ISSN 2347 – 3983
Volume 8. No. 7, July 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter171872020.pdf
https://doi.org/10.30534/ijeter/2020/171872020

Implementing User defined Attribute and Policy based Access Control

Dr. JKR Sastry, B. TrinathBasu
[1]KoneruLakshmaiah Education Foundation, Vaddeswaram, India, drsastry@kluniversity.in
[2]KoneruLakshmaiah Education Foundation, Vaddeswaram, India, miriiyala68@kluniversity.in

ABSTRACT

Each component in OpenStack provides fine-grained control over the access of data and service through OpenStack component defined policies and Role-based access provided at the system level. OpenStack does not offer support for user-specific access control. Confidentiality of the data, as such, is left to the responsibility of the user. User has no way to define its policies to allow access to the data.

In this paper, a method is defined that can be implemented within OpenStack that allows users to set attributesand policy-based access control to data resources.

The method presents the components to be included inOpenStack and the way the additional components interact with native OpenStack components to affect the confidentiality and policy-based access control.

The components developed in Python language and the Native python programs interact with the user-defined python programs through the use of Restful API. The policies defined by the user are high-level security policies that are converted to low-level fine-grained security policies as defined by OpenStack. The users protect the data through attribute-based encryption and decryption

Key words: OpenStack, Attribute-based access control, User policy-based access control

1. INTRODUCTION

OpenStack is an Open source software that is used widely for building private clouds. Through the use of OpenStack, Infrastructure as service can be provided (IaaS), which includesthe provision of Virtual machines and storage services. Users can implement Platform as service (PaaS) and Software as services (SaaS) on top of IaaS. All the storage components such as Trove, Swift, Glance, and cinder provide exceptional grained access control services about which the user has no idea.

In an organization, functionaries and administrators have to exchange information on each other collaboratively at the individual level. Roles based access control defeats the concept of individual users authorizing access to the data owned by each other. A user defines their policies to provide access to their data to others. These policies are set at a higher level while the OpenStack system components still implement the fine-grained policies.

Thus there is a necessity to convert high-level user-defined policies to excellent grained policies defined by the System based components. There is a necessity to verify whether user-defined access control can be adapted through making calls to the System defined fine-grained policies. Adaption of user-defined policies helpsto share the users generally within the same organization and the users that work for different organizations.

OpenStack uses XACML for dealing with the System defined access policies. Affecting the access control that fits a policy is achieved by making XACML requests to an object owned policy server.The users can use a language to define its policies, which can then be directed as a message,and the same is converted to a fine-grained policy as adopted by the OpenStack fine-grained policy enforcement.

In this paper, a method is presented that allows the user to convey his policy in the JSON language. Then the same is converted by a Translator into systems defined policy enforcement in terms of ACML requests.OpenStack does not provide any functionality that allows the users to share the data with others. In a real organizational setup, this Kind of requirements exists on a day to basis.

2. PROBLEM DEFINITION

The main problem thus,the implementation of user-defined policies and user-defined access control,is integrated with System defined plans and access control.

3. RELATED WORK

3.1 Policy-Based Related work

Cloud computing systems have to address many challenges concerning Authorisation, Authentication, access control,

confidentiality, and integrity of the data even though the technologies are helping the businesses to derive several advantages for supporting cost-effective IT solutions [1].

Most of the solutions recommended in the literature focussed on introducingthe cryptography layer [2] [3] as an additional layer within a cloud computing system.

Domain-specific predicates have been proposed [4] for enforcing the integrity and confidentiality of the data. Cryptography is extensively used for enforcement of confidentiality ate storage layer and policy layer of the cloud computing system. The semantic gap between the secrecy between the storage layer and files is bridged through the use of cryptography.

Many authors addressed confidentiality issues through the use of fragmentation [5] or the combined use of cryptography and fragmentation [6]. The concept of splitting data into different fragments that are stored either in the plain or encrypted text is advocated extensively. Splitting is done in such a way the confidential information is not leaked while retrieving the partitions.

Several access models have been implemented in the literature, one of which is predominantly use is RBAC (Role-Based Access Control Model). The owner of the Data is allowed to store the data in an encrypted manner and also allows provided access rights to other users through the assignment of a specific Role, which allows access to particular data. [7].

