Concordia Institute for Information System Engineering
(CIISE)

INSE 6620
Cloud Computing Security and Privacy

Project Report

Performing and detecting a security attack on a virtual network built using OpenStack.

Submitted to:
Prof. Lingyu Wang

Student ID	Name	Contribution
40001711	Akida Abdaleh	Attack Detection & Evasion
40232485	Chris Regy Vallikunnathu	Installation & Configuration
40235192	Enamul Karim Tamzid	Installation & Configuration
40240114	Fariha Noor	Network Attack Simulation
40232489	Harleen Kaur	Installation & Configuration
40185647	Mahmood Dhabi	Network Attack Simulation
40257131	Mukul Prabhu	Network Attack Simulation
40255233	Pritom Samanta Picklu	Network Attack Simulation
26447813	S M Harun-Or Rashid	Installation & Configuration

TABLE OF CONTENT

1. Introduction 3

2. Implementation 3

   2.1 Installation of OpenStack using DevStack 3

   2.2 Network Setup 5

   2.3 Keypair Creation 6

   2.4 Security Group 6

3. Network Attack and Detection 9

   3.1 Proof of creation of private network and compute instances 11

   3.2 The Attack 11

4. Challenges faced during installation 12

5. Reference 13

1. Introduction

The rapid growth of cloud computing is revolutionizing the information technology industry. In simple terms, cloud computing offers seamless delivery of in-demand computing services including, but not limited to, servers, storage, databases, networking and software over the internet. The unimpeded rise of cloud computing platforms brings forth a new set of security challenges that need to be addressed on a priority basis. Cloud computing technologies force the introduction of new and innovative security techniques which are drastically different from the ones being used in traditional computing systems. As a result, cloud security is at the forefront of modern-day research.

Through this project, we aim to gain a deeper practical understanding of cloud computing security by launching a virtual network-based attack using OpenStack, a widely popular open-source cloud computing infrastructure software project. The implementation of this project can be broadly categorized into three major phases:

i. Installation & Configuration of OpenStack Infrastructure and Required Computing Resources
ii. Network Attack Simulation
iii. Detection & Evasion of Attack Using Snort

The details of the implementation of the above phases are presented in the following sections. We also present the techniques used, challenges faced and our learnings through the course of this project.

2. Implementation

Our team performed the installation of the OpenStack platform on virtual machines. For the creation of virtual machines, we used Oracle VM VirtualBox. The OS image used to set up the virtual machine is Ubuntu 22.04.3 LTS (Jammy Jellyfish).

Several deployment options are available to achieve seamless installation of the OpenStack platform. Some of the popular options include DevStack, PackStack (only RHEL & CentOS), MicroStack and Sunbeam. As DevStack is the most popular option among developers and testers to automatically spin up clouds in the OpenStack CI environment, we proceeded with this deployment option. DevStack is a series of extensible scripts used to quickly bring up a complete OpenStack environment. The steps involved are detailed as follows.

2.1 Installation of OpenStack using DevStack

Firstly, we ensured that the system was up-to-date. Then, we created a new user called “stack” using the following command. This user is a non-root user with sudo privileges.

```
sudo adduser -s /bin/bash -d /opt/stack -m stack
```

To enable root privileges for the “stack” user and allow it to run without password, we used the following command:

```
echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
```

As the stack user, we then cloned devstack from the git repository onto the system using the following command:

```
git clone https://opendev.org/openstack/devstack
```

Next, we made the necessary changes to the `local.conf` file and ran the stack.sh script to install OpenStack. Below are screenshots of the `local.conf` file and “installation complete” message from our machine:

```
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
~
```

```
This is your host IP address: 172.31.3.136
This is your host IPv6 address: ::1
Horizon is now available at http://172.31.3.136/dashboard
Keystone is serving at http://172.31.3.136/identity/
The default users are: admin and demo
The password: secret

Services are running under systemd unit files.
For more information see:
https://docs.openstack.org/devstack/latest/systemd.html

DevStack Version: 2023.2
Change: 4c45bec6ebb965202d8d7d7832c093f47ecc2910 GLOBAL_VENV: add more binaries 2023-08-12 11:35:08 +0200
OS Version: Ubuntu 22.04 jammy

2023-08-14 13:29:27.107 | stack.sh completed in 1182 seconds.
```

The DevStack script has installed the following critical components and services to the OpenStack environment:

• **Nova** is a critical service that facilitates the creation of compute instances, i.e., virtual machines, which will be actively used in this project.
• **Neutron** offers Network as a Service (NaaS) enabling network connectivity across the virtual environment.
• **Swift** is a fault-tolerant object storage service that stores and retrieves unstructured data objects using a RESTful API.
• **Cinder** offers persistent data storage that is made available through a self-service API, which enables users to utilize storage resources without being bothered about their types or location. The block devices in OpenStack are known as Cinder Volumes.

