Table of Contents

Stress Testing Under Different vRouter Configurations:

P1) Script to start the project

```
#Create Workplace

mkdir workspace
cd workspace/
```

```
#Install and start InfluxDB

wget https://s3.amazonaws.com/influxdb/influxdb_0.12.1-1_amd64.deb
sudo dpkg -i influxdb_0.12.1-1_amd64.deb
sudo apt-get update && sudo apt-get install influxdb
sudo service influxdb start
```

```
#Create a Database and a user

curl -G http://localhost:8086/query --data-urlencode "q=CREATE USER RootUser WITH PASSWORD 'Grafana' WITH ALL PRIVILEGES"
curl -G http://localhost:8086/query --data-urlencode "q=CREATE DATABASE TRAFFIC"
curl -G http://localhost:8086/query --data-urlencode "q=CREATE DATABASE Detect"
```

```
#Download Python dependencies

sudo apt-get install python-setuptools python-dev build-essential
sudo easy_install pip
sudo pip install numpy
sudo pip install sets
sudo pip install statistics
sudo pip install ciso8601
sudo pip install python-influxdb
```

#Install and pull the git repository

```
sudo apt-get install git
mkdir NetworkTrafficCollection
git clone https://github.com/BU-NU-CLOUD-SP16/Network-traffic-collection.git
NetworkTrafficCollection
```

P2) How To Parse Pcap File

(1). Pcap file format:

Global Header	Packet1 Header	Packet1 Data	Packet2 Header	Packet2 Data	...

The file has a global header, and each packet has a packet header and a packet data.

(2). Global Header:

```
typedef struct pcap_hdr_s {
                guint32 magic_number;
                guint16 version_major;
                guint16 version_minor;
                gint32 thiszone;
                guint32 sigfigs;
                guint 32 snaplen;
                guint32 network;
} pcap_hdr_t;
```

Python command to parse this header:

pcap_file_header_fmt = ['magic', 'version_major', 'version_minor', 'zone', 'max_len', 'time_stap', 'link_type']
global_head = unpack('IHHIIII', text[:24]
global_head_dict = dict(zip(pcap_file_header_fmt, global_head))

(3). Packet Header

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;
        guint32 ts_usec;
        guint32 incl_len;
        guint32 orig_len;
} pcaprec_hdr_t;
```

Python command to parse this header:

pcap_header_fmt = ['gmt_time', 'micro_time', 'pcap_len', 'len']
packet_head = unpack('IIII', text[:16])
pcap_head_dict = dict(zip(pcap_header_fmt, packet_head))

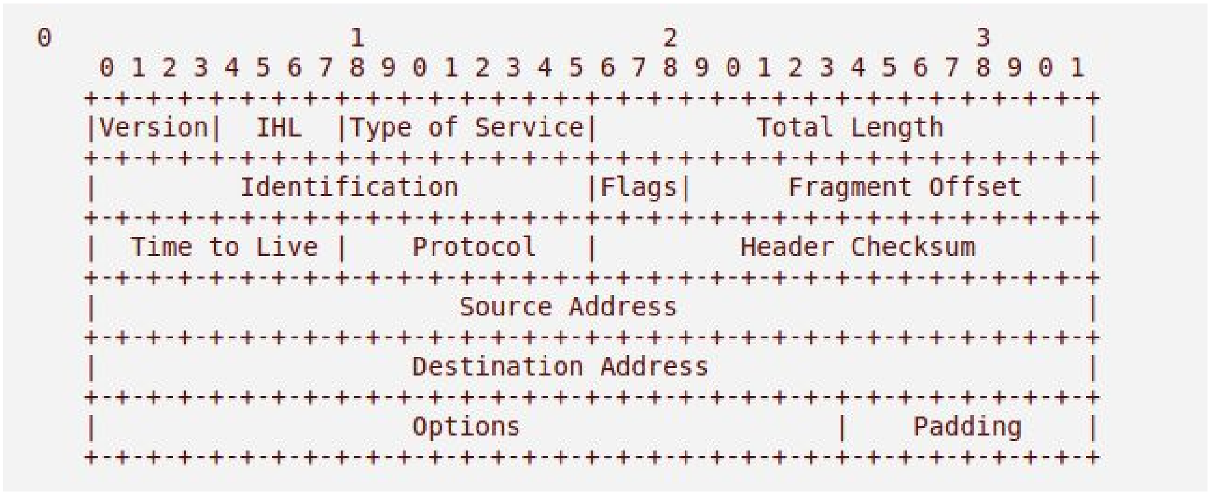
(4). Packet Data

Packet data format:

Mac Header	IP Header	TCP Header	(HTTP Header…)

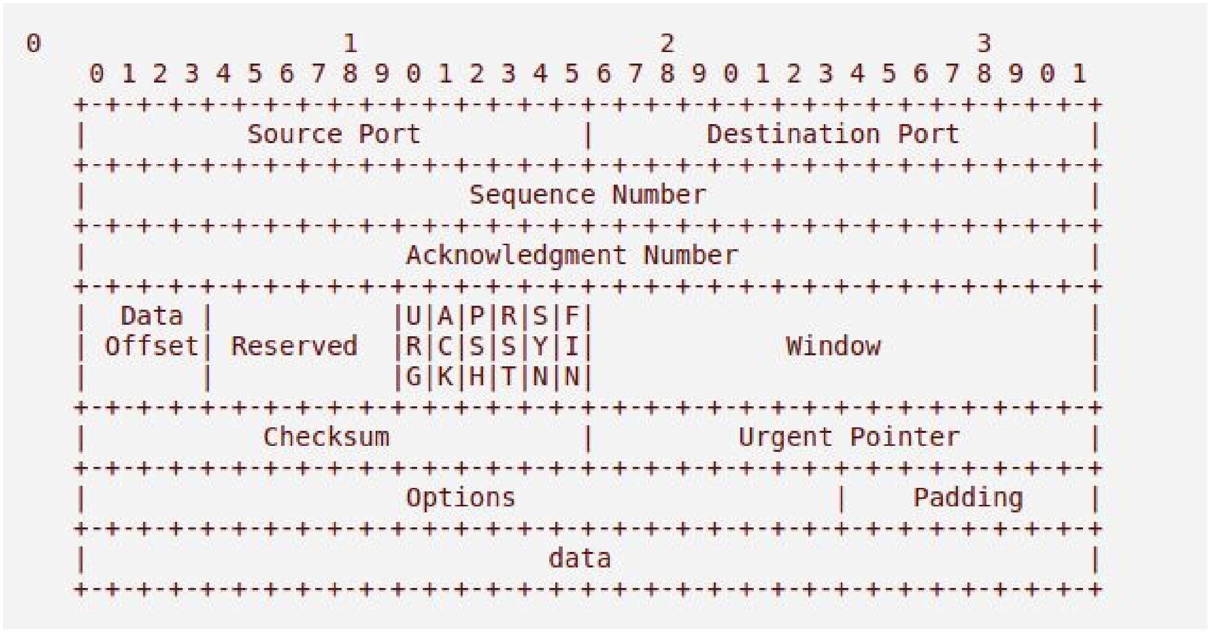
Mac head is first 14 bytes.

IP Header:



