Network Traffic Monitoring System for an IaaS in Cloud

Ashwin Kini

Computer Science, CIDSE Arizona State University

adkini@asu.edu

Lakshmi Srinivas

Computer Engineering, CIDSE Arizona State University

lsriniv2@asu.edu

Subhadarshi Samal Computer Science, CIDSE Arizona State University ssamal1@asu.edu

*Abstract*— Infrastructure as a service (IaaS) model gives the cloud vendors the power to deploy the computing resources as services to their consumers. We need a robust network traffic monitoring service in the cloud architecture to provide more insights on network traffic to the cloud administrator. This provides complex traffic visibility solutions which can be engineered for a diverse set of use cases, ranging from network administration and trouble-shooting to application/network security, data analytics and more. Monitoring is required in cloud for various purposes such as Load balancing, detection of network attacks and traffic analysis.

Multiple security and network challenges are targeted towards the cloud. This project aims at building a robust traffic monitoring system on top of OpenStack leveraging the Tap-as-a-Service API in order to perform traffic monitoring and using this data to perform network analytics and classification by implementing cloud in miniature model which is called DevStack. This traffic monitoring system is further integrated with Intrusion detection system (IDS) to provide detection of malicious traffic entering into the cloud system.

Keywords— Cloud, IaaS, TaaS, Traffic Monitoring, Network Analytics

# Introduction

Infrastructure as a Service (IaaS) provides the consumers with capability to manage and provision processing resources, storages and networks. The cloud administrator can manage and control the operating systems running on top of the underlying hardware and harness its potential. IaaS follows an on demand model in the cloud service models. There are many IaaS vendors in the market eg : Amazon EC2.

IaaS vendors follow a pricing model based on time and resources consumed. Many consumers are opting for public Iaas based clouds. It has become very important for deploying network monitoring tools in such Iaas environments for better visibility and performance related issues. Cloud administrators can gain further insights from the monitored traffic and use this data for further analysis. This is typically true on Public clouds managed by huge IaaS vendors.

Network Monitoring is an important feature that allows network admins to debug their virtual traffic and analyze network traffic associated with it. Many other features like IDS etc. can be integrated with network monitoring tools thereby giving more visibility. Network traffic monitoring is the process of reviewing, analyzing and managing network traffic for any abnormality or process that can affect network performance, availability and/or security. It is a network management process that uses various tools and techniques to study computer network-based communication/data/packet traffic.

TaaS (Tap as a service) is a feature developed to mirror traffic on OpenStack provisioned networks. This will help administrators to debug the traffic and gain visibility into Virtual machines by monitoring and analyzing network traffic associated with them. This service is built on top of Neutron API and exposes API for developers to build many applications for network related requirements.

We leverage the Tap as a Service (TaaS) API and analyze the traffic on a target node. We build a private cloud on OpenStack and leverage TaaS.  Neutron allows us to build a virtual network on OpenStack and manage them. The application will leverage the TaaS API and we plan to do network analysis and classification on top of it. TaaS API does port mirroring on the switches that connect the VM’s spawned on top of OpenStack. This allows us to analyze the traffic in the tenant network.

In an organization, there are many possible signs of incidents which may go unnoticed each day. These events can be studied mainly by analyzing network behavior or by reviewing computer security event logs. Having a single approach and a unified platform helps with this very difficult and challenging task to monitor and report in near-real time

This project implements a log based Intrusion Detection to create a comprehensive security monitoring platform over cloud.

NIDS use NICs running in promiscuous mode to capture and analyze raw packet data in real time. Most NIDS use one of two methods to identify an attack: statistics anomaly detection or pattern-matching (or signature-based) detection.

*Anomaly-based Detection:* In the statistics anomaly detection model, the software discovers intrusions by looking for activity that is different from a user’s or system’s normal behavior. Since anomaly detection techniques signal all anomalies as intrusions, false alarms are expected when anomalies are caused by behavioral irregularity instead of intrusions. Hence, pattern recognition techniques and anomaly detection techniques are often used together to complement each other.

*Signature-based Detection:* A signature based NIDS examines visible network activity, looking for suspicious activity such as port scans, packets with fake source addresses, etc. Not all malicious activity is visible on a given host’s NIC, so the location and number of such scanners must be carefully thought out.

Signature based detection works by comparing packets to a set of attack signatures, which are stored in some form of database and must be frequently updated as new attacks emerge. (This works like many virus scanners.) Such scanners can detect attacks well and quickly, but not zero-day attacks

The advantages to a pattern-matching IDS include an easier implementation and smaller learning curve for the administrator, as well as the generating fewer false positives. However, they are more easily fooled by the obfuscation and evasion techniques of black hats.

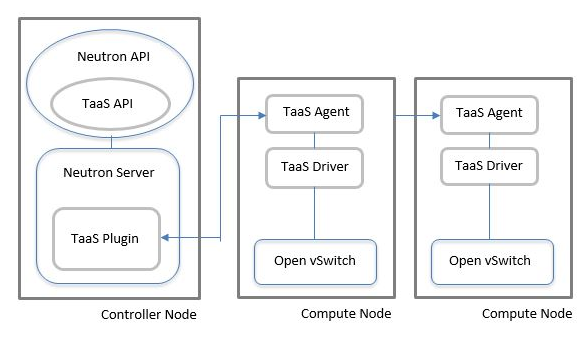


Figure 1: Tap-as-a-Service in Cloud Deployment

# System Models

## System Model

The traffic monitoring system will be implemented by using OpenStack as the cloud platform. As show in Figure 1, the system consists of a Vlab instance on which the cloud is hosted. We installed DevStack on it to incorporate the OpenStack environment with all its components. Since the neutron component of OpenStack deals with all the network related functionalities, we concentrate on this component for the development of our project.

Tap as a Service (TaaS) API is integrated with the neutron component. Tap as a Service API allows us to do port mirroring in a tenant network. By using port mirroring we mirror all the traffic of a network on one single port .We have 2 target VMs whose traffic will be monitored by port mirroring all the traffic to another VM (Network Traffic Analytics). An external network which consists of an Ubuntu/Kali VM is used to perform attacks/ send traffic to the cloud instances and thereby demonstrate the network attacks prevention on the cloud.

We use tools like tcpdump, scapy etc on Network Traffic Analytics VM. Then integrate it with IDS like Snort which can give the tenant admins more visibility into network attacks in the tenant network.

