**Abstract:**

OpenStack is a cloud Software to manage virtual infrastructures (virtual CPU, virtual memory virtual storage and so on) of ‘*Infrastructure as a service*’ Cloud. Although there are several cloud infrastructure management software available (ex. … ), OpenStack gets immense attention from the industries due to its robustness in functionality and security and nature of being open source. Although, there exists no outcry against OpenStack security, we intend to experiment security features of openstack. Openstack, Being a very large platform, turns it hard to experiment all of its security issues. To create an scope for our project, we investigated the ‘keystone’ module of openstack which is responsible for Authentication between Cloud customer and different modules of openstack. …

**Introduction:**

Cloud computing is a relatively novel topic in Information Technology that attracts significant attention from public and private sectors nowadays. Cloud computing is characterized by the following essential features as suggested by NIST– On-demand self-service, Broad network access, Resource pooling, Rapid elasticity and Measured service[1]. To consumers cloud computing provides different service capabilities in pay as you go strategy which vary from following service models - Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

Openstack is a framework to manage ‘Infrastructure as a Service’ Cloud which provides virtual infrastructure such as virtual CPU, virtual Storage, Virtual Network and so on to its consumers. Figure 1 shows a very abstract level view of the openstack framework which shows how openstack lies on top of standard hardware and create shared services to be used by the virtual compute, network and storage services.

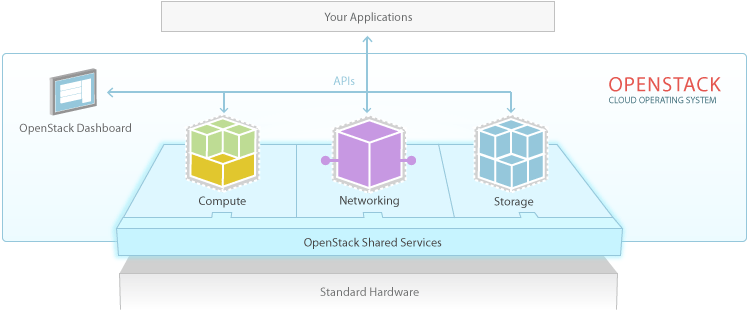


Fig1:

The robustness of OpenStack comes from its modularized and distributed structure which its adoptability. Figure 2 shows different modules of openstack. A brief description of each of the module is given below.

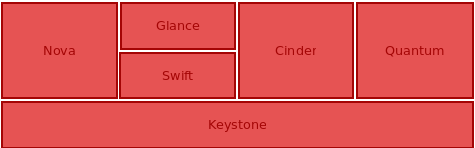


Figure 2:

**Nova:**

Nova is the project name for OpenStack Compute, a cloud computing fabric controller, the main part of an IaaS system. Individuals and organizations can use Nova to host and manage their own cloud computing systems. [2] The main purpose of Nova is to perform actual computation associated with any service request.

**Glance:**

The Glance module provides services for discovering, registering, and retrieving virtual machine images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image.[3]

**Swift:**

The OpenStack Object Store project, known as Swift, offers cloud storage software so that you can store and retrieve lots of data in virtual containers. It's based on the Cloud Files offering from Rackspace.[4]

**Cinder:**

Cinder provides an infrastructure for managing volumes in OpenStack. It was originally a Nova component called nova-volume, but has become an independent project since the Folsom release.[5]

Quantum:

Neutron is an [OpenStack](https://wiki.openstack.org/) project to provide "networking as a service" between interface devices (e.g., vNICs) managed by other Openstack services (e.g., nova).[6]

**Keystone:**

Keystone is the openstack’s authentication service. It provides Identity, Token, Catalog and Policy services for use specifically by projects in the OpenStack family. Figure 3 shows that how keystone module sits between Cloud Customer and other Openstack modules.

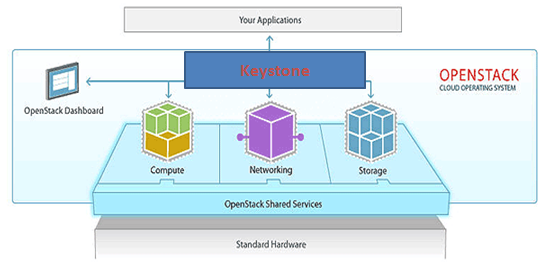
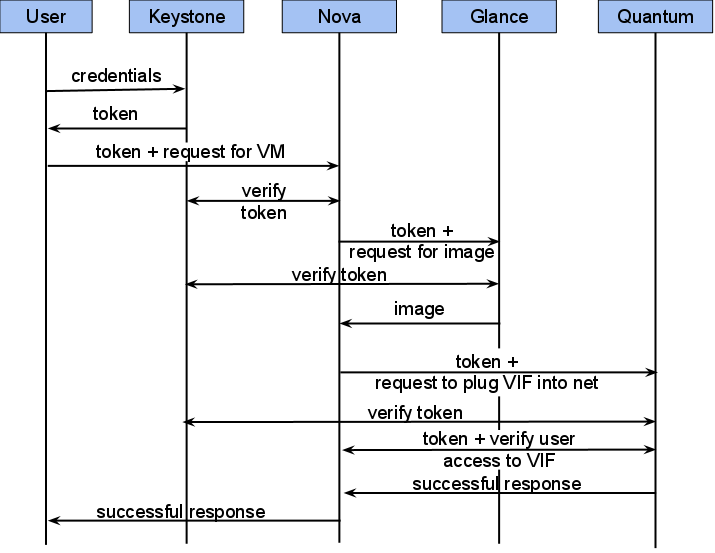


Figure 3: Keystone as an authentication between Openstack Service and Consumer Application.