```
 0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |Version|  IHL  |Type of Service|          Total Length         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |         Identification        |Flags|      Fragment Offset    |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |  Time to Live |    Protocol   |         Header Checksum       |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                       Source Address                          |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                    Destination Address                        |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                    Options                    |    Padding    |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP Header:



```
0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |          Source Port          |       Destination Port        |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                        Sequence Number                        |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                    Acknowledgment Number                      |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |  Data |           |U|A|P|R|S|F|                               |
  | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
  |       |           |G|K|H|T|N|N|                               |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |           Checksum            |         Urgent Pointer        |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                    Options                    |    Padding    |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                             data                              |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Python command to parse these headers and data:

ip_header_fmt = ['version,ihl', 'tos', 'tot_len', 'id', 'frag_off', 'ttl', 'protocol', 'check', 'saddr', 'daddr']

tcp_header_fmt = ['src_port', 'dst_port', 'seq_no', 'ack_no', 'tcp_offset_res', 'tcp_flags', 'window', 'cksum', 'urg_pt']

ip_head = unpack('!BBHHHBBHII', skb[14:34])

ip_head_dict = dict(zip(ip_header_fmt, ip_head))

ip_head_length = (ip_head_dict['version,ihl'] & 0xF) * 4

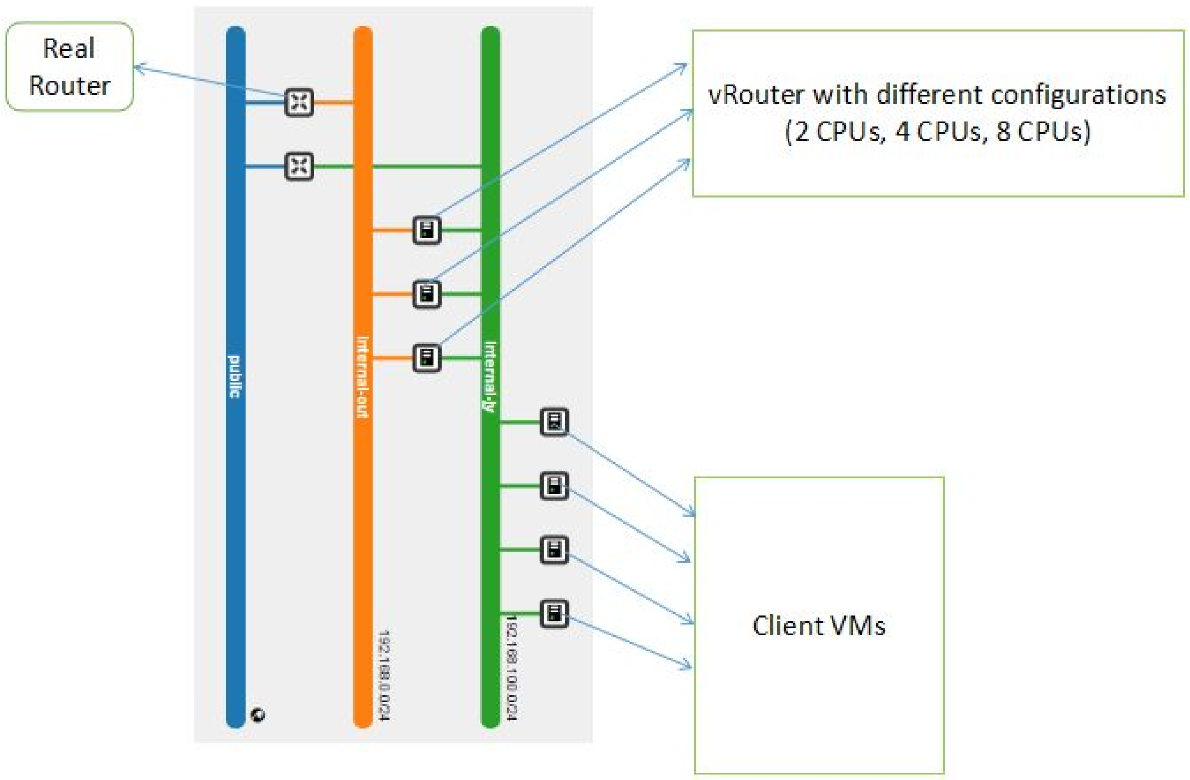
tcp_head = unpack('!HHLLBBHHH', skb[14+ip_head_length:14+ip_head_length+20])