It has been explained the way RBAC can be used to affectthreeessential principles related to confidentiality, which include the least privilege, data abstraction,and data separation. RBAC is found to be effective but still many changes are to be made to cloud computing systems to implement RBAC within cloud computing systems [8]

It has been proposed that the use of data-centric cryptosystems is the most appropriate method for ensuring the confidentiality of that Data. But the technique leads to many complications, which include the necessity of crucial management, Certificate Management, carrying actual encryption and decryption, distribution of the keys, etc. Many administrative tasks get added, which sometimes reflects on the response time within which user responses are provided. In contrast to this approach, the application of policies that affect access control is found to be a better policy. Polices provide different levels of access to different users. Encryption models cover just a set of users as per the distribution of keys while the policy-based model is global that satisfies many users who found to fit into the policies of the organizations

The owner of the data loses control to it once the Data is put on the Cloud. Users cannot enforce any kind of data access control. Some of the data cannot even delete the data as many data archiving methods work in behind within the Cloud for ensuring the safety [9[[10].

Many users developed Software that controls access to the data. Users applications developed to affect the access control to the data [11]. Fine-grained access polices attached to Data, and the same is effected based on the context in which the Data access. The association between the Data and polices recognized as annotations that are defined using a policy language. The association is also called a seal on the Data, which is sometimes perceived as a process of encrypting the data as per policies that apply to the users [12]. The policy enforcement processes proposed by the above-cited works, associate to every single Data a defined policy.

The policy is attached to every data element, making it complicated to manage such Kind of association. The size of the data will tremendously increase. It takes a lot of time to deal with and accessing such data. Further research addressed the attachment of the policies to a set of data to reduce the complexity of associating policies to the data. Polices are connected to the containers, partitions, user spaces, instances, etc. A combination of RBAC and Attribute-based access control model is implemented[13]

Many works have been presented for attribute-based access control to the OpenStack. The uses of ABAC has been studied considering different scenarios, which include cloud federation and federating identity management [14] [15].

The collocation between the tenants in a cloud under the IaaS platform has been studied [16]. A unified ABAC model has been presented that can be configured to effect discretional Role-based access control, and the way the model implemented within OpenStack has been shown [17]

A role centric and attributed based model (RABAC) presented [18] which XACML [19] language for building the model. XACML is a general-purpose accesscontrol policy language for managing access to resources. Many of the objects created within the cloud on-demand, especially the data objects. Data objects, as such, cannot be pre-identified. The objects, as such, cannot be predefined in advance so that policy association can be established.Thus there is a requirement to attach the polices to the objects dynamically and also preserve the users' requirements of enforcing the access control as the demand from time to time. Many contributions are made for enhancing the security with OpenStack system which all focussed at different aspects of security enhancement with OpenStack [20][21][22][23][24][25][26][27][28][29][30][31].

4. ACCESS CONTROL IMPLEMENTATION WITHIN OPENSTACK

4.1 Policy-Based Access implemented with OpenStack

Each service in OpenStack is operated based on policies designed for a specific function. The component "SWIFT"

deals with a certain set of policies, while the component like "Cinder" uses some other police. Polices are ruled that must be satisfied for allowing a user to carry an operation using a specific service

The cloud provider documents the required polices to be enforced by the services. The policies are designed such that any regulatory or legal requirements are taken care of. Once the polices are documents and written down as rules and stored, the same can be modified interactively at a later date. The regulations that represent the policiesdeveloped using the conditions and processes for doing a different Kind of operations and the Kind of objects dealt by specific services. Different kinds of operations that can be included in the rules include Enabling, disabling, creating, modifying, deleting, and assign privileges to the resources that a user can have. The policies are periodically reviewed, and modifications carried if required.

OpenStack provides access control only for fine-grained access to the services. They do not support user-defined policies 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 API access, 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 Compute service, 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.

In OpenStack, access control is defined based on the functional roles assigned to the users. The user-defined roles are converted into System defined roles for allocating to the same to the user and then use the System identified roles for affecting the access control.

OpenStack offers Access control based on the functional roles assigned to the users. The access control is enforced through the allocation of a system defined roles to the users. Thus there is a requirement of user-defined functional rules to System defined rules, achieved through the use of XACML language for translation.