*Snort* – This is the sensor component responsible for monitoring the raw traffic and comparing the traffic to rules. Snort is currently the most popular free network intrusion detection software. Advantage of snort is that it allows for raw packet data analysis. This allows for examination of a packet down to the payload to determine what caused the alert, why the something caused the alert, and whether action needs to be taken. Snort monitors or "sniffs" network packets and logs and can alarm the administrator of a packet matching a specific rule or containing specific information. It can be configured to block the source of the anomaly or make a system configuration change. Snort is best used on small to medium sized networks, single hosts, or on segments of a large network. It is not recommended to be used to monitor an entire large scale network. Snort can be configured to be a packet monitor (sniffer) to capture and log all network packets coming across the specified network interface.

Snort version 2.9.8.3 is used for this system model. You can download Snort at http://www.snort.org/downloads.html.

*Components of Snort:*

Snort is basically a combination of multiple components. All the component work together to find a particular attack and then take the corresponding action that is required for that particular attack. Basically it consists of following major components as shown in figure 3 [12]:

1. Packet Decoder

2. Preprocessors

3. Detection Engine

4. Logging and Alerting System

5. Output Modules

Packet comes from internet and enters into packet decoder and it goes through several phases, required action is taken by snort at every phase like if detection engine found any miscellaneous content in packet then it drops that packet and in the way towards output module packet is logged in or alert is generated.

*Barnyard2* – This processes the alerts generated by snort and processes them in to a database format. It is an open source interpreter for Snort unified2 binary output files. Its primary use is allowing Snort to write to disk in an efficient manner and leaving the task of parsing binary data into various formats to a separate process that will not cause Snort to miss network traffic. Snort needs rules and periodically have to update them.

*Snorby* – The final part of our installation is a web GUI that we can use to monitor and manage any alerts generated by snort. There are several options, my preference is Snorby. This is the visual front end to the event data that is written in to the database. The basic fundamental concepts behind Snorby are simplicity, organization and power.

Snorby boasts a robust set of features including:

1. Metrics & Reports -Drill down into your data by day, week, month, or custom timetables and even export to pdf.

2. Classify events into a number of predefined classifications or create your own

3. Full packet and session data monitoring using OpenFPC, Solera DS Appliances, and Solera’s DeepSee

4. Keyboard friendly hotkeys let you navigate the interface without a mouse

5. Extensibility via third party plugins

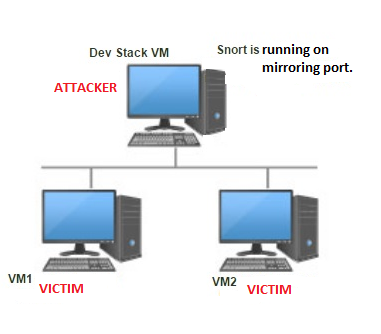


Figure 1: System Model

## Software

For the implementation of Traffic Monitoring System for an Infrastructure as a Service (IaaS) in Cloud we choose OpenStack as our cloud software. DevStack is an opinionated script which quickly creates an OpenStack development environment and can be used for starting or running OpenStack services. The details on the software requirement for the development of this project is as follows:

1. Hardware OS: Ubuntu 16.04
2. Cloud Setup Requirements : Devstack for OpenStack , Neutron API, Tap as a Service (TaaS) API
3. Host VMs OS : Ubuntu 16.04 LTS (Linux)
4. Languages: Python
5. Software & Tools: Wireshark, tcpdump, snorby, snort, Barnyard2, plotty

# Project Description

In this project, we will be creating a private IaaS cloud using OpenStack. Neutron API in OpenStack is a framework which allows the cloud users to build network topologies and to configure advanced network policies in the Cloud. The project is aimed at implementing a robust traffic monitoring system for the cloud instances and utilizing this information for network traffic acquisition, classification, and analysis. It is also targeted to create a statistical data of the collected traffic flow.

## Project Overview

We have divided the project into 5 tasks. Our Midterm goal is to implement 3 tasks successfully which is more than 50% of the project. Our final goal is to collaborate network traffic monitoring with network traffic analysis.

## Task 1 : Cloud Environment Setup.

This is the start of the project. The aim is to setup project environment in cloud platform. Mainly setting up the VMs and installing OpenStack on top of them.

[OpenStack](http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html) is an open source cloud computing platform that supports all types of cloud environments. It provides an Infrastructure-as-a-Service ([IaaS](http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html)) solution through a variety of complemental services. Each service offers an application programming interface ([API](http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html)) that facilitates this integration.

The cloud environment setup is as shown in Figure 3. The implementation is done using OpenStack, by installing DevStack on Vlab (ThothLab) VM. We make use of a Single Flat Network architecture, which is a shared network visible to all tenants. The setup details is as follows:

* One public network to access the external network.
* One tenant network which consists of VM instances.
* A router/ gateway to connect the public and private networks.
* Three VMs are hosted on a private network, one Ubuntu VM for traffic monitoring and two as Target VMs whose traffic needs to be monitored.

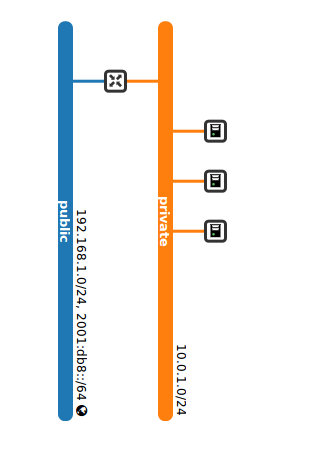


Figure 3: OpenStack Setup on ThothLab VM

## Task 2: Setting of TaaS (Traffic Monitoring System)

Aim of this task is to successfully integrate TaaS in OpenStack and direct traffic to the TaaS Agent (VM). The traffic flow data is used to analyze the type of traffic inflow and outflow in the network.

Tap-as-a-Service (TaaS) is an extension to the OpenStack network service (Neutron). It offers an API that enables a tenant (or the cloud administrator) to monitor ports in Neutron provisioned networks. It provides remote port mirroring capability for tenant virtual networks. Port mirroring involves sending a copy of packets entering and/or leaving one port to another port.

For enabling TaaS service on the TaaS agent and mirroring traffic from all the other VMs in the same private network onto the TaaS agent, we need to create Tap service and Tap flows:

* ***Tap-Service:***

Tap-Service instance is created with Neutron port serving as the destination side of a port-mirror session and this port is assigned to the monitoring Virtual machine to consume the monitored traffic, TaaS agent in Figure 4.

