authenticated encryption, yet. A minimum version of 1.0.1 is required, but this library already included in the OpenStack global requirements repository.
2	python-Jose	This library only supports token signing or JWS. It does not support JWE, or authenticated encryption, yet. OpenStack global requirements did not include the provision of this library

Serial Number	Library Name	Description of the Library
3	JWCrypto	This library supports both JWS and JWE, but both the environments included in OpenStack global requirements.
4	Authlib	This library supports both JWS and JWE, but its licensing is incompatible with OpenStack as it is AGPL

.

7. CONCLUSION

OpenStack is a prominent open-source software for providing the infrastructure as a service to the users. It is necessary to investigate the sufficiency of the security built into the Open stack as OPEN STACK contemplated to be used by many users.

Management of tokens issues to the users is the most crucial issue. The system must be such that it is difficult to attack the initial logins or the tokens exchanged for providing authentication to the user.

The fernet tokens used within the keystone is weak as it exposes much vulnerability that can be exploited by the attackers. Fernet token formatting is also nonstandard. A JSON token follows the open standards and therefore are ruggedized. The fernet classes can be generalized, and the functionality required for the development of tokens, digitally signing the tokens, validating the algorithms and keys, and also for rotating the keys using the JSON library effected.

REFERENCES

1. Razib Hassan Khan, Jukka Ylitalot and Abu Shohel Ahmed, OpenID Authentication As A Service in OpenStack, 7th International Conference on Information Assurance and Security (IAS), PP. 372-377, 2017
2. R. T. Fielding, "Architectural Styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.
3. Jamie Bodley-Scott, "IDM09, Access or Identity, http: //www. opengroup. org/j ericho/idm2009_jbs .pdf."
4. "Euca-Tools, Eucalyptus Community, http://open.eucalyprus.com/wiki-/toolsecosystem, last accessed 10th May 2011."
5. "Python-OpenID 2.2.5, http:// pypi.python.org/ pypi/python-openid, last accessed 5th May 2011
6. "Amazon AWS EC2 API Reference, https://docs.amazonwebservices. Com /awsec2 /latest/ apireference/, 2011."
7. Rackspace US, Inc., "Openstack compute developer guide api 1.0, 2011."
8. "OpenID Foundation, http://openid.net, last accessed 14th June 2011."
9. D. Recordon and D. Reed, "OpenID 2.0: a platform for user-centric identity management," in Proceedings of the second ACM workshop on Digital identity management, ser. DIM '06. New York, NY, USA: ACM, 2006, pp. 11-16. [Online]. Available: http://doi.acm.Org/10.l 145/1179529.1179532
10. M. Erdos and S. Cantor, "Shibboleth architecture protocols and profiles, Http://shibboleth. internet2. edu/shibboleth-documents .html."
11. R. Philpott, E. Maler, N. Ragouzis, J. Hughes, P. Madsen, and T. Scavo, "OASIS Open 2008, Security Assertion Markup Language (SAML) V2.0 Technical Overview, Committee Draft 02, http://docs.oasisopen. org/security/saml/ post2.0/ sstc-saml-tech-overview-2.0 .html," March 2008.
12. J. Rosenberg and D. Remy, Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption. Pearson Higher Education, 2004.
13. S. Almulla and C. Y. Yeun, "Cloud computing security management," in Engineering Systems Management and Its Applications (ICESMA), 2010 Second International Conference on, 30 2010-April 1 2010, pp. 1-7.
14. D. Gollmann, "Computer security," Wiley Interdisciplinary Reviews: Computational Statistics, vol. 2, no. 5, pp. 544-554, 2010. [Online]. Available: http://dx.d0i.0rg/l 0.1002/wics. 106
15. Sazzad Masud and Ram Krishnan, Kerberos-Based Authentication for OpenStack Cloud Infrastructure as a Service, IT CoNvergence PRActice (INPRA), volume: 3, number: 2 (June), pp. 1-24
16. S. Mandy. Drawbacks of the digital signature. http://computerfun4u.blogspot.com/2009/02/ drawbacks-of-using-digital-signature.html.
17. P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical Report 800-145, National Institute of Standards and Technology, 2011. https://doi.org/10.6028/NIST.SP.800-145
18. Rackspace. OpenStack: The Open Source Cloud Operating System. http://www.openstack.org/ software/
19. C. Kaufman, R. Perlman, and M. Speciner. Network Security: Private Communication in a Public World. Prentice-Hall, 2002.
20. R. Xu. Keystone authentication. http:// OpenStackoz.blogspot.com/2014/08/ keystone-authentication.html
21. Marek Denis, Jose Castro Leon, Emmanuel Ormancey, Paolo Tedesco, Identity federation in OpenStack - an introduction to hybrid clouds, 21st International Conference on Computing in High Energy and Nuclear Physics, DOI:10.1088/1742-6596/664/2/022015
22. Darshan Tank, Akshai Aggarwal, and Nirbhay Chaubey, 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
23. Ristov S, Gusev M, Kostoska M. Security assessment of OpenStack open-source cloud solution, Proceedings of the 7th southeast European Doctoral Student Conference (DSC2012). 2012: 577-587.