Users sometimes need to define at run time, the other users with whom the data can be shared. Such Kinds of requirements are quite frequent.. It is the user who establishes other users who can be provided with privileges and permissions to share the data. The OpenStack does not meet such Kind of requirement. Thus there is a requirement to convert user-defined access polices to System defined access policies.

4.2 Access control enforcement method implemented within OpenStack

The access control model implemented in OpenStack deals seven distinct objects that include users, projects, roles, services, operations, and tokens. The element "Group" is also included to consider a set of users. In addition to theses, the concept of domain and Tenants is also used. A domain is a set of users that have access to an application or module. A tenant is a set of users who all have access to the same VM. An administrator manages domains and tenants

Users are persons authenticated to use resources such as projects, VMs, Storage, etc. Roles are elements that associate users with resources. The role-project pairs identified with the permissions which provide access rights to the users for accessing the services attached to the projects. Users with a specific Role attached to a project will have individual permissions to access the services and resources.

Besides, a different resource called object-type is also considered. Object-type refers to a set of objects of the same type, such as VMs, Images, files, storage units such as partitions, user spaces, instances, etc. Operations are also another kind of entity that defines different processes that can be used to access the objects. The service components use the operations to be carried on the objects.

Every user, after logging into OpenStack using user name and password through keystonegiven with a token which can be used to access specific resources as per the access rights of the user. Every Token designed to contain the information related to the user, the projects that can be obtained, and the roles that the user can play concerning the projects.

Roles and Permissions derived from the Tokens and the same are used for enforcing the access to the services and the objects accessed by those services.This Kind of model is called the Role-based Access Control method—the model used by OpenStackshown in Figure-1.

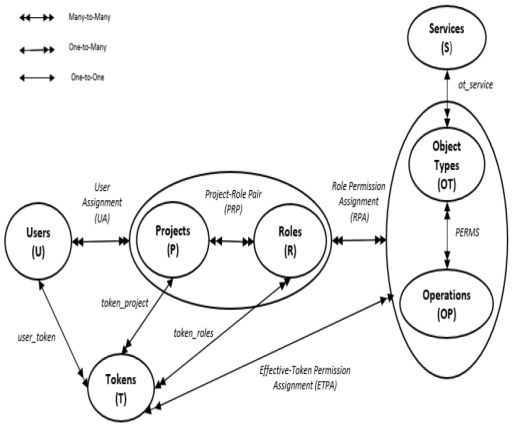

Figure 1: Role-based Access control model implemented by OpenStack

Keystone Module implements the access control system.

4.3 Drawbacks of Access control Mechanism within OpenStack

- Access Control is not implemented as per user choices, especially the issue of sharing information as per the decision of the user is not considered
- User-defined Polices also not considered. OpenStack implements only System defined policies

5. INVESTIGATIONS AND FINDINGS

5.1 Enhanced Architecture of OpenStack for achieving user-defined access control

Enhancements to the OpenStack architecture have been carried toaccommodate and implement the user-defined policies and sharing the data. The modified architecture shown in Figure 2.

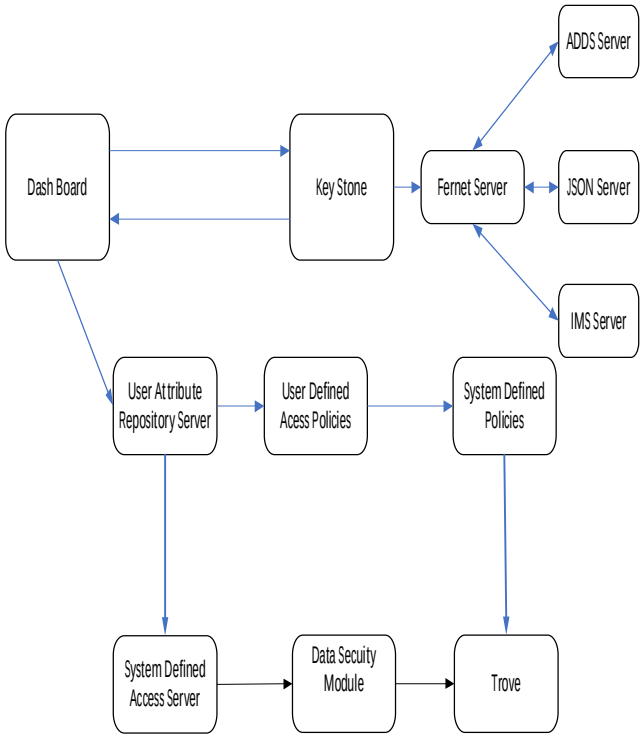

Figure 2: Enhanced Architecture for implementing user-defined aces control and data sharing