* ***Tap-Flow:***

Tap-Flow needs to be created to associate a port that needs to be monitored to the tap-service instance. And this port needs to be attached to the corresponding Target VM1 and VM2, whose traffic needs to

be mirrored.

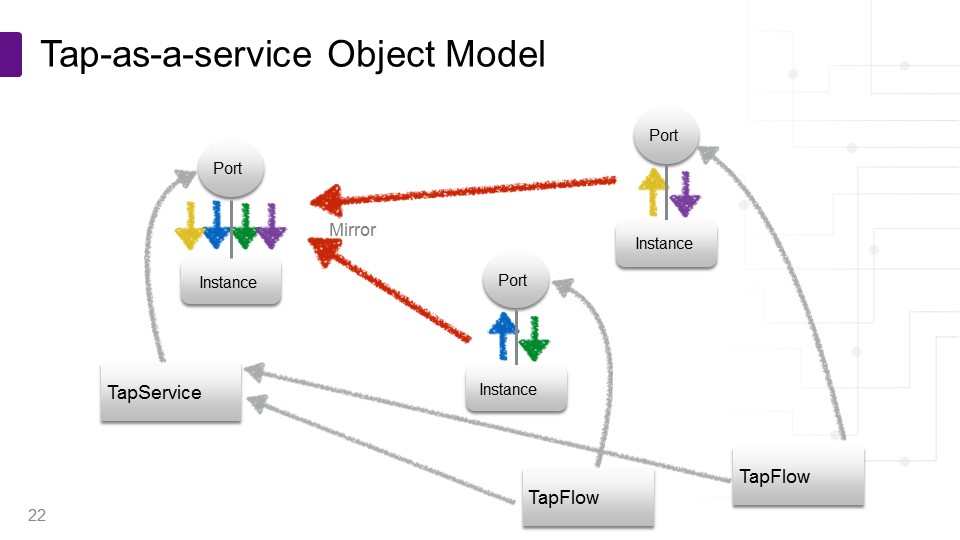


Figure 4: Object Model for Tap as a Service.

The figure 4 shows a object model representation of Tap as a Service. We see Tap flows attached to ports that needs to be monitored. The traffic through these ports are mirrored on a port to which Tap service is attached. Hence Tap as a service helps in managing and mirroring the traffic which helps in setting up the traffic monitoring system in a tenant network.

The Tap as a Service is available as an extension that can be integrated with neutron and can be used to setup port mirroring in a tenant network.

The figure 5 depicts a setup with Taas integrated onto the ports attached to instances whose traffic needs to be monitored.

A detailed setup and configurations information can be found on gitlab repository [1].

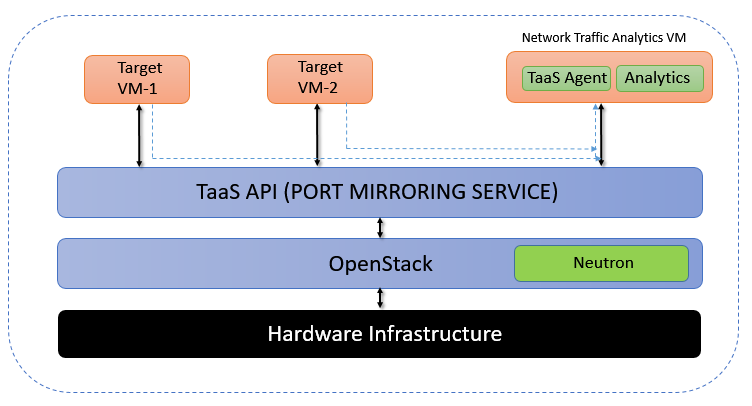


Figure 5: Tap-as-a-Service (TaaS) Deployment in an OpenStack Cloud

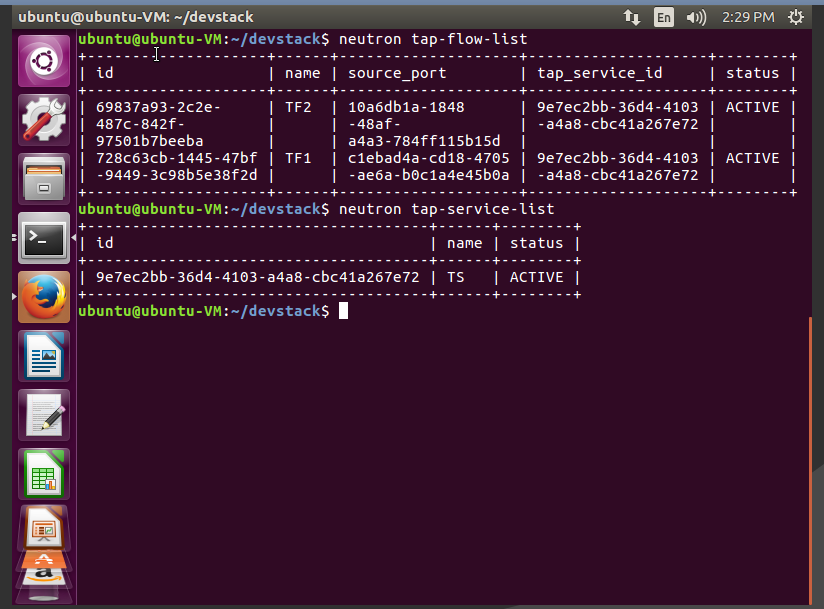


Figure 6: TaaS setup on DevStack

## Task 3: Network Traffic Acquisition (Monitoring)

The key objective behind network traffic monitoring is to ensure availability and smooth operations on a computer network. Network monitoring incorporates network sniffing and packet capturing techniques in monitoring a network. Network traffic monitoring generally requires reviewing each incoming and outgoing packet.

This task involves gathering of data on the TaaS agent (VM) and providing a means (GUI or text file) to interpret this data by the network administrator to perform analysis on the acquired information.

The aim of this task is to give the tenant admin means to observe the entire network traffic. Task 2 will mirror traffic from target ports to the monitoring port. Hence, we can use tools like tcpdump, iptraf etc. to sniff traffic from the interface of the monitoring VM connected to tap-service port. The output of the tcpdump can then be saved as pcap files. These pcap files can be further used for network traffic classification, visualization and also to detect network intrusion etc for further tasks.