24. http://www.hytrust.com/cloud-sddc-study/
25. S. Ristov, M. Gusev and A. Donevski, "Security Vulnerability Assessment of OpenStack Cloud," 2014 Sixth International Conference on Computational Intelligence, Communication Systems and Networks, Tetova, 2014, pp. 95-100 https://doi.org/10.1109/CICSyN.2014.32
26. [26] [3] https://docs.openstack.org/security-guide/identity.html
27. Slipetskyy R. Security issues in OpenStack, Master's thesis, Norwegian University of Science and Technology, 2011.
28. Ishan GidwaniIshan, Dasrath Mane. Security Issues In OpenStack, International Journal of Computer Science and Information Technology Research, Vol. 3, Issue 2, pp: (1147-1158), Month: April - June 2015
29. B. Cui and T. Xi, "Security Analysis of OpenStack Keystone," 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Blumenau, 2015, pp. 283-288. DOI: 10.1109/IMIS.2015.44
30. Ericsson, Keystone Security GAP and Threat Identification (Quick Study), OpenStack Folsom Release, 2014
31. Cirrus, O.: Open cirrus - open cloud computing research testbed (Apr 2012), https://opencirrus.org/
32. Ng, C.H., Ma, M., Wong, T.Y., Lee, P.P.C., Lui, J.C.S.: Live deduplication storage of virtual machine images in an open-source cloud. Proceedings of 2011, 12th ACM/IFIP/USENIX International Conference on Middleware. Pp. 81–100.
33. Rohit Shere, Sonika Srivastava and R.K. Pateriya, A Review of Federated Identity Management of OpenStack Cloud, International Conference on Recent Innovations in Signal Processing and Embedded Systems (RISE), 2017
34. J.M. Alve, T.G. Rodrigues, "Multi-Factor Authentication with OpenID in Virtualized Environments," IEEE Latin America Transactions, Vol. 15, No. 3, pp. 528-533, March 2017 https://doi.org/10.1109/TLA.2017.7867604
35. Vicor Chang and Muthu Ramacharandra, "Towards Achieving Data Security with the Cloud Computing adoption framework", IEEE Transactions on services computing, Vol.9, No.1, pp. 138-151, Feb. 2016
36. Krysztof Benedyczak, "Unicore 7 - Middleware services for Distributed and Federated Computing", IEEE, pp. 613-620, 2016.
37. Benjain E, Identity Harmonization for Federated HPC Grid and Cloud Services, IEEE proceedings, Pp. 621-627, 2016.
38. Rohit Ahuja, An identity is preserving access control scheme with flexible system privilege revocation in cloud computing, IEEE- 11th Asia Joint Conference on Information Security, pp. 39-47, 2016.
39. Bhale Pradeep Kumar, "Achieving Cloud Security using Third-Party Auditor, MD5 and Identity based Encryption", International Conference on Computing, Communication, and Automation, pp. 1304-1309, 2016.
40. Jaweher Zouari, An Identity as a service framework for the cloud, IEEE Proceedings, Pp. 1-5, 2016.
41. Georgios Katsikogiannis, "An Identity and Access Management approach for SOA," IEEE International Symposium on Signal Processing and Information Technology, pp. 1-6, 2016 https://doi.org/10.1109/ISSPIT.2016.7886021
42. Quratulain Alam, "Formal Verification of the xDAuth Protocol," IEEE, pp.1-14, 2016
43. Yong Yu, "Identity-based Remote Data Integrity is hacking with perfect data privacy-preserving for cloud storage," IEEE, pp. 1-11, 2016.
44. Xiaoing Jia, Efficient Revocable ID-based signature with cloud revocation server, IEEE Proceedings, Pp. 1-9, 2017
45. Carlos Eduardo Da Silva, Thomás Diniz, Nelio Cacho, and Rogério de Lemos' Self-adaptive authorization in OpenStack cloud platform, Journal of Internet Services and Applications, Journal of Internet Services and Applications (2018) 9:19, https://doi.org/10.1186/s13174-018-0090-7
46. Mell PM, Grance T. SP 800-145. The NIST Definition of Cloud Computing. Tech. Rep., National Institute of Standards and Technology. MD: Gaithersburg; 2011.
47. Duncan A, et al. Cloud Computing: Insider Attacks on Virtual Machines during Migration. In: Trust, Security, and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference https://doi.org/10.1109/TrustCom.2013.62
48. Garkoti G, Peddoju S, Balasubramanian R. Detection of Insider Attacks in Cloud-Based e-Healthcare Environment. In: Information Technology (ICIT), 2014 International Conference on; 2014. p. 195–200. https://doi.org/10. 1109/ICIT.2014.43.
49. Stolfo S, Salem M, Keromytis A. Fog Computing: Mitigating Insider Data Theft Attacks in the Cloud. In: Security and Privacy Workshops (SPW), 2012 IEEE Symposium on; 2012. p. 125–128. https://doi.org/10.1109/SPW.2012.19.
50. Cappelli DM, Moore AP, Trzeciak RF. The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crime, 1st ed. Addison-Wesley Professional; 2012.
51. Duncan A, Creese S, Goldsmith M. Insider Attacks in Cloud Computing. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on; 2012.p. 857–862. https://doi.org/ 10.1109/ TrustCom.2012.188.
52. Cole DE. Insider threats and the need for fast and directed responded. Tech. Rep.: SANS Institute InfoSec Reading Room; 2015.
53. Bailey C, Chadwick DW, de Lemos R. Self-adaptive federated authorization infrastructures. J Compute