• **Keystone** is the OpenStack service that is responsible for identity and authentication of all OpenStack services.
• **Glance** is an image service used to store and retrieve virtual machine and container images.

2.2 Network Setup

Before proceeding to set up the required compute instances, we first configured a virtual network so that the instances can communicate with each other and the internet. The provider (public) network is available by default upon installation of OpenStack. This public network enables connectivity to the internet. For internal communication between the two compute instances, we created a self-service network that is then connected to the public network through a router. The steps we followed are detailed below:

We created a network named “net1” using the following command:

```
network create net1
```

Then, we created a subnet under this network with the following command:

```
subnet create --network net1 --dns-nameserver 8.8.4.4 --gateway 172.16.1.1 --
subnet-range 172.16.1.0/24 <SUBNET_NAME>
```

To view the list of existing networks and subnets, we can use the `network list` command.

```
(openstack) network list
+--------------------------------------+---------+----------------------------------------------------------------------------+
| ID                                   | Name    | Subnets                                                                    |
+--------------------------------------+---------+----------------------------------------------------------------------------+
| 355f87be-ce09-4848-9c21-84291a035a1b | shared  | 1f111202-9bb9-4b2c-a6c3-08cd5a9a8650                                       |
| 378d2dfe-dc8a-4ef5-a728-376c4f212a34 | public  | 3163ade7-97ff-4d78-bd41-0a5a4659d60d, 40eb92bf-a3c4-4f30-a5a4-723a2ddaa399 |
| 67d8f04a-daba-4718-8428-c4152694742f | private | 3ba096a6-2f8d-4a4e-972e-29135df8a8ba, 5f7601a1-3fac-4a85-ad1f-43d0db41b9d4 |
| 6d30c6d6-bed1-49ef-b6d9-9af1072643f0 | net1    | 68e082f5-6b3f-486e-8372-428b3c8c30cf                                       |
+--------------------------------------+---------+----------------------------------------------------------------------------+
```

Next, we created a router named “router2” and connected it to the subnet created in the previous step using the following commands:

```
router create router2
```

```
router add subnet router2 <subnet_name>
```

Further, in order to enable internet connectivity on the self-service network, we connected the other end of the router to the default provider (public) network using the following command:

```
router set router2 --external-gateway public
```

To view the list of routers installed, we can use the `router list` command.

```
+--------------------------------------+---------+--------+-------+----------------------------------+
| ID                                   | Name    | Status | State | Project                          |
+--------------------------------------+---------+--------+-------+----------------------------------+
| 04a74ddc-16a7-442f-b5c6-be8330c33f86 | router2 | ACTIVE | UP    | 42256cda67f149eb8faada03ab25a2ba |
| b56eae33-cb99-4def-845e-dc0040204b72 | router1 | ACTIVE | UP    | 42256cda67f149eb8faada03ab25a2ba |
+--------------------------------------+---------+--------+-------+----------------------------------+
```

The required network setup and configuration is now complete.

2.3 Keypair Creation

Before setting up the compute instances, we need to generate a keypair that will be used to access the instances. The keypair is the public key of an OpenSSH key pair to be used for access to created servers.