Users can initiate through DashBoard for either affecting a user-defined policy or for affecting the data sharing as per the user requirement. A python-based program is written to implement the translation of the user policies and Vice Versa. Users can also request for updating their attributes, which will be stored in a centralized repository. Users can also request sharing specific data with other users. The data sharing request is passed to the System defined Access server where the Public Key and Private Key structure generated and stored within the repository. The Public Key istransmitted to the user who shares the same with all those with whom the Initiating user would like to share the data. Every time a user wants to store the data, the same is encrypted using the user private key and the same submitted to the respective service for storing the same within the repositories related to the services.

5.2 Implementing user-defined user polices

OpenStack policies are defined using XACML Language. User-defined policies can be added to System defined Policies that can be used to modify the existing System defined policy. The user-defined policies in XML are converted to XACML language, and the same is executed to effect changes to the relating repository system defined polices.

The XACML policy is composed of four components:

- PEP (policy enforcement point) controls the data access.
- PDP (policy decision point) locates the many access rules (using the policy enforcement process), evaluates them to the access request, and returns a decision: deny or permit.
- PIP (policy information point) collects the missed information in access requestsconcerning the XACML syntax.
- PAP (policy administration point) permits to manage the rules (create, modify, or delete a rule)

User-defined policies represented using the XML language. The user request can be either an access request or an ACL request. In the case of ACL request, the XML recognizes it as a security policy rule and adds it to the XACML policy as a new rule via the PAD. However, in case of the access request, the XML translator sends the XACML request to the PEP that sends it to PDP for verification.

Three algorithms are written for converting XML request to XACML request, Translating XML request to XACML request, and then update the policy.json file and for translating XACML request response to XML request response.

Algorithm for translating an XML request into an XACML request

- Input: XML access request
- Output: XACML request

for all request do

Decompose XML request (request access)
return user, container, action, account

Mover user to sub
Move the container to res
Mover action to action

Move account to env

Transform XACML (sub, res, action, env)

return XACML request

Algorithm for converting XML request to XACML based rule

- Input: XML acl request
- Output: update the XACML policy for all ACL do

Decompose acl(acl)

if X container write

then

return user, container, account, write

Move user to sub
Mover container to res
Move account to env
Move to write to action

else

Move user to sub
Move the container to res
Move account to env
Move read to action

end if

if container.xml exists then

updatecontainer.xmladd policy (sub,res, action, env)

else

Createpolicy (container.xml)

end if

return container.xml

end for

Algorithm for converting XACL response to XML response

- Input: xacml access request
- Output: curl request

for all request do

decomposexacml request(request acess)
return sub, resource, action, env

Move user to sub
Move the container to res
Move action to action
Move env to account

transform curl (user,container,action)
return XACML request

end for

5.3 Implementing user-defined access control within OpenStack

Users own data of their own and want to share the data as they like with other users. The user-defined access must be based on the attributes of the user. Users can be clearly distinguished based on their characteristics. The characteristics of the users and user-defined access rights are stored in a Repository. A System component within OpenStack generated each user's private and public Key and shared with the users, and the user, in turn, shares the public with other users with whom the user wants to share the data. A Group is created using the users who are allowed to exchange the data through an issue of a system command, and a role is created, and Kind of operations that can be carried using the Role is created, and the Role is attached to the Group

When a request for storing thedata received from the user, the system component encrypts the data using the Private Key and sends across an application for storing the data to one of the services that deal with the data.

When a user requests theData having the required Role provided with the Data, decrypted using the public of the sender. The service deals with the validation of the key pair with the help of a certificate server run by it.