System Sci. 2014; 80(5):935–52. https:// dx.doi.org/10.1016/j.jcss.2014.02.003. http://www.sciencedirect.com/ science/article/PII /S0022000014000154
54. Pasquale L, et al. Securitas: A Tool for Engineering Adaptive Security. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. FSE ’12. New York: ACM; 2012. p. 19:1–19:4. https://doi.org/10.1145/2393596.2393618. https:// doi.acm.org/ 10.1145/2393596.2393618
55. Schmerl B, et al. Architecture-based Self-protection: Composing and Reasoning About Denial-of-service Mitigations. In: Proceedings of the 2014 Symposium and Bootcamp on the Science of Security. HotSoS ’14.
56. Yuan E, Esfahani N, Malek S. A systematic survey of self-protecting software systems. ACM Trans Auton Adapt Syst. 2014; 8(4):17:1– https://doi.org/10.1145/2555611. http://doi.acm.org/10.1145/2555611.
57. De Lemos R, Giese H, Müller H, Shaw M, Andersson J, Litoiu M, Schmerl B, Tamura G, Villegas N, Vogel T, Weyns D, Baresi L, Becker B, Bencomo N, Brun Y, Cukic B, Desmarais R, Dustdar S, Engels G, Geihs K, GÃÂu″ schka K, Gorla A, Grassi V, Inverardi P, Karsai G, Kramer J, Lopes A, Magee J, Malek S, Mankovskii S, Mirandola R, Mylopoulos J, Nierstrasz O, Pezza M, Prehofer C, Schafer W, Schlichting R, Smith D, Sousa J, Tahvildari L, Wong K, Wuttke J. Software engineering for self-adaptive systems: A second research roadmap. In: de Lemos R, Giese H, Müller H, Shaw M, editors. Software Engineering for Self-Adaptive Systems II, Lecture Notes in Computer Science, vol 7475. Berlin: Springer; 2013. p. 1–32. https://doi. org/10.1007/978-3-642-35813-5_1.
58. Schultz E, A framework for understanding and predicting insider attacks. Computer Security. 2002;21(6):526–31. https://doi.org/10.1016/S0167-4048(02)01009-X.
59. Cappelli DM, Moore AP, Trzeciak RF. The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crime, 1st ed. Addison-Wesley Professional; 2012.
60. George Silowash AM, Cappelli D, et al. Common sense guide to mitigating insider threats. Tech. rep. CERT Carnegie Mellon; 2012.
61. Colwill C. Human factors in information security: The insider threat - who can you trust these days?. Inf Secur Tech Rep. 2009;14(4):186–96. https:// doi.org/10.1016/j.istr.2010.04.004.
62. Clercq JD. Single Sign-On Architectures. London: Springer-Verlag; 2002. p. 40–58. http://dl.acm.org/citation.cfm?id=647333.722879
63. Chadwick DW, et al. PERMIS: A Modular Authorization Infrastructure. Concurr Comput: Pract Exper. 2008;20(11):1341–57. https://doi.org/10. 1002/cpe.v20:11. http://dx.doi.org/10.1002/cpe.v20:11
64. Sandhu RS, et al. Role-Based Access Control Models. Computer. 1996; 29(2):38–47. https://doi.org/10.1109/2.485845. http://dx.doi.org/10. 1109/2.485845.
65. Hu VC, et al. SP 800-162. Guide to Attribute-Based Access Control (ABAC) Definitions and Considerations. Tech. Rep., National Institute of Standards and Technology. VA: McLean and Clifton; 2014.
66. Chadwick DW. Federated Identity Management. In: Foundations of Security Analysis and Design V, Lecture Notes in Computer Science, vol 5705. Berlin: Springer; 2009. p. 96–120. https://doi.org/10.1007/978-3- 642-03829-7_3.
67. Hu VC, et al. SP 800-162. Guide to Attribute-Based Access Control (ABAC) Definitions and Considerations. Tech. Rep., National Institute of Standards and Technology. VA: McLean and Clifton; 2014.