```
tcp_head_dict = dict(zip(tcp_header_fmt, tcp_head))
offset = tcp_head_dict['tcp_offset_res']
tcp_head_length = 4*(offset >> 4)
```

(5) .HTTP Data:

We check whether each packet data is HTTP protocol, if it is, then we parse the HTTP data to have the action, user_agent, result_code, referrer, type, etc., and write to database. If not we simply drop this packet.

P3) Overall Network Topology in MOC



From the diagram above, you can see that vRouters sit between internal-jy(green one) and internal-out(orange one). And the Real-Router is actually sitting between internal-out(orange one) and public(blue one).

How to configure vRouter?

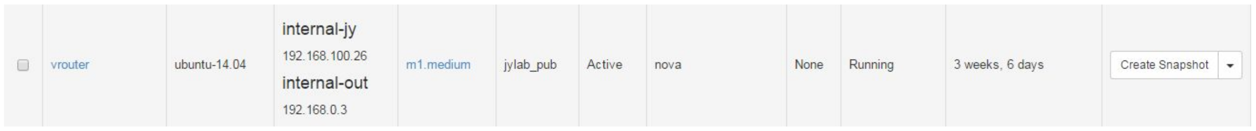
Overview

The Linux kernel usually possesses a packet filter framework called netfilter (Project home: netfilter.org). This framework enables a Linux machine with an appropriate number of network cards (interfaces) to become a router capable of NAT. We will use the command utility 'iptables' to create complex rules for modification and filtering of packets. The important rules regarding

NAT are - not very surprising - found in the 'nat'-table. This table has three predefined chains: PREROUTING, OUTPUT and POSTROUTING. And we will modify these chains.

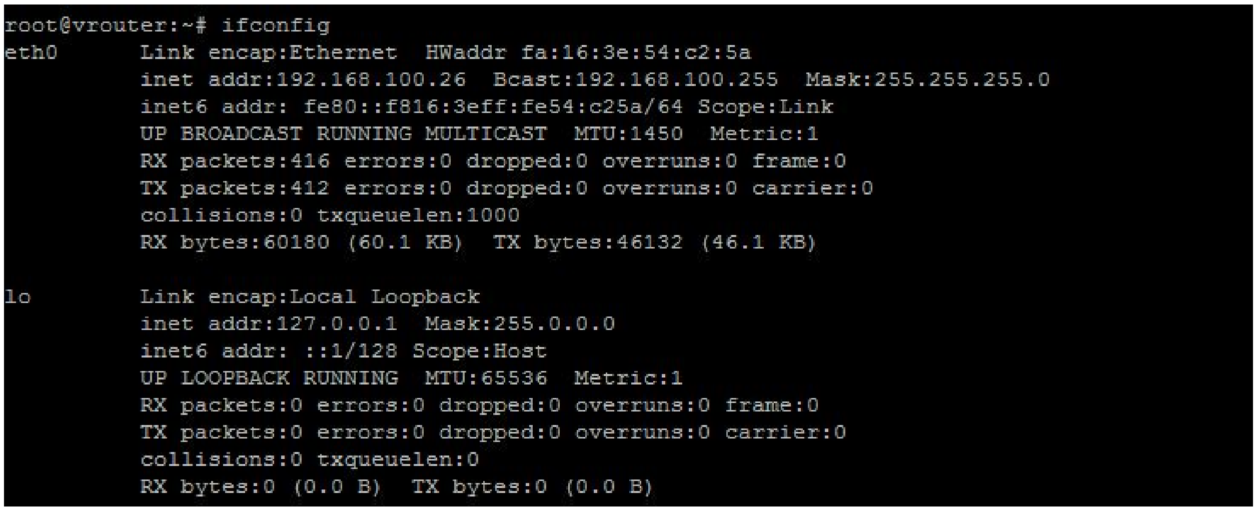
Initial Configuration

There’s a misconfiguration in ubuntu 14.04 img, it can’t detect multiple network interfaces properly in openstack environment. We need to adjust it manually at beginning after creating and booting a VM.



☐	vrouter	ubuntu-14.04	internal-jy 192.168.100.26 internal-out 192.168.0.3	m1.medium	jylab_pub	Active	nova	None	Running	3 weeks, 6 days	Create Snapshot	▾

From the diagram above, we can see that each vRouter has two interfaces(one connects to internal network(192.168.100.26), the other one connects to external network(192.168.0.3)). But here(diagram below), we can only see eth0(the one connects to internal network), we are missing eth1.



```
root@vrouter:~# ifconfig
eth0      Link encap:Ethernet  HWaddr fa:16:3e:54:c2:5a
          inet addr:192.168.100.26  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe54:c25a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:416 errors:0 dropped:0 overruns:0 frame:0
          TX packets:412 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:60180 (60.1 KB)  TX bytes:46132 (46.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Fix Misconfiguration