User-Defined access control Implementation scheme

- ❑ The users defined with an ID and password
- ❑ Users identified with a set of attributes having specific values. The attributes of the user stored in a repository within OpenStack
- ❑ A user can initiate a message that has the list of users with whom the information can be exchanged and the Kind of operations that can be carried by the users
- ❑ A separate user group is created with its members as the listed users
- ❑ A role is created, and the Kind of operations that can be carried using the roles are assigned to the user group
- ❑ Whenever a user wants to store the data within OpenStack, a service request is made to User-Access-Control Module newly developed and installed in the OpenStack

5.4 Experimentation and Results

Operational setup

Before Implementing the Command Language Interface carry the following operational command language sequences

1. Create User "ABC," User Group-ABC, MQY Role
2. Create user "XYZ" to User Group-ABC
3. Assign Create, Insert operations to MQY role
4. Assign MQY-Role to the User Group-ABC
5. User to Initiative a Request to create DB Table Test-Table having record Number (NUM (10) and data columns (X(120))
6. User "ABC" to Initiate a Request to create the record in Test-Table having the Data ("This is the Test Data")
7. User "XYZ" to initiate a Request for the record in Test-Table
8. The XYZ user is provided with the Data ("This is the Test Data"), thus providing the data for which the user has no rights

Users Initialisation Setup

1. Develop and Implement User-Access-Control service within OpenStack and include the facility into service list
2. Within Keystone Create Users, User Groups, Roles, and operations
3. Assign the user to New-user-group
4. Assign operations to Roles
5. Assign roles to New-user-group
6. Assign User-Access-Control service to New-user-group
7. Login to OpenStack system as administrator
8. Initiate attribute creation through the use of Token returned by Keystone System and requesting for User-Access-Control service
9. The user to Login to the OpenStack
10. User to initiate a request to create private Key and Private Key, based on attributes stored in the database and store the same using a Trove Database system. The user receives the key pair who distribute the public key to all the users with whom the user wants to share data.
11. The user initiates a user-defined policy in XML format, indicating the Kind of operations allowed by him by the users who are contained within the specified user group. The user-defined policies are converted into System defined polices and stored
12. The user policy is translated to System defined policy and stored
13. The user initiates a data related action by initiating a request to use-access-control. The access control system component encrypts the data using the private key of the user and Initiates the storage of the same through a data security module and TROVE database

Experimentation setup

1. User "ABC" to Initiate a Request to create a record in Test-Table having the Data ("This is the Test Data")
2. User "XYZ" to initiate a Request for the record in Test-Table
3. The XYZ user denied with the data. Gets an error as "Un-Authorised Access")

6. CONCLUSIONS

OpenStack implements fine-grained System defined polices for affecting the access control to the user data. Every user-defined with the functional responsibilities which are to be addressed through user roles, which are provided some access rights through the ability to make some operations concerning the resources provided in OpenStack.

Every user also works in tandem with others and therefore requires sharing the data as they desire. Also, sometimes even Role-based access is allowed. Some policies defined at the system level deny access to services and resources.

There is a need to consider the user-defined policies that can be converted to System driven policies so that twin objectives of enforcing the user-defined policies while at the same adhering to system-level policies can be achieved.

In this paper, user-defined policies are defined in XML, and translation mechanisms have been presented using which XML request responses are translated into XACML language, which is used for organizing the Access rules within the System defined access policies.

Again access control should be implemented as per the choices of the users. Users must decide with whom the data can be shared. Therefore there should be an additional layer within the OpenStack to facilitate user-defined access control.

REFERENCES

1. S. De Capitani di Vimercati, S. Foresti, and P. Samarati, "Managing and accessing data in the cloud: Privacy risks and approaches," in Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on, Oct 2012, pp. 1–9. https://doi.org/10.1109/CRISIS.2012.6378956
2. Vahldiek-Oberwagner, E. Elnikety, A. Mehta, D. Garg, P. Druschel, R. Rodrigues, J. Gehrke, and A. Post, "Guardat: Enforcing data policies at the storage layer," in Proceedings of the Tenth European Conference on Computer Systems, ser. EuroSys'15. New York, NY, USA: ACM, 2015, pp. 13:1– 13:16.