We created a new keypair named “mykey” using the following command:

```
keypair create –private-key mykey
```

To view the list of keypairs, we use the `keypair list` command.

```
(openstack) keypair list
+-------+-------------------------------------------------+------+
| Name  | Fingerprint                                     | Type |
+-------+-------------------------------------------------+------+
| mykey | b2:34:aa:64:f3:4e:4d:cf:cd:84:21:1d:f5:03:b7:13 | ssh  |
+-------+-------------------------------------------------+------+
```

2.4 Security Group

Next, we proceeded to set up a security group. A security group allows us to define network access rules that can be used to limit the types of traffic that have access to instances. The default security group denies all incoming traffic and allows only outgoing traffic to the instance. Hence, we created a new security group named “testsec” that allows inbound TCP, UDP, ICMP access with the following commands.

```
security group create testsec

security group rule create --proto tcp --dst-port 1:65525 testsec

security group rule create --proto udp --dst-port 1:65525 testsec

security group rule create --proto icmp testsec
```

The details of the newly created security group are as displayed through the following screenshot:

```
| name           | testsec
                            |
| project_id     | 547fac6a93f2438da212a4addee8b06c
                            |
| revision_number | 4
                            |
| rules          | created_at='2023-08-14T22:54:15Z', direction='ingress', ethertype='IPv4', id='28de65ff-d53f-4e43-bbe9-771fa6bf78c8', norm
alized_cidr='0.0.0.0/0', port_range_max='65525', port_range_min='1', protocol='tcp', remote_ip_prefix='0.0.0.0/0', standard_attr_id='52', upd
ated_at='2023-08-14T22:54:15Z' |
|                | created_at='2023-08-14T22:55:34Z', direction='ingress', ethertype='IPv4', id='4f030521-6466-4981-b340-3ad2157c348c', norm
alized_cidr='0.0.0.0/0', protocol='icmp', remote_ip_prefix='0.0.0.0/0', standard_attr_id='54', updated_at='2023-08-14T22:55:34Z'
                            |
|                | created_at='2023-08-14T22:53:32Z', direction='egress', ethertype='IPv4', id='62fc371e-5f12-41af-82bb-2890b4907a20', stand
ard_attr_id='50', updated_at='2023-08-14T22:53:32Z'
                            |
|                | created_at='2023-08-14T22:54:58Z', direction='ingress', ethertype='IPv4', id='8725b900-07f0-4b73-93ef-fea47e2d82ad', norm
alized_cidr='0.0.0.0/0', port_range_max='65525', port_range_min='1', protocol='udp', remote_ip_prefix='0.0.0.0/0', standard_attr_id='53', upd
ated_at='2023-08-14T22:54:58Z' |
|                | created_at='2023-08-14T22:53:32Z', direction='egress', ethertype='IPv6', id='e8e4bd4c-d56d-4287-ad6b-c44151d4b596', stand
ard_attr_id='51', updated_at='2023-08-14T22:53:32Z'
```

2.5 Launching Compute Instances

The final step in the installation process was the creation of the required compute instances. The default operating system available for instances created in OpenStack is CirrOS. However, this OS distribution has minimal capabilities which were not sufficient for our operations. Hence, we uploaded an Ubuntu 22.04.3 (Jammy Jellyfish) image to be used to compute instances. The image was uploaded using the following command:

```
openstack image create --disk-format qcow2 --container-format bare --public -
-file /home/ubuntu/jammy-server-cloudimg-amd64-disk-kvm.img ubuntu-image
```

The list of images available can be retrieved using the `image list` command.

```
(openstack) image list
+--------------------------------------+--------------------------+--------+
| ID                                   | Name                     | Status |
+--------------------------------------+--------------------------+--------+
| 907bae3a-265b-4296-aa87-41e9bcd1d54d | centos63-image           | active |
| 73f7ec5e-1674-4952-a853-1a4ccc21824b | cirros-0.6.2-x86_64-disk | active |
| 05e7c11e-2bfe-4a3f-825d-0abcb79fd38e | ubuntu                   | active |
+--------------------------------------+--------------------------+--------+
```