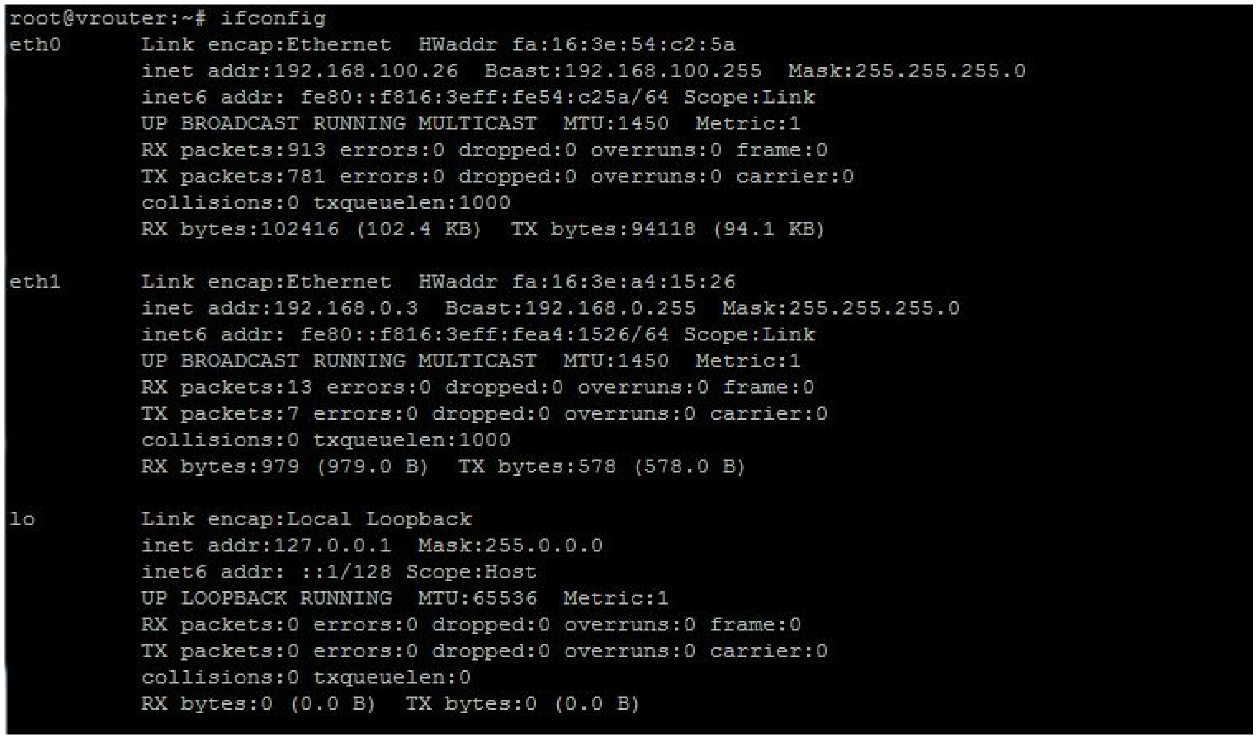
In order to fix it, we need to modify "/etc/network/interfaces" file, please add the green part to this file(according to your IP addresses of interfaces). You will need "sudo" permission to do it.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
      address 192.168.100.26
      netmask 255.255.255.0
      gateway 192.168.100.1

auto eth1
iface eth1 inet static
      address 192.168.0.3
      netmask 255.255.255.0
      gateway 192.168.0.1

# Source interfaces
# Please check /etc/network/interfaces.d before changing this file
# as interfaces may have been defined in /etc/network/interfaces.d
# NOTE: the primary ethernet device is defined in
# /etc/network/interfaces.d/eth0
# See LP: #1262951
source /etc/network/interfaces.d/*.cfg
```

After modifying "/etc/network/interfaces" file, please run the following commands to restart the network and bring up eth1.

```
sudo service networking restart
sudo ifup eth1
sudo ifconfig eth1 mtu 1450 (if it's not 1450, then we will have problem of sshing to this machine via eth1, by default mtu = 1500)
```

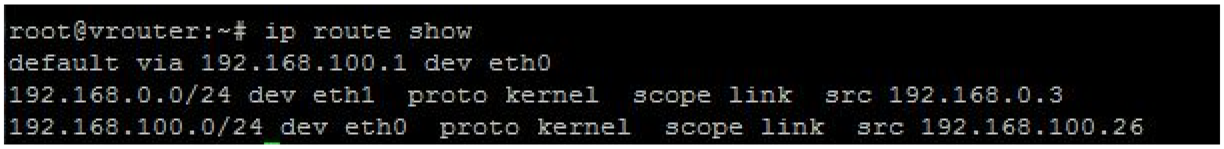
Now you should be able to see eth1 running properly as follows:



```
root@vrouter:~# ifconfig
eth0      Link encap:Ethernet  HWaddr fa:16:3e:54:c2:5a
          inet addr:192.168.100.26  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe54:c25a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:913 errors:0 dropped:0 overruns:0 frame:0
          TX packets:781 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:102416 (102.4 KB)  TX bytes:94118 (94.1 KB)

eth1      Link encap:Ethernet  HWaddr fa:16:3e:a4:15:26
          inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fea4:1526/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:979 (979.0 B)  TX bytes:578 (578.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Change routing rules on vRouter

By default the routing path on vRouter looks like this:



```
root@vrouter:~# ip route show
default via 192.168.100.1 dev eth0
192.168.0.0/24 dev eth1  proto kernel  scope link  src 192.168.0.3
192.168.100.0/24 dev eth0  proto kernel  scope link  src 192.168.100.26
```

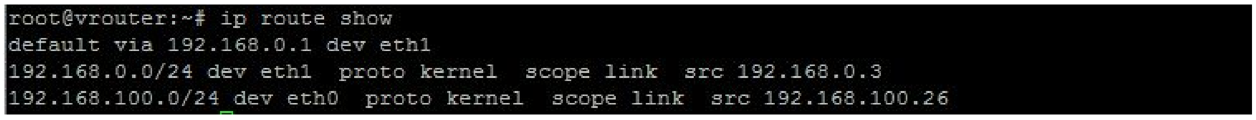
We can see that all traffic go to “192.168.100.1”, we also need to modify the routing to re-direct the traffic to “192.168.0.1” instead of “192.168.100.1”.

---

route del default gw 192.168.100.1

route add default gw 192.168.0.1

After running the commands above, we can re-check the routing table as follows:



```
root@vrouter:~# ip route show
default via 192.168.0.1 dev eth1
192.168.0.0/24 dev eth1  proto kernel  scope link  src 192.168.0.3
192.168.100.0/24 dev eth0  proto kernel  scope link  src 192.168.100.26
```

Further Configuration

Here are the commands that we need to run on vRouter:

#This command runs "bash" as a super user.

sudo bash

#(To determine a rule's line number, list the rules in the table format and add the `--line-numbers` option)

# sudo iptables -L --line-numbers

```
sudo iptables -P FORWARD DROP
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -A POSTROUTING -s 192.168.100.0/24 -j SNAT --to 192.168.0.3
iptables -A FORWARD -s 192.168.100.22 -j ACCEPT

iptables -t nat -A PREROUTING -d 192.168.0.3  -j DNAT --to 192.168.100.22
iptables -t nat -A PREROUTING -d 192.168.0.3 -p tcp --dport 80 -j DNAT --to 192.168.100.22:80
iptables -A FORWARD -d 192.168.0.3 -p tcp --dport 80 -j ACCEPT

sysctl net.ipv4.ip_forward=1
```

How to configure from MOC OpenStack side?