Uses of traffic monitoring:

* Analyze and monitor network traffic and performance.
* Identify bandwidth hogging users and applications
* Validate network traffic prioritization policies

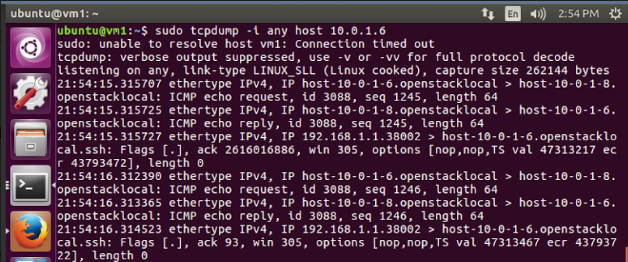


Figure 7: tcpdump capture output

## Task 4: Network Traffic Classification and Analysis

This task defines in details about the traffic classification and analysis of the network traffic obtained upon successful monitoring of the cloud instances. This data can be further processed to gain granular and detailed information on network traffic. Using this data, we can do the following:

*Traffic Classification:*

The classification of data is based on Protocol – Layer 4 (TCP, UDP), Layer 3 (ICMP), Layer 2 (ARP), Layer 5 (HTTP, SSH). A simple python code is developed for performing this task, the source code for which can we found on Gitlab[1]. The python code written gives the following deliverables:

* Classifying the traffic based on different protocols:

The tcpdump output is stored onto a pcap file which is later used for processing of the data. The classification is done as post-analysis. A python code is written for this purpose.

* Counts the number of packets belonging to each protocol:

In Figure 6, we see the traffic being classified based on different protocols which is displayed on the plotty webpage, and can later be downloaded. We can also find the pcap files for each kind of traffic which can be used for further processing.



Figure 8: Output of traffic classifier

* Plotting of the graph:

Plotty packages are used in the python code which takes in values and plots the required graph and stores it under the “My files” tab in the mentioned plotty account. A sample output is as shown in Figure 7.

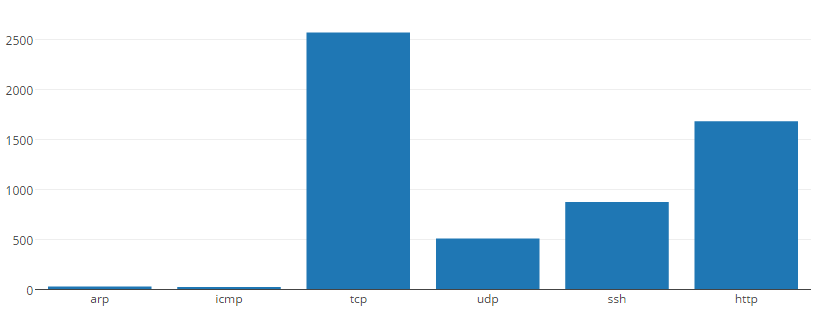


Figure 9: Graphical representation of traffic statistics

*Traffic Analysis (Integration of Intrusion Detection System):*

Another usage of this data is to address some security threats that are targeted towards the cloud by closely monitoring suspicious activities by using snort rules and alerting the administrator during intrusions.

We consider that the classification has mainly 4 dimensions. Source Address, Destination Address, Source Port, Destination Port. The number of these parameters can be extended without affecting the method (can be extended for protocols).

How Snort works?

1. Packet Decoder - The packet decoder captures the packet from different types of network interfaces and setup packets for preprocessing.
2. Preprocessor - Pre-processors are used to arrange and modify packets before analyzed by the detection engine. And they try to detect some basic anomalies in the packet header. Preprocessors are very important for any IDS to prepare data packets to be analyzed against rules in the detection engine.

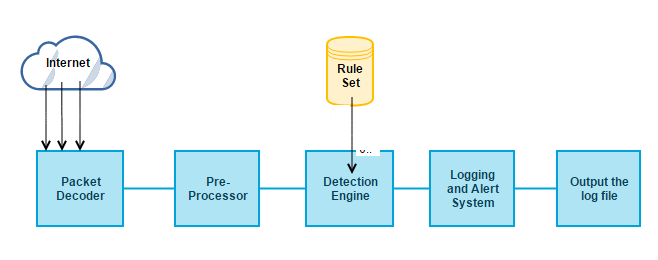


Figure 10: Snort work flow

1. Detection Engine - Detection Engine is the heart of the snort. Its responsibility is to analyze all the packets passing through it for the sign of intrusion by making use of certain predefined rule. If the packet matches the rule appropriate action will be taken; otherwise the packet is dropped. It may also take different amount of time to respond for different packets irrespective of how many rules we define.
2. Logging and Alert System - Depending upon what detection engine finds out, the activity is logged or alert is generated. Logs are kept in simple text files or tcp-dump style files. The location of logs and alert can be modified using –l command in the command prompt.
3. Output Modules- It is used to control the type of output produced by the logging and alert system. Some of its function includes may be generating log reports, sending SNMP traps, logging into databases (like MySql), sending a message to sysylog server etc.

*Snort as Sniffer:*

Snort acts like tcpdump in sniffer mode. The Snort sniffer mode output is slightly different than the other command-line sniffers. It is actually very easy to read and you may find you prefer it for quick captures. One very nice feature of sniffer mode Snort is the network traffic summary at the end of the capture.

# snort –v –d –e -v Dump the packet header to standard output -d Dump the packet payload (includes all TCP, UDP, ICMP packets) -e Display the link layer data

*Snort as a Packet Logger:*

Once after the sniffing is done packet has to be logged. Logging is as simple as adding the -l option, followed by the directory in which you wish to store the logs. Snort's default logging directory is \snort\log. We can use the -d, -a, and -e options to control the amount of information logged for each packet # snort –l {log-directory} will log the packet to specified folder.

*Snort as NIDS:*

When used as an NIDS, Snort provides near real-time intrusion detection capability. Snort does not log the captured file but it applies the rule, if any matches found then it will log or alert the system. snort –c C:\Snort\etc\snort.conf will starts the snorts in NIDS mode.

This command displays the alert in the log directory. #snort –vde –l c:\snort\log –c c:\snort\etc\snort.conf

*Analyzing and Displaying Snort logs (Snorby and Barnyard2):*

We used Snorby and Barnyard2 to represent snort logs in GUI. The graphical representation also can show the analysis of network traffic based on severity of the intrusion.