Once the required image was uploaded, we were finally ready to spin up the compute instances. OpenStack offers many different flavors of compute instances. Flavors define the compute, memory, and storage capacity of nova computing instances. The different flavors available on OpenStack are as follows:

```
(openstack) flavor list
+----+-----------+-------+------+-----------+-------+-----------+
| ID | Name      |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+----+-----------+-------+------+-----------+-------+-----------+
| 1  | m1.tiny   |   512 |    1 |         0 |     1 | True      |
| 2  | m1.small  |  2048 |   20 |         0 |     1 | True      |
| 3  | m1.medium |  4096 |   40 |         0 |     2 | True      |
| 4  | m1.large  |  8192 |   80 |         0 |     4 | True      |
| 42 | m1.nano   |   128 |    1 |         0 |     1 | True      |
| 5  | m1.xlarge | 16384 |  160 |         0 |     8 | True      |
| 84 | m1.micro  |   192 |    1 |         0 |     1 | True      |
| c1 | cirros256 |   256 |    1 |         0 |     1 | True      |
| d1 | ds512M    |   512 |    5 |         0 |     1 | True      |
| d2 | ds1G      |  1024 |   10 |         0 |     1 | True      |
| d3 | ds2G      |  2048 |   10 |         0 |     2 | True      |
| d4 | ds4G      |  4096 |   20 |         0 |     4 | True      |
+----+-----------+-------+------+-----------+-------+-----------+
```

Out of the above available flavors, our compute instances were created using the ds1g flavor. We used the following command to launch two compute instances, named "testserver" and "testserver2", and attach the instances to pre-created network, security group and keypair.

```
server create --flavor ds1G --image 5edd7027-8fc8-4e9e-8c47-e1007633825a --nic net-id=47a051d3-8b6c-4fd6-b616-cb9df870d451 --security-group testsec --key-name mykey selfservice-instance
```

The list of instances created can be viewed using the server list command.

```
(openstack) server list
+--------------------------------------+----------------------+--------+-----------------------------------------+-------------------------+
---------+
| ID                                   | Name                 | Status | Networks                                | Image                   |
 Flavor  |
+--------------------------------------+----------------------+--------+-----------------------------------------+-------------------------+
---------+
| 0f2dc5af-d214-4429-93de-7d338be199cc | selfservice-instance2 | ACTIVE | selfservice=172.16.1.127, 172.24.4.241 | ubuntu                  |
 ds1G    |
| c5f325ab-0b55-4a23-bdf7-c0ce37a4f214 | selfservice-instance4 | ACTIVE | selfservice=172.16.1.208, 172.24.4.42  | ubuntu                  |
 ds1G    |
| 01780294-bfba-4e9c-80a5-9af00955b9b6 | selfservice-instance  | ACTIVE | selfservice=172.16.1.128, 172.24.4.110 | cirros-0.6.2-x86_64-disk |
 m1.tiny |
+--------------------------------------+----------------------+--------+-----------------------------------------+-------------------------+
---------+
(openstack)
```

Following is a screenshot from the compute instance that was created:

```
  Usage of /:   11.9% of 9.51GB   Users logged in:     0
  Memory usage: 11%               IP address for ens3: 172.16.1.20
  Swap usage:   0%

Expanded Security Maintenance for Infrastructure is not enabled.

0 updates can be applied immediately.

Enable ESM Infra to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

With this, the installation of OpenStack and the creation of the network and compute instances are now complete. In the upcoming sections, we detail the steps involved in simulating the network attack and detecting it using Snort.

3. Network Attack and Detection

We have created two ‘Ubuntu’ instances, one for attacking and another for detecting the attack using ‘Snort’. Here selfservice-instance2 and is respectively the target and attacker instances. We have associated floating ips to both of these instances so that we can make ssh connection from outside of the private network.

```
(openstack) server list
+--------------------------------------+-----------------------+--------+----------------------------------------+--------------------------+
---------+
| ID                                   | Name                  | Status | Networks                               | Image                    |
 Flavor  |
+--------------------------------------+-----------------------+--------+----------------------------------------+--------------------------+
---------+
| 0f2dc5af-d214-4429-93de-7d338be199cc | selfservice-instance2 | ACTIVE | selfservice=172.16.1.127, 172.24.4.241 | ubuntu                   |
 ds1G    |
| c5f325ab-0b55-4a23-bdf7-c0ce37a4f214 | selfservice-instance4 | ACTIVE | selfservice=172.16.1.208, 172.24.4.42  | ubuntu                   |
 ds1G    |
| 01780294-bfba-4e9c-80a5-9af00955b9b6 | selfservice-instance  | ACTIVE | selfservice=172.16.1.128, 172.24.4.110 | cirros-0.6.2-x86_64-disk |
 m1.tiny |
+--------------------------------------+-----------------------+--------+----------------------------------------+--------------------------+
---------+
(openstack)
```

Attacker Instance (selfservice-instance4, 172.24.4.42**)**