Problem

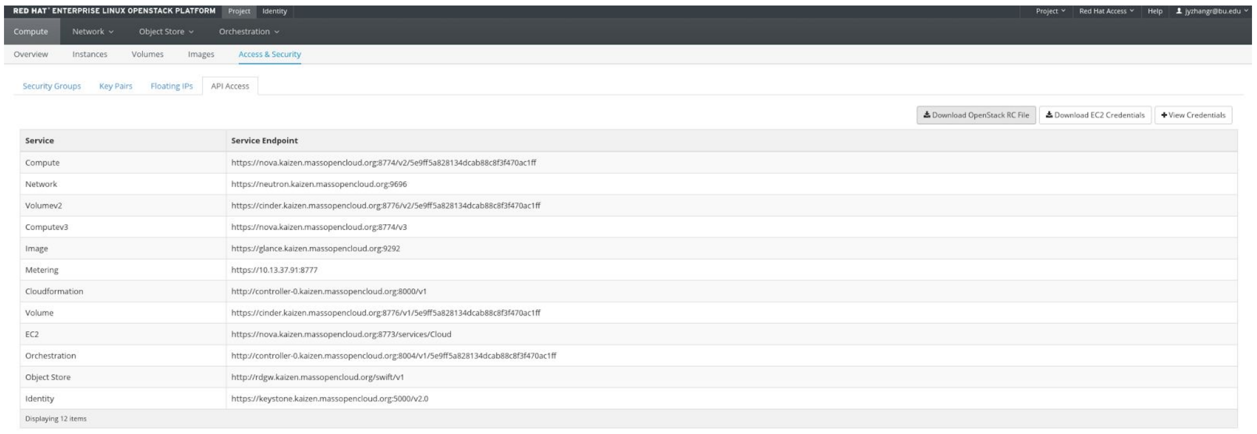
There is a security-feature issue in OpenStack. "Source verification" is done and you are not allowed to send forged packets out by default. In our case, the REPLY/RESPONSE packets were going out from the router to the destined machine, but were getting dropped in-between by OpenStack's security group since the source-ip was not the same as the interface's ip-address.

It only allows packets whose IP-address matches with the IP-address of the nic (attached to that network).

Let's say you were trying to ping 10.2.0.XX from 192.168.0.YY, then the reply packets will have source-ip as 10.2.0.XX and router will change the mac-addresses. When it goes out from the router's interface (192.168.0.YY), the packet has source-ip as 10.2.0.XX and the security-group rules will drop those packets.

Solution:

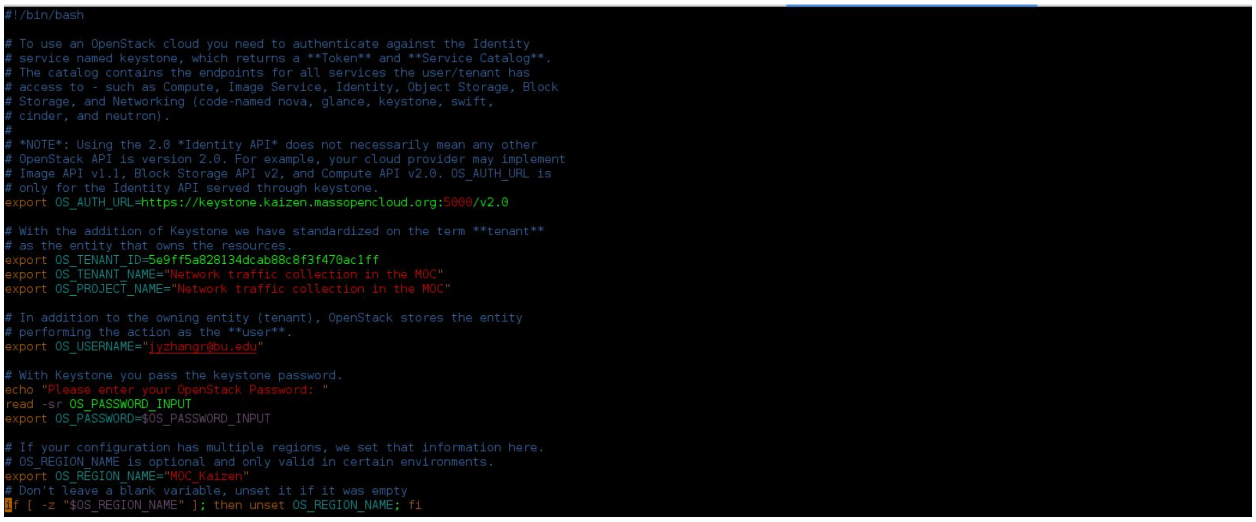
Follow the diagram below, download the “OpenStack RC File” under “Access & Security” section:



Service	Service Endpoint
Compute	https://nova.kaizen.massopencloud.org:8774/v2/5e9ff5a828134dcab88c8f3f470ac1ff
Network	https://neutron.kaizen.massopencloud.org:9696
Volumev2	https://cinder.kaizen.massopencloud.org:8776/v2/5e9ff5a828134dcab88c8f3f470ac1ff
Computev3	https://nova.kaizen.massopencloud.org:8774/v3
Image	https://glance.kaizen.massopencloud.org:9292
Metering	https://10.13.37.91:8777
Cloudformation	http://controller-0.kaizen.massopencloud.org:8000/v1
Volume	https://cinder.kaizen.massopencloud.org:8776/v1/5e9ff5a828134dcab88c8f3f470ac1ff
EC2	https://nova.kaizen.massopencloud.org:8773/services/Cloud
Orchestration	http://controller-0.kaizen.massopencloud.org:8004/v1/5e9ff5a828134dcab88c8f3f470ac1ff
Object Store	http://rdgw.kaizen.massopencloud.org/swift/v1
Identity	https://keystone.kaizen.massopencloud.org:5000/v2.0

Displaying 12 items

The file looks like:



```
#!/bin/bash

# To use an OpenStack cloud you need to authenticate against the Identity
# service named keystone, which returns a **Token** and **Service Catalog**.
# The catalog contains the endpoints for all services the user/tenant has
# access to - such as Compute, Image Service, Identity, Object Storage, Block
# Storage, and Networking (code-named nova, glance, keystone, swift,
# cinder, and neutron).
#
# *NOTE*: Using the 2.0 *Identity API* does not necessarily mean any other
# OpenStack API is version 2.0. For example, your cloud provider may implement
# Image API v1.1, Block Storage API v2, and Compute API v2.0. OS_AUTH_URL is
# only for the Identity API served through keystone.
export OS_AUTH_URL=https://keystone.kaizen.massopencloud.org:5000/v2.0

# With the addition of Keystone we have standardized on the term **tenant**
# as the entity that owns the resources.
export OS_TENANT_ID=5e9ff5a828134dcab88c8f3f470ac1ff
export OS_TENANT_NAME="Network traffic collection in the MOC"
export OS_PROJECT_NAME="Network traffic collection in the MOC"

# In addition to the owning entity (tenant), OpenStack stores the entity
# performing the action as the **user**.
export OS_USERNAME="jyzhangr@bu.edu"

# With Keystone you pass the keystone password.
echo "Please enter your OpenStack Password: "
read -sr OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT

# If your configuration has multiple regions, we set that information here.
# OS_REGION_NAME is optional and only valid in certain environments.
export OS_REGION_NAME="MOC_Kaizen"
# Don't leave a blank variable, unset it if it was empty
if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
```

Upload the file to vRouter and run the following commands:

---

source <filename>.sh # it will ask you to put your MOC account password

#Then you need to install “python-neutronclient” on the vRouter, in Ubuntu you can use apt-get

sudo apt-get install python-neutronclient

#After install “python-neutronclient”, you will be able to run neutron commands on vRouter