One of the issues that came with snort is that how snort can keep processing the network traffic without dropping packets and performing extensive output operations such as send alerts and log them to the syslog or a database. One of the solutions was to make snort multithreaded, but it was a nightmare for developers to maintain a stable multithreaded version. So Barnyard2 is used to satisfy this purpose. Barnyard2 was first created to isolate processing of output data from snort and keep the later focusing on more fundamental operations to monitor the network traffic. As snort has several modes, Barnyard2 also provides two modes which are batch processing and continual processing. First, in batch processing mode, Barnyard2 will process the each and every pre-specified unified files and then quit. The advantages of this mode are pulling tangible data from a unified file, reloading old data into a database, or testing new plug-ins used in snort. Second, in continual mode, events can be processed instantly if they triggered snort alerts. Barnyard2 has one more capability to localize alert messages in very easy way because the data is loaded from sid-msg.map and gen-msg.map files. Unlike snort which has 48 rule files, preprocessors, and rule options.

Snorby is front end web application for any application that logs events in the unified2 binary output format. It doesn’t perform any network monitoring tasks by itself, instead it depends on other IDS’s such as Snort to report data to it. Since snorby itself is just a dashboard, it allows them to have sensors for events of their choice or even have custom sensors in the IDS of their choice. It also supports full packet capture so that the users can see complete packets which makes it easier to determine if an attack occurred. Snorby can generate hourly/daily/weekly or monthly graphs and reports for monitoring data. It allows classification of events based on preset classes or based on user defined custom classes.

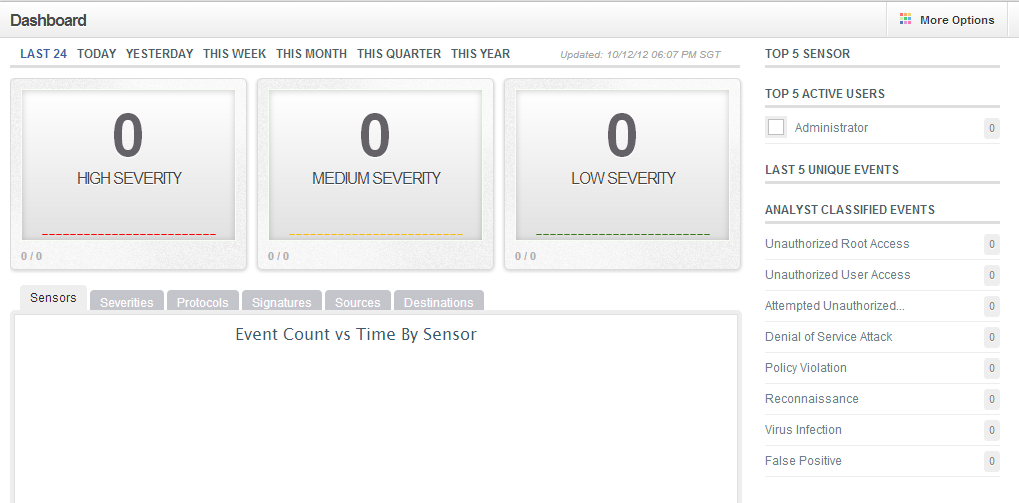


Figure 11: Snorby’s Dashboard

*Snort Rules:*

Snort uses a simple, lightweight rules description language that is flexible and quite powerful. Snort rules must be completely contained on a single line, the Snort rule parser doesn't know how to handle rules on multiple lines.

Snort rules are divided into two logical sections, the rule header and the rule options.  The rule header contains the rule's ction, protocol, source and destination IP addresses and netmasks, and the source and destination ports information.  The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

Here is an example rule: alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; msg: "mountd access";)

The text up to the first parenthesis is the rule header and the section enclosed in parenthesis is the rule options.  The words before the colons in the rule options section are called option keywords.  Note that the rule options section is not specifically required by any rule, they are just used for the sake of making tighter definitions of packets to collect or alert on (or drop, for that matter).  All of the elements in that make up a rule must be true for the indicated rule action to be taken.  When taken together, the elements can be considered to form a logical AND statement.  At the same time, the various rules in a Snort rules library file can be considered to form a large logical OR statement.

Snort rule Implementation to analyze network traffic:

1) Rule to prevent ICMP flood attack: alert icmp any any -> any any (msg:"Alert! ICMP FLOOD DETECTION!!!"; itype: 8; threshold: type threshold, track by\_dst, count 10, seconds 30; rev:001; sid:10000001; priority:1;)

Explanation:

* A rule action. In this rule the action is “alert”, which means that an alert will be generated when conditions are met. Remember that packets are logged by default when an alert is generated. Depending on the action field, the rule options part may contain additional criteria for the rules.
* Protocol. In this rule the protocol is ICMP, which means that the rule will be applied only on ICMP-type packets. In the Snort detection engine, if the protocol of a packet is not ICMP, the rest of the rule is not considered in order to save CPU time. The protocol part plays an important role when you want to apply Snort rules only to packets of a particular type.
* Source address and source port. In this example both of them are set to “any”, which means that the rule will be applied on all packets coming from any source. Of course port numbers have no relevance to ICMP packets. Port numbers are relevant only when protocol is either TCP or UDP.
* Direction. In this case the direction is set from left to right using the -> symbol. This shows that the address and port number on the left hand side of the symbol are source and those on the right hand side are destination. It also means that the rule will be applied on packets traveling from source to destination. You can also use a <- symbol to reverse the meaning of source and destination address of the packet. Note that a symbol <> can also be used to apply the rule on packets going in either direction.
* Destination address and port address. In this example both are set to “any”, meaning the rule will be applied to all packets irrespective of their destination address. The direction in this rule does not play any role because the rule is applied to all ICMP packets moving in either direction, due to the use of the keyword “any” in both source and destination address parts.
* The msg rule option tells the logging and alerting engine the message to print along with a packet dump or to an alert.
* The sid keyword is used to uniquely identify Snort rules. This information allows output plugins to identify rules easily. This option should be used with the rev keyword.

2#2100 Reserved for future use

100-999,999 Rules included with the Snort distribution

36#361,000,000 Used for local rules.

* The rev keyword is used to uniquely identify revisions of Snort rules. Revisions, along with Snort rule id's, allow signatures and descriptions to be refined and replaced with updated information. This option should be used with the sid keyword.
* The priority tag assigns a severity level to rules. A classtype rule assigns a default priority (defined by the config classification option) that may be overridden with a priority rule.
* The itype keyword is used to check for a specific ICMP type value.
* The threshold keyword defines a rate which must be exceeded by a source or destination host before a rule can generate an event.