```
ubuntu@selfservice-instance4:~$ ifconfig
ens3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1442
        inet 172.16.1.208  netmask 255.255.255.0  broadcast 172.16.1.255
        inet6 fe80::f816:3eff:fe7d:6ee7  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:7d:6e:e7  txqueuelen 1000  (Ethernet)
        RX packets 732622  bytes 420491642 (420.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1628842  bytes 259680427 (259.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2922  bytes 217028 (217.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2922  bytes 217028 (217.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@selfservice-instance4:~$
```

Target Instance (selfservice-instance2, 172.24.4.241**)**

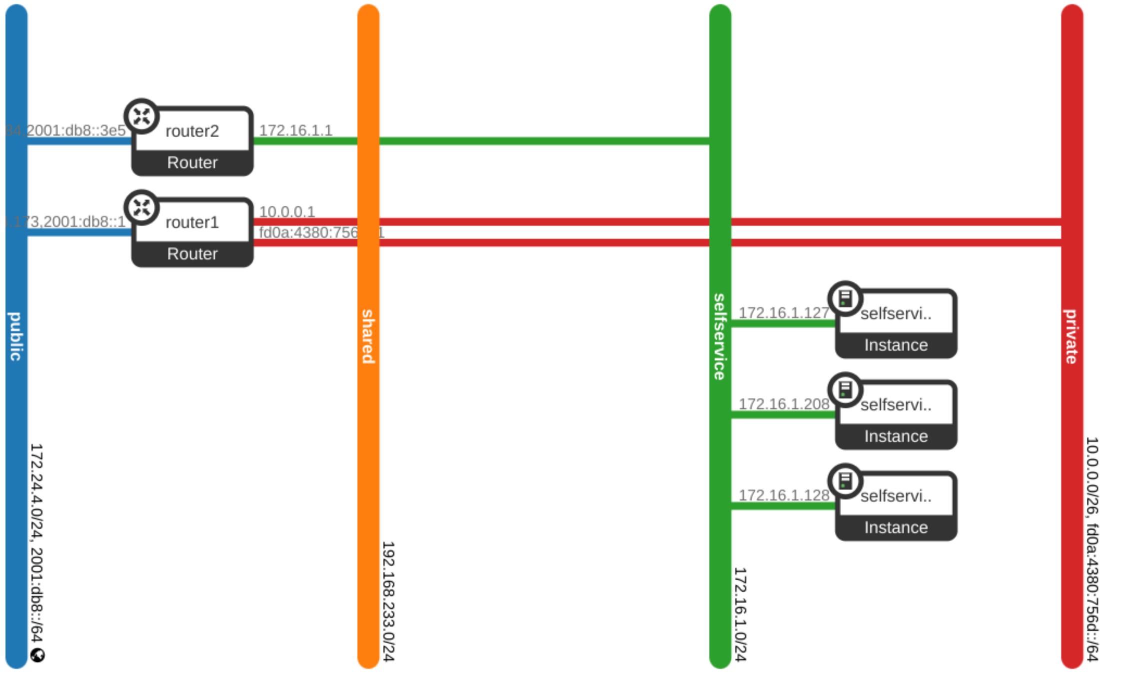
```
ubuntu@selfservice-instance2:/etc/snort/rules$ ifconfig
ens3: flags=4419<UP,BROADCAST,RUNNING,PROMISC,MULTICAST>  mtu 1442
        inet 172.16.1.127  netmask 255.255.255.0  broadcast 172.16.1.255
        inet6 fe80::f816:3eff:fec6:9871  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:c6:98:71  txqueuelen 1000  (Ethernet)
        RX packets 369299  bytes 292886805 (292.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 411508  bytes 218916144 (218.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 502  bytes 42766 (42.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 502  bytes 42766 (42.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@selfservice-instance2:/etc/snort/rules$
```

3.1 Proof of creation of private network and compute instances

Here we have created a private network called ‘selfservice’ which is connected to ‘router2’ and the router is connected to the external(public) network. Inside the ‘selfservice’ network we have our two ubuntu instances (172.16.1.127 & 172.16.1.208)



3-

3.2 The Attack

For the attack we have prepared a small script to simulate a flood attack on our target instance. On the target instance we have installed Snort to detect intrusion on our attack. We have a rule about detecting such network intrusion. The first image is from our Attack Instance where our attack script is present. In the Target Instance we have snort running in Alert mode, it then prints on the console once the rule is broken.