#Display the nets

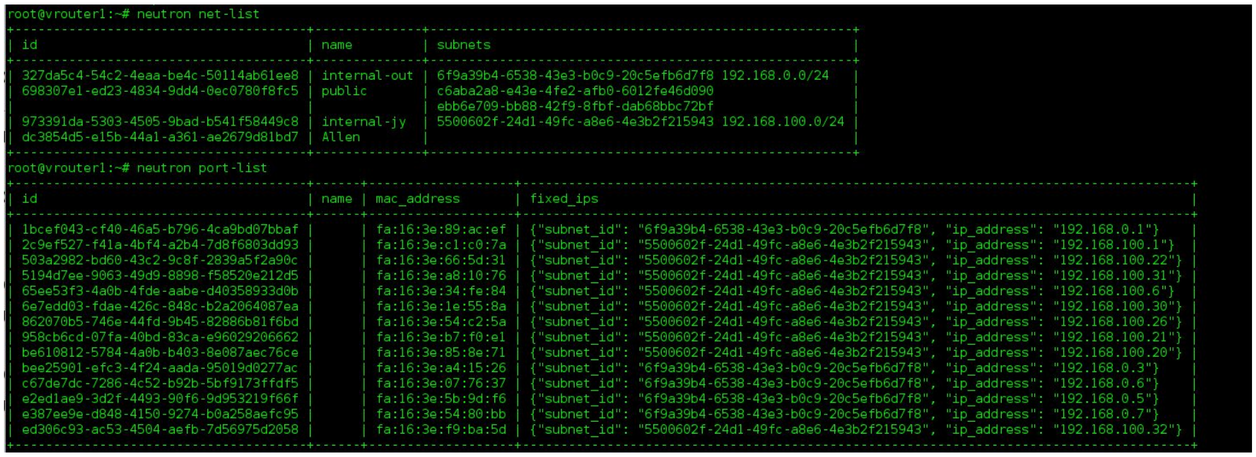
neutron net-list

#Display the ports

neutron port-list

Diagrams

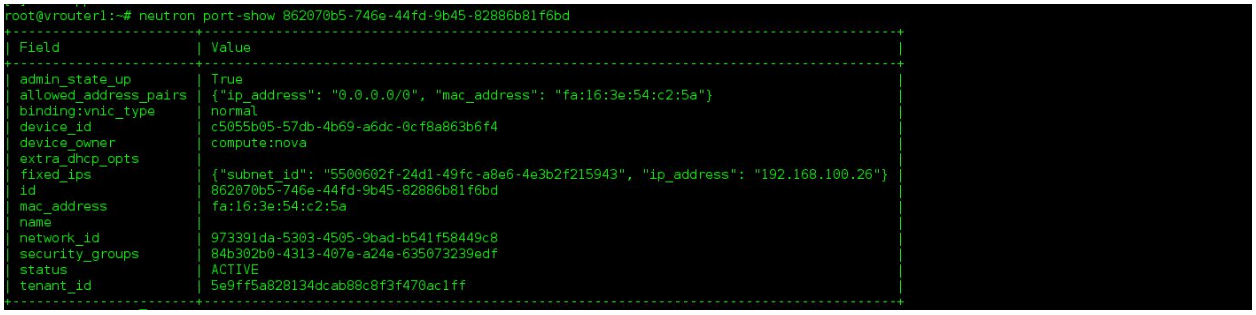
Please refer to the diagram below for networks and ports information:



```
root@vrouter1:~# neutron net-list
+--------------------------------------+--------------+-----------------------------------------------------+
| id                                   | name         | subnets                                             |
+--------------------------------------+--------------+-----------------------------------------------------+
| 327da5c4-54c2-4eaa-be4c-50114ab61ee8 | internal-out | 6f9a39b4-6538-43e3-b0c9-20c5efb6d7f8 192.168.0.0/24   |
| 698307e1-ed23-4834-9dd4-0ec0780f8fc5 | public       | c6aba2a8-e43e-4fe2-afb0-6012fe46d090                |
|                                      |              | ebb6e709-bb88-42f9-8fbf-dab68bbc72bf                |
| 973391da-5303-4505-9bad-b541f58449c8 | internal-jy  | 5500602f-24d1-49fc-a8e6-4e3b2f215943 192.168.100.0/24 |
| dc3854d5-e15b-44a1-a361-ae2679d81bd7 | Allen        |                                                     |
+--------------------------------------+--------------+-----------------------------------------------------+
root@vrouter1:~# neutron port-list
+--------------------------------------+------+-------------------+--------------------------------------------------------------------------------------+
| id                                   | name | mac_address       | fixed_ips                                                                            |
+--------------------------------------+------+-------------------+--------------------------------------------------------------------------------------+
| 1bcef043-cf40-46a5-b796-4ca9bd07bbaf |      | fa:16:3e:89:ac:ef | {"subnet_id": "6f9a39b4-6538-43e3-b0c9-20c5efb6d7f8", "ip_address": "192.168.0.1"}    |
| 2c9ef527-f41a-4bf4-a2b4-7d8f6803dd93 |      | fa:16:3e:c1:c0:7a | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.1"}  |
| 503a2982-bd60-43c2-9c8f-2839a5f2a90c |      | fa:16:3e:66:5d:31 | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.22"} |
| 5194d7ee-9063-49d9-8898-f58520e212d5 |      | fa:16:3e:a8:10:76 | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.31"} |
| 65ee53f3-4a0b-4fde-aabe-d40358933d0b |      | fa:16:3e:34:fe:84 | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.6"}  |
| 6e7edd03-fdae-426c-848c-b2a2064087ea |      | fa:16:3e:1e:55:8a | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.30"} |
| 862070b5-746e-44fd-9b45-82886b81f6bd |      | fa:16:3e:54:c2:5a | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.26"} |
| 958cb6cd-07fa-40bd-83ca-e96029206662 |      | fa:16:3e:b7:f0:e1 | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.21"} |
| be610812-5784-4a0b-b403-8e087aec76ce |      | fa:16:3e:85:8e:71 | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.20"} |
| bee25901-efc3-4f24-aada-95019d0277ac |      | fa:16:3e:a4:15:26 | {"subnet_id": "6f9a39b4-6538-43e3-b0c9-20c5efb6d7f8", "ip_address": "192.168.0.3"}    |
| c67de7dc-7286-4c52-b92b-5bf9173ffdf5 |      | fa:16:3e:07:76:37 | {"subnet_id": "6f9a39b4-6538-43e3-b0c9-20c5efb6d7f8", "ip_address": "192.168.0.6"}    |
| e2ed1ae9-3d2f-4493-90f6-9d953219f66f |      | fa:16:3e:5b:9d:f6 | {"subnet_id": "6f9a39b4-6538-43e3-b0c9-20c5efb6d7f8", "ip_address": "192.168.0.5"}    |
| e387ee9e-d848-4150-9274-b0a258aefc95 |      | fa:16:3e:54:80:bb | {"subnet_id": "6f9a39b4-6538-43e3-b0c9-20c5efb6d7f8", "ip_address": "192.168.0.7"}    |
| ed306c93-ac53-4504-aefb-7d56975d2058 |      | fa:16:3e:f9:ba:5d | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.32"} |
+--------------------------------------+------+-------------------+--------------------------------------------------------------------------------------+
```

We can see that our port(192.168.100.26) id is:”862070b5-746e-44fd-9b45-82886b81f6bd”

Please make sure that “allow_address_pairs” looks like the diagram below:



```
root@vrouter1:~# neutron port-show 862070b5-746e-44fd-9b45-82886b81f6bd
+-----------------------+---------------------------------------------------------------------------------+
| Field                 | Value                                                                           |
+-----------------------+---------------------------------------------------------------------------------+
| admin_state_up        | True                                                                            |
| allowed_address_pairs | {"ip_address": "0.0.0.0/0", "mac_address": "fa:16:3e:54:c2:5a"}                 |
| binding:vnic_type     | normal                                                                          |
| device_id             | c5055b05-57db-4b69-a6dc-0cf8a863b6f4                                            |
| device_owner          | compute:nova                                                                    |
| extra_dhcp_opts       |                                                                                 |
| fixed_ips             | {"subnet_id": "5500602f-24d1-49fc-a8e6-4e3b2f215943", "ip_address": "192.168.100.26"} |
| id                    | 862070b5-746e-44fd-9b45-82886b81f6bd                                            |
| mac_address           | fa:16:3e:54:c2:5a                                                               |
| name                  |                                                                                 |
| network_id            | 973391da-5303-4505-9bad-b541f58449c8                                            |
| security_groups       | 84b302b0-4313-407e-a24e-635073239edf                                            |
| status                | ACTIVE                                                                          |
| tenant_id             | 5e9ff5a828134dcab88c8f3f470ac1ff                                                |
+-----------------------+---------------------------------------------------------------------------------+
```

If not, you can update it by using command:

---

neutron port-update <port-id> --allowed_address_pairs type=dict list=true ip_address='0.0.0.0/0'