68. De Lemos R, Giese H, Müller H, Shaw M, Andersson J, Litoiu M, Schmerl B, Tamura G, Villegas N, Vogel T, Weyns D, Baresi L, Becker B, Bencomo N, Brun Y, Cukic B, Desmarais R, Dustdar S, Engels G, Geihs K, GÃÂu″ schka K, Gorla A, Grassi V, Inverardi P, Karsai G, Kramer J, Lopes A, Magee J, Malek S, Mankovskii S, Mirandola R, Mylopoulos J, Nierstrasz O, Pezza M, Prehofer C, Schafer W, Schlichting R, Smith D, Sousa J, Tahvildari L, Wong K, Wuttke J. Software engineering for self-adaptive systems: A second research roadmap. In: de Lemos R, Giese H, Müller H, Shaw M, editors. Software Engineering for Self-Adaptive Systems II, Lecture Notes in Computer Science, vol 7475. Berlin: Springer; 2013. p. 1–32. https://doi. org/10.1007/978-3-642-35813-5_1.
69. Kephart JO, Chess DM. The Vision of Autonomic Computing. IEEE Comput. 2003; 36(1):41–50. http://dx.doi.org/10.1109/MC.2003.1160055.
70. Yaping Chi; Gefei Li; Ying Chen, Xiaohong Fan, Design and Implementation of OpenStack Cloud Platform Identity Management Scheme, Published in 2018 International Conference on Computer, Information and Telecommunication Systems (CITS)
71. Feng Dengguo, Zhang Min, Zhang Yan, et al. Research on cloud computing security [J].Journal of Software, 2011,22 (1): 71-83.
72. Yu Nenghai, Hao Zhuo, Xu Jiajia, et al. Review of the progress of cloud security research [J].Journal of Electronics, 2013,41 (2): 371-381.
73. KHAN R H, YLITALO J, AHMED A S. OpenID Authentication as a Service in OpenStack[C]//IEEE. 7th International Conference on Information Assurance and Security, December 5-8, 2011, Malacca, Malaysia. New Jersey: IEEE, 2011: 372-377.
74. ANISETTI M, ARDAGNA C A, DAMIANI E, et al. Toward Security and Performance Certification of Open Stack[C]//IEEE. 2015 IEEE International Conference on Cloud Computing, June 27-July 2, 2015, New York, USA. New Jersey: IEEE, 2015: 564-571.
75. CUI Baojiang, XI Tao. Security Analysis of OpenStack Keystone[C]//IEEE. 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous

Computing, July 8-10, 2015, Santa Catarina, Brazil. New Jersey: IEEE, 2015: 283-288.

76. The Chi Yaping, Wang Huili, Yuan Ze Bo, and another authentication mechanism OpenStack Research and Improvement [J] Jilin University (Information Science), 2015 (11): 700-706.
77. JKR Sastry, M Trinath Basu, 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
78. JKR Sastry, M Trinath Basu, 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
79. M.Trinath Basu1, Dr. JKR Sastry, A fully security included Cloud Computing Architecture, International Journal of Engineering & Technology, Volume 7, Issue 2.7, Page 807-812, 2018
80. Dr. JKR Sastry, M Trinath Basu, Securing Multi-tenancy systems through user spaces defined within database level, Jour of Adv Research in Dynamical & Control Systems, Volume 10, issue 7, Page 405-412, 2018
81. 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
82. DBK Kamesh, JKR Sastry, 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
83. 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
84. NVS Pavan Kumar, Dr.JKR Sastry, 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
85. NVS Pavan Kumar, Dr.JKR Sastry, 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
86. NVS Pavan Kumar, Dr.JKR Sastry, 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