#### Snort rule to prevent icmp flood attack:

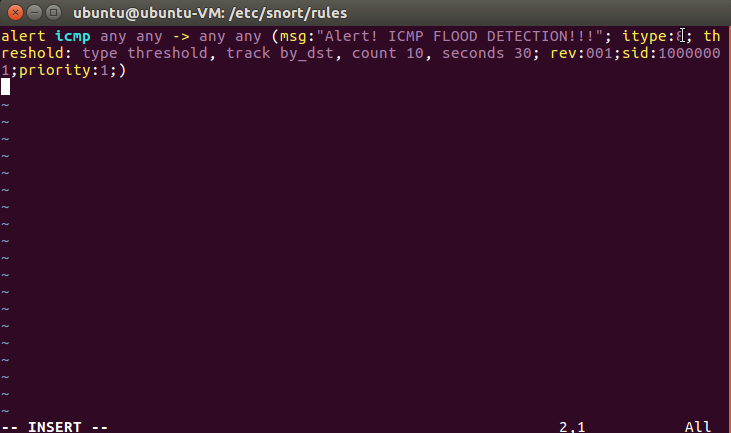


Figure 12: Snort rule to prevent icmp flood attack

1. Icmp flood attack

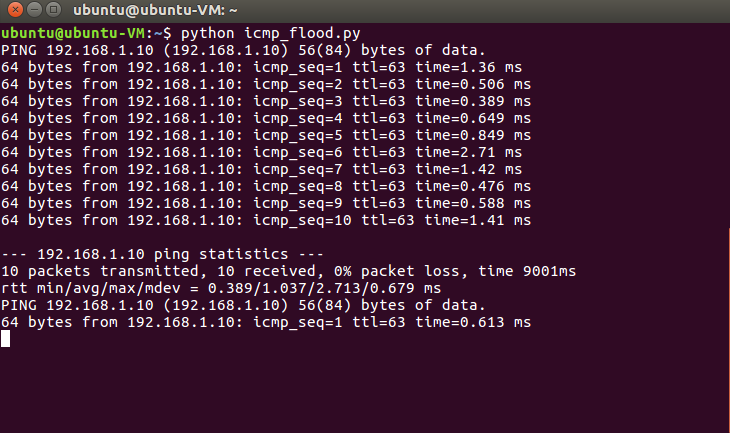


Figure 13: Performing icmp flood attack

The command used for ICMP flood attack:

It is used to flood large amounts of data packets to the victim's computer in an attempt to overload it.

hping3 --flood --rand-source --icmp -p 445 (target ip)

1. Snort rule is able to detect the attack

Snort writes events to binary log file. Barnyard2 reads those snort log file snort.u2 and stores it into the

MySQL database.

This is done using the command:

sudo barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -w /var/log/s nort/barnyard2.waldo -g snort -u snort

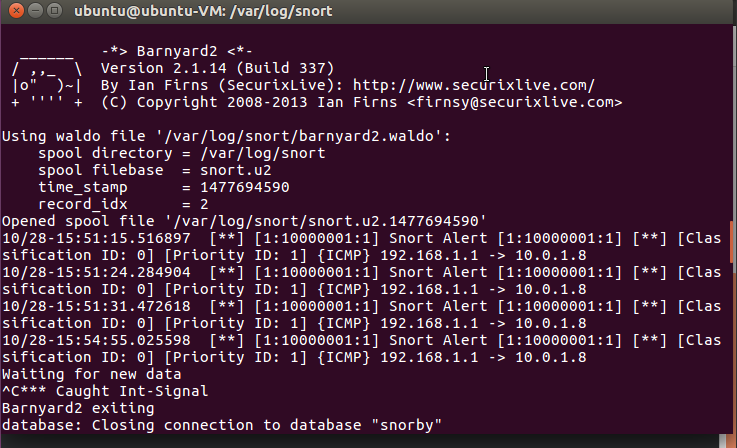


Figure 14: Writing events into MySQL Database

1. Severity analysis on Snorby GUI

Snorby pull the data from the database and provides a graphical representation of the attacks with severity levels for the administrator to further act according to prevent these malicious traffic.

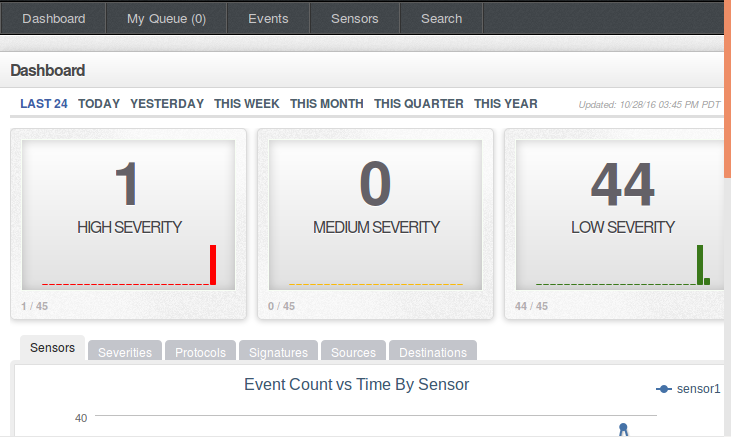


Figure 15: Snorby pull data from database and displays

2) Rule to detect Smurf attack: alert icmp any any -> any any (msg: “Alert! SMURF DETECTION !!!”; itype:8; threshold: type limit, track by\_src, count 10, seconds 30; sid: 10000004;rev:001)

Smurf attack command: hping3 -1 --flood -a VICTIM\_IP BROADCAST\_ADDRESS

-a: to spoof IP address -1: used for ICMP protocol

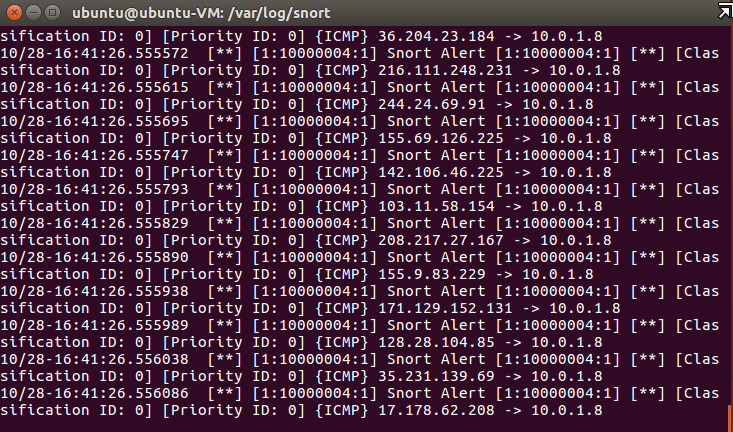


Figure 16: Attack log captured in Snort

3) Rule to detect SYN-FLOOD attack: alert tcp any any -> any 80 (msg:”Alert! SYN FLOOD DETECTION!!”; flags:S;

flow: stateless; threshold: type limit, track by\_src, count 70, seconds 1; sid: 10000006;rev:001)