```
ubuntu@selfservice-instance4:~$
ubuntu@selfservice-instance4:~$
ubuntu@selfservice-instance4:~$ sudo python3 attack.py
Attacking...
0.12251877784729004 ms
Attacking...
0.0051000118255615234 ms
Attacking...
0.004021644592285156 ms
Attacking...
0.005507469177246094 ms
Attacking...
0.0041272640228271484 ms
```

```
08/16-01:18:42.058030  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.058030  [**] [1:384:5] ICMP PING [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.058653  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.16.1.127 -> 172.24.4.42
08/16-01:18:42.058653  [**] [1:408:5] ICMP Echo Reply [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.16.1.127 -> 172.24.4.42
08/16-01:18:42.071030  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.071030  [**] [1:384:5] ICMP PING [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.071570  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.16.1.127 -> 172.24.4.42
08/16-01:18:42.071570  [**] [1:408:5] ICMP Echo Reply [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.16.1.127 -> 172.24.4.42
08/16-01:18:42.084280  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.084280  [**] [1:384:5] ICMP PING [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.084813  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.16.1.127 -> 172.24.4.42
08/16-01:18:42.084813  [**] [1:408:5] ICMP Echo Reply [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.16.1.127 -> 172.24.4.42
08/16-01:18:42.099219  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.099219  [**] [1:384:5] ICMP PING [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.24.4.42 -> 172.16.1.127
08/16-01:18:42.099769  [**] [1:499:4] Intrusion Detected! [LARGE PACKET SIZE] [**] [Classification: Potentially Bad Traffic] [Priority: 2] {I
CMP} 172.16.1.127 -> 172.24.4.42
08/16-01:18:42.099769  [**] [1:408:5] ICMP Echo Reply [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.16.1.127 -> 172.24.4.42
```

4. Challenges faced during installation

Throughout the installation phase, we were riddled with a number of errors and challenges. Many of the errors were observed during installation of OpenStack using DevStack script.

- An error that we received during the execution of the DevStack script is related to the python wheel package not being installed which prohibited us from running ./stack.sh To resolve this error, we tried to manually install the wheel package, but it threw the same error. We eventually resolved the error by making the following changes to `requirements/setup.py` file.

```
import setuptools

setuptools.setup(
    setup_requires=['pbr>=2.0.0', 'wheel'],
    pbr=True)
```

- In the case of network creation needed to learn the difference between the available networks present and what it meant to have a provider network and a self-serviced network. At first, we tried creating a provider network, not knowing that the public network present is already acting as the provider. Errors were creating in the provider network because there can be only one provider with a flat network that’s connected to the internet

- It is known that on the GUI we need to enter our admin credentials on the Horizon dashboard to have privileges, but we didn't encounter any prompt after installation so continued ahead with the network creation, later on, we realised that some of the commands were not accessible to us. After some research on the net we realised we needed to use “source openrc admin", and the credentials were taken from the local.conf file, thus realising its importance. This command isn’t present in the documentation

- At first, we used the default security group that was present to create our instance, but then faced issues when connecting to the instance. It was then we checked out the permissions on the default group and had to enable TCP for ssh and ICMP for ping

- During our testing of creating RSA keys, we faced issues while logging into our instance and got a connection refused error. We then speculated the multiple RSA keys could be the reason for it. Rightly so we were able to log in after keeping a single RSA key

- When we created an instance with cirros, we realised that it doesn't have a package manager, so we needed to create another image using Ubuntu as our base. When we created the image without specifying the hw_disk_bus as virtio, also our usual ISO files does not work, it needs a special cloud modified OS.

5. Reference

- https://docs.openstack.org/python-openstackclient/rocky/cli/index.html
- https://docs.openstack.org/devstack/latest/
- https://docs.openstack.org/install-guide/launch-instance.html
  https://www.researchgate.net/publication/327536160_Experimental_Analysis_of_DDoS_Attacks_on_OpenStack_Cloud_Platform_ICCCN_2018_NITTTR_Chandigarh_India
- https://www.snort.org/