3. D. Sangeetha, V. Vijayakumar, V. Thirunavukkarasu, and A. Ramesh, "Enhanced security of phr system in cloud using prioritized level based encryption," in Recent Trends in Computer Networks and Distributed Systems Security, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2014. https://doi.org/10.1007/978-3-642-54525-2_5
4. S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Encryption-based policy enforcement for cloud storage," in Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on, June 2010, pp. 42–51.
5. P. Samarati and S. D. C. di Vimercati, "Data protection in outsourcing scenarios: Issues and directions," in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ser. ASIACCS'10. New York, NY, USA: ACM, 2010.
6. V. Cipriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Fragmentation and encryption to enforce privacy in data storage," in Computer Security ES- ORICS 2007, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 171–186.
7. L. Zhou, V. Varadharajan, and M. Hitchens, "Enforcing role-based access control for secure data storage in the cloud,"Comput. J. vol. 54, no. 10, pp. 1675–1687, Oct. 2011. https://doi.org/10.1093/comjnl/bxr080
8. W. Li, H. Wan, X. Ren, and S. Li, "A refined RBAC model for cloud computing," in Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on, May 2012, pp. 43–48.
9. M. Henze, R. Hummer, and K. Wehrle, "The cloud needs cross-layer data handling annotations," in Security and Privacy Workshops (SPW), 2013 IEEE, May 2013, pp. 18–22.
10. C. Cachin, K. Haralambiev, H.-C. Hsiao, and A. Sorniotti, "Policy-based secure deletion," in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 259–270.
11. C. Squicciarini, G. Petracca, and E. Bertino, "Adaptive data protection in distributed systems," in Proceedings of the Third ACM Conference on Data and Application Security and Privacy, ser. CODASPY'13. New York, NY, USA: ACM, 2013, pp. 365–376. https://doi.org/10.1145/2435349.2435401
12. N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, "Policy-sealed data: A new abstraction for building trusted cloud services," in Presented as part of the 21st USENIX Security Symposium (USENIX Security 12). Bellevue, WA: USENIX, 2012, pp. 175–188.
13. D. Kuhn, E. Coyne, and T. Weil, "Adding attributes to role-based access control," vol. 43, no. 6 June 2010, pp. 79–81.
14. D. W., Chadwick, K. Siu, C. Lee, Y. Fouillat, and D. Germonville, "Adding federated identity management to OpenStack," Journal of Grid Computing, vol. 12, no. 1, pp. 3–27, 2014.
15. C. A. Lee and N. Desai, "Approaches for virtual organization support in OpenStack," in IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2014, pp. 432–438.
16. X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model are covering DAC, MAC and RBAC," in IFIP Annual Conference on Data and Applications Security and Privacy. Springer, 2012, pp. 41–55.
17. X. Jin, R. Krishnan, and R. Sandhu, "Role and attribute-based collaborative administration of intra-tenant cloud IaaS," in IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014, pp. 261–274.
18. X. Jin, R. Sandhu, and R. Krishnan, "RABAC: role-centric attribute-based access control," in International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, 2012, pp. 84–96 https://doi.org/10.1007/978-3-642-33704-8_8
19. "XACML." [Online]. Available: https://en.wikipedia.org/wiki/XACML
20. 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
21. M. TrinathBasu, JKRSastry, Improving the OpenStack Authentication system through federation with JASON Tokens, International Journal of Advanced Trends in Computer Science and Engineering, Volume 8, Issue 6, Pages 3596-3614,2019 https://doi.org/10.30534/ijatcse/2019/143862019
22. TrinathBasu, JKRSastry, Strengthening Authentication within OpenStack Cloud Computing System through Federation with ADDS System, International Journal of Emerging Trends in Engineering Research, Volume 8, No. 1, Page, 213-238, 2020 https://doi.org/10.30534/ijeter/2020/29812020
23. JKRSastry, M TrinathBasu, Multi-Factor Authentication through Integration with IMS System, International Journal of Emerging Trends in Engineering Research, Volume 8, No. 1, Page, 88-113, 2020
24. 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
25. 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 https://doi.org/10.11591/ijece.v6i4.10676
26. S. L. SUSHMITHA, Dr. D. B. K. JKRSASTRY, 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

27. 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
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. https://doi.org/10.11591/ijece.v9i2.pp1385-1392
29. 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 https://doi.org/10.11591/ijeecs.v13.i1.pp65-71
30. M TrinathBasu, JKRSastry, Enhancing Data Security under Multi-Tenancy within OpenStack, International Journal of Advanced Trends in Computer Science and Engineering, Volume 9, Issue 1, 2020, pp .533-544
31. Dr.JKRSastry, M. TrinathBasu, Enhancement of Security within OpenStack – Some measures, International Journal of Emerging Trends and Engineering Research, Volume 8, Issue 3, 2020, pp. 919-938