The flags keyword is used to check if specific TCP flag bits are present. The following bits may be checked:

F: FIN - Finish (LSB in TCP Flags byte)

S: SYN - Synchronize sequence numbers

R: - RST – Reset

SYN\_FLOOD attack command: hping3 -S (target ip) –a (spoofed IP) -p 80 –flood

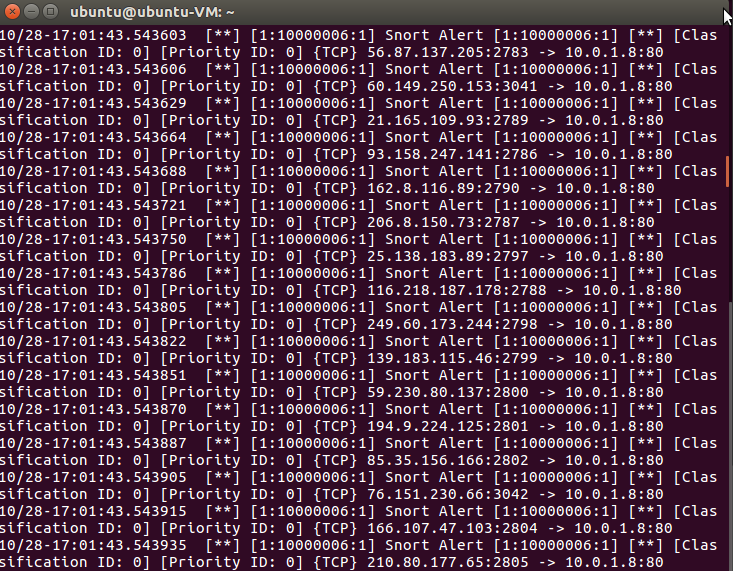


Figure 17: SYN \_FLOOD attack log captured in snort

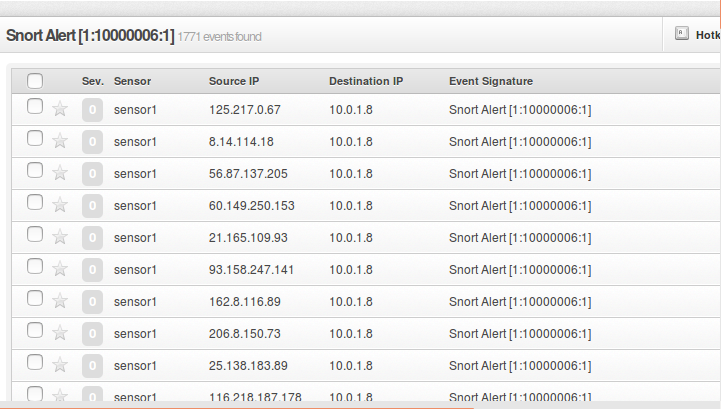


Figure 18: Severity Alert in Snorby

4) Rule to detect UDP FLOOD attack: alert udp any any -> any 80 (msg:”Alert! UDP FLOOD DETECTION!!”;

Priority:2; sid:10000008;rev:001)

UDP FLOOD attack command: hping3 –flood –rand-source –udp –p 445 (target ip)

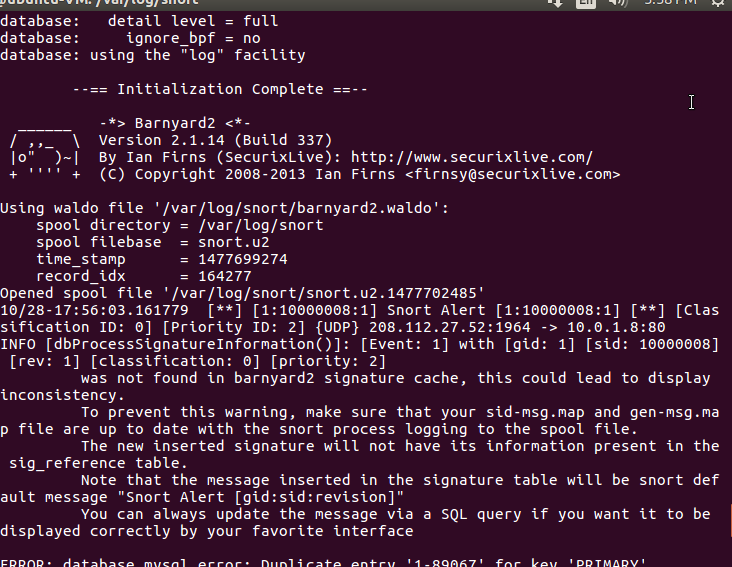


Figure 19: UDP FLOOD log captured in snort

## Task 5: Documentation

In order to produce a good report the data gathering and appropriate documentation of design, methodology, implementation and reporting of results play a vital role. So this task will be done continuously as the project progresses. Documenting our implementation methods and findings is done simultaneously for each task. Final outcomes and success/failures will be mentioned at the end in the final report.

## Project Task Allocation

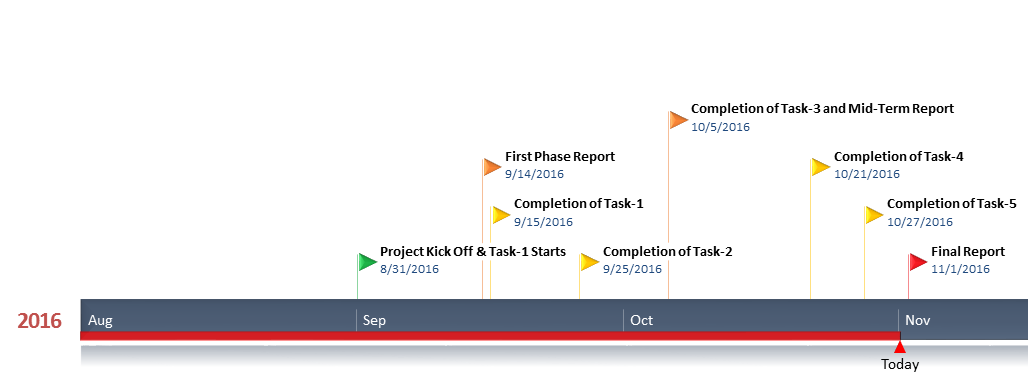
|  |  |  |  |
| --- | --- | --- | --- |
| **Task Name** | **Person Responsible** | **Work (%)** | **Progress** |
| Cloud Environment setup | Ashwin, Lakshmi, Subhadarshi | 10 | Complete |
| Setting of TaaS | Ashwin, Lakshmi | 25 | Complete |
| Network Traffic Monitoring | Lakshmi, Subhadarshi | 25 | Complete |
| Network Traffic Acquisition, Classification and Analysis | Subhadarshi, Ashwin | 25 | Complete |
| Documentation | Ashwin, Lakshmi, Subhadarshi | 15 | Complete |

## Deliverables

The deliverables of this project can be classified as follows.