Keystone is a significant module in openstack in the way that

1. Every communication between a openstack module (service provider) and service consumer is intervened by keystone. Keystone verifies the identity of the requester (role, location and so on) and based on the verification it provides the requester a token which is further used to perform subsequent operation by the requester. Figure 4 shows how keystone generates a token for request and subsequently verifies it upon being received from other services of openstack.
2. As token generation is essential part of every valid communication, unavabilability or compromise of keystone module, turns the whole openstack unusable.

****

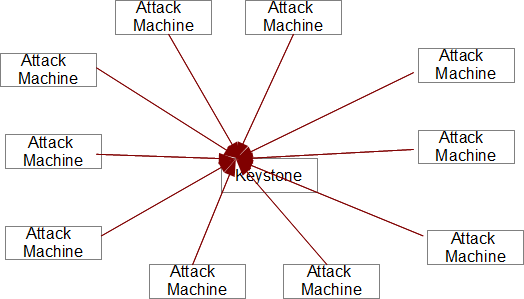
**Figure 4:**

**Experiment:**

In the project we set up two different experiments first of which is to experience keystone’s ability to tolerate Denial of Service attack and the other to check the randomness of generated token by the keystone service.

Experiment 1 - Resiliency of Keystone on DDOS Attack

In this experiment we have a set up an DDoS attack environment with 10 VMs at UTSA ICS Cloud. The keystone service in running on a different VM machine in the same cloud. Figure 5 depicts our attack environment. Table 1 shows the configuration of each machine**.**

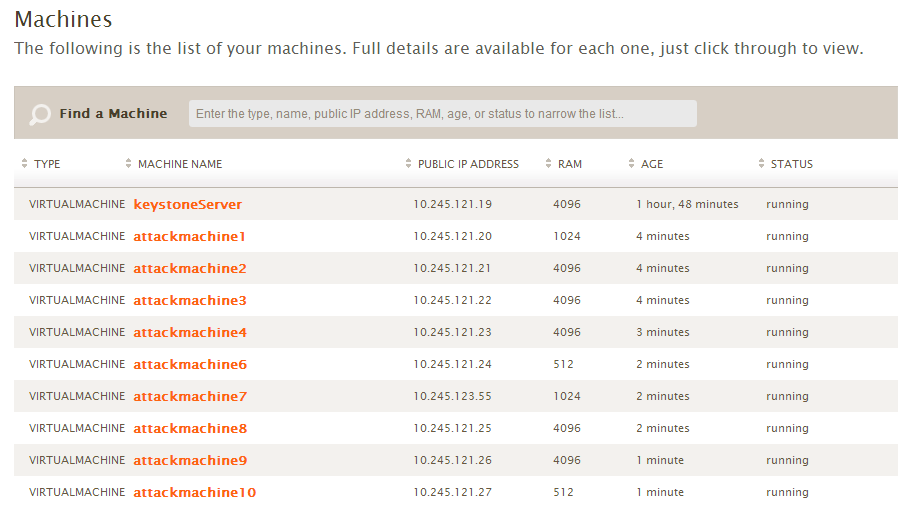
****

**Fig 5a:**

|  |  |  |
| --- | --- | --- |
| Machine Type | Virtual CPU | Virtual Memory |
| Keystone Server | **1 Conf: ?** | **4 GB** |
| Attack Machine | **1 Conf:** | **4 GB** |

**Table 1:**

**Figure 5b shows the created VMs at UTSA’s ICS Cloud.**

****

**Fig 5b: Virtual Machine on ICS Cloud for our attack configuration.**

**Experiment Result:** We have monitored two different performance on the CPU utilization of the keystone server. In the first case, we have measured the surge of CPU usage of the keystone process alone. Fig 6. Shows the rise the CPU usage of the keystone process as we run the attack script from the attack machine. As we can see CPU usages of the process increases and remain above 50% usage as the attack continues.

**Fig 6: CPU Usage of the Keystone Process (Attack is shown during the duration of 40s to 250s)**

In the second case we measure the overall cpu usage of the system. Fig 7. Shows overall cpu utilization of the system under dos attack. As we can see, even if the CPU utilization of the keystone process goes beyond 50%, over all CPU utilization of the machine running VM remains close to 10%.

**Fig 7: CPU Usage of the Server machine (Attack is shown during the duration of 40s to 250s)**

**Technical discussion of the Experiment:**

In this section, we describe the design of our DoS attack framework. We have two different script in our design (shown in fig 8a) – one of which is a python script (shown in Fig 8b) which creates 1000 theads in each attack VM and each thread generates continuous attack requests. The ‘Attack Script’ is attached as appendix with this report.

****

Fig 8a: DoS Attack Design Fig 8b: Attack Script anatomy

We have another script running on the keystone server which monitors the overall cpu utilization of the server and cpu utilization of the keystone process. For measuring overall performance of the server we have used ‘sar’ utility command and for measuring process specific cpu utilization we used ‘pidstat’ of ‘sysstat’ utility suite[7].

**Result Evaluation:**

We have experienced that when all attack vms run attack script, keystone service becomes significantly busy with cpu utilization of > 50%. On the other hand, overall CPU utilization of the server machine jump from 2% of average use to >10% of usage which is not significantly large but yet notable. We assume that more sophisticated DoS attack with variation of attack request can make the keystone service more occupied resulting more exhaustion of overall CPU usage of the server machine.

**Experiment 2:**

**References:**

1. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
2. <http://docs.openstack.org/developer/nova/>
3. <http://docs.openstack.org/developer/glance/>
4. <http://docs.openstack.org/developer/swift/>
5. <https://wiki.openstack.org/wiki/Cinder>
6. <http://www.openstack.org/software/openstack-networking/>
7. <http://sebastien.godard.pagesperso-orange.fr/>