* First is to design and implement a Network Traffic Monitoring System.
* Second is to provide a GUI to represent the statistical traffic analysis from the first deliverable.

## Project Timeline



# Risk Management of the project

OpenStack setup is a tedious job on a standalone machine and also resource allocation. Integration of TaaS and network monitoring application will be a demanding task. TaaS is a new feature added to the OpenStack functionalities. We are building on top of TaaS which might be challenging. We are considering ThothLab as the primary option for the project. Stability of the Devstack VM on Thothlab is a concern. We plan to review every week so that we stay on time.

# Conclusion

The Network Traffic Monitoring system helps cloud administrators’ to capture traffic and perform analysis. This feature can be integrated into OpenStack. To monitor the traffic going to and from all the OpenStack instances we use TaaS which does port mirroring on the Monitoring VM. We also implement a network traffic analyzer and classifier to categorize data and use this to further use this for detecting malicious traffic and prevent attacks. The network analysis is implemented mainly based on signature based intrusion detection. It will be efficient to capture all the known types of attacks. But it may fail to detect any new type of attack. The attack detection has to be anomaly based along with signature based. Our future work includes efficient anomaly detection mechanism in the IDS to analyze irregularities in traffic. Again, future work can also include integrating it with other systems for a more robust and secure system model.

##### Acknowledgment

We would like to express our gratitude to Dr. Dijiang Huang for providing us an opportunity to explore projects related to Cloud and OpenStack through Cloud Computing course. Also, we would like to thank our TA, Ankur Chowdhary for his guidance and continuous support through the course of this project.

##### References

1. <http://gitlab.thothlab.org/adkini/NetworkTrafficMonitoring.git>
2. <http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html>
3. [https://www.gigamon.com/blog/2015/10/26/tap-as-a-service-enabling-traffic-monitoring-in-openstack-clouds-2/J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.](https://www.gigamon.com/blog/2015/10/26/tap-as-a-service-enabling-traffic-monitoring-in-openstack-clouds-2/J.%20Clerk%20Maxwell,%20A%20Treatise%20on%20Electricity%20and%20Magnetism,%203rd%20ed.,%20vol.%202.%20Oxford:%20Clarendon,%201892,%20pp.68–73.)
4. <http://www.ijsr.net/archive/v4i2/21021503.pdf>
5. http://www.ussrback.com/docs/papers/IDS/snort\_rules.htm
6. Jadeja, Yashpalsinh, and Kirit Modi. "Cloud computing-concepts, architecture and challenges." Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on. IEEE, 2012.
7. Spring, Jonathan. "Monitoring cloud computing by layer, part 1." IEEE Security & Privacy 9.2 (2011): 66-68.

Demo Link: <https://youtu.be/0K6tusUQaUU>

APPENDIX

## SCRIPT USED FOR AUTOMATION OF CLOUD SETUP

#!/bin/bash

####AUTHOR : Ashwin

##Network name and subnet name

networkName="privateNetwork"

subnetName="privateNetwork\_sub"

##router name

routerName="router"

##range of IP

range="10.0.1.0/24“

## Import Key for SSH

KEY="key"

##Specify the boot Image

BOOTIMG="ubuntu"

##Specify the flavor

FLAVOR="ds1G"

##Specify the name of Vm's

NAMEVM1='VM1'

NAMEVM2='VM2'

##Tap service and flow names

TapService='TS'

TapFlow1='TF1'

#TapFlow2='TF2'

##source the session

source ~/devstack/openrc admin

##create network

echo "creating network "${networkName}""

var=$(neutron net-create "${networkName}" --port\_security\_enabled=False)

## Create router

echo "creating router"

neutron router-create "$routerName"

##Create subnet

echo "creating subnet"

neutron subnet-create "$networkName" "$range" --name "$subnetName"

##external interface ID

extInterface=$(neutron net-list | \

grep public | \

awk '{print $2}')

##Internal Interface ID

intInterface=$(neutron subnet-list | \

grep "$subnetName" | \

awk '{print $2}')

##Router ID

routerID=$(neutron router-list | \

grep "$routerName" | \

awk '{print $2}')

##Attach gateway to router

echo "creating router gateway"

neutron router-gateway-set "$routerName" "$extInterface"

##Attach interface private to router

echo "creating interface add"

neutron router-interface-add "$routerName" "$intInterface"

## Create port for Tap service

echo "creating tap service port"

portIDService=$(neutron port-create "$networkName" | grep -w id | awk '{print $4}')

## Create port for Tap flow

echo "creating tap flow port"

portIdflow1=$(neutron port-create "$networkName" | grep -w id | awk '{print $4}')

portIdflow2=$(neutron port-create "$networkName" | grep -w id | awk '{print $4}')

## Create tap service

echo "creating tap service"

neutron tap-service-create --name "$TapService" --port "$portIDService"

## Create Tap flow

echo "creating tap flow"

neutron tap-flow-create --name "$TapFlow1" --port "$portIdflow1" --tap-service "$TapService" \

--direction "BOTH"

echo $portIDService

echo $portIdflow

## Boot Vms now

echo "creating boot vms"

VMUUID=$(nova boot \

--nic port-id="$portIdflow1"\

--image "${BOOTIMG}" \

--flavor "${FLAVOR}" \

--key-name "${KEY}" $NAMEVM1);

VMUUID1=$(nova boot \

--nic port-id="$portIdflow2" \

--image "${BOOTIMG}" \

--flavor "${FLAVOR}" \

--key-name "${KEY}" $NAMEVM2);