P4) Enabling TaaS at devStack

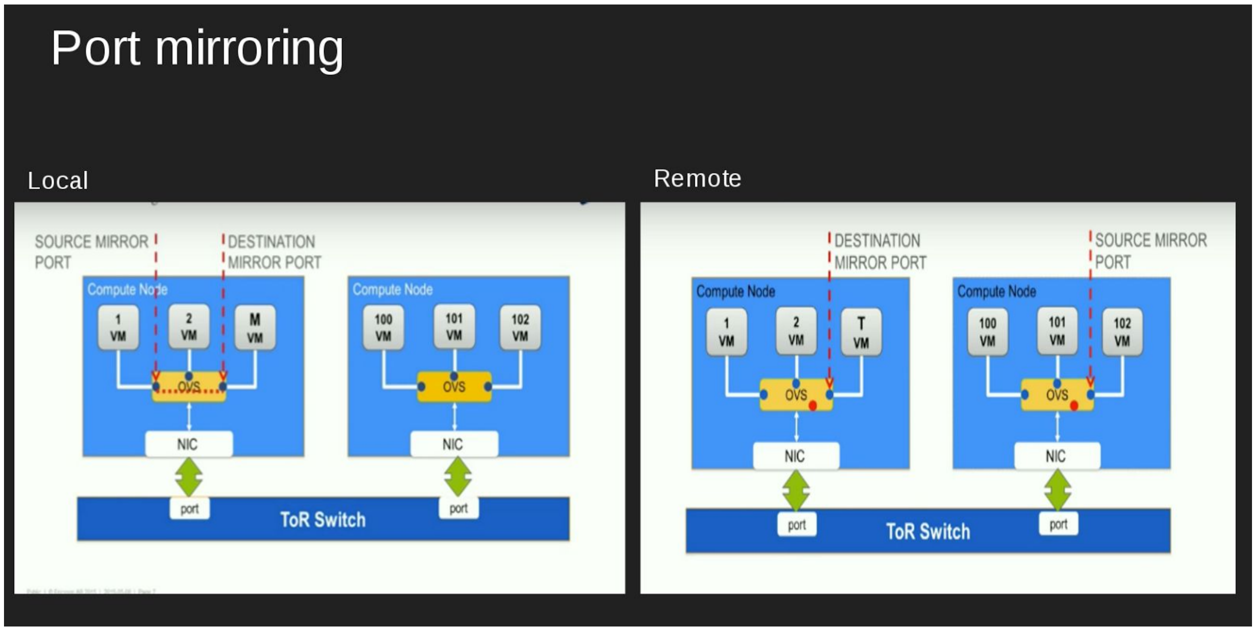
Introduction

Tap-as-a-Service (TaaS) is an extension to the OpenStack network service (Neutron). 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, which is usually different from the original destinations of the packets being mirrored.

This service has been primarily designed to help tenants (or the cloud administrator) debug complex virtual networks and gain visibility into their VMs, by monitoring the network traffic associated with them. TaaS honors tenant boundaries and its mirror sessions are capable of spanning across multiple compute and network nodes. It serves as an essential infrastructure component that can be utilized for supplying data to a variety of network analytics and security applications (e.g. IDS).

Port mirroring



Enable TaaS in devStack

Enable TaaS in devStack is pretty simply and straightforward. You need to insert the following lines of codes to local.conf file of devStack

---

enable_plugin tap-as-a-service https://github.com/openstack/tap-as-a-service

enable_service taas

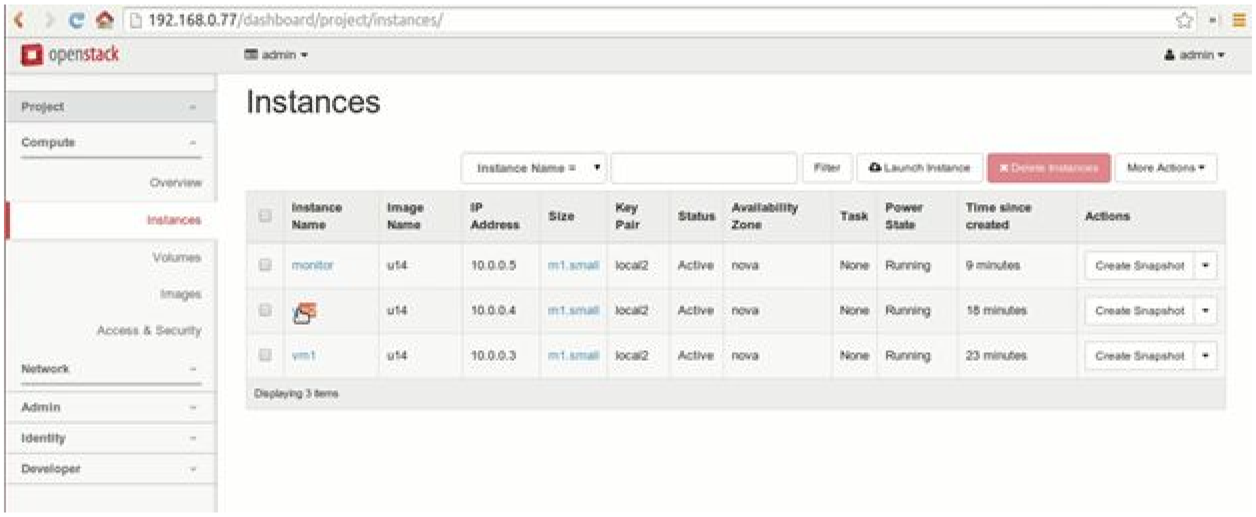
enable_service taas_openvswitch_agent

# Because TaaS requires the 'Port Security' Neutron ML2 extension. We need to make sure that # it’s enabled. Adding the following to 'local.conf' while installing DevStack will enable 'Port #Security' extension. (It's enabled by default)

Q_ML2_PLUGIN_EXT_DRIVERS=port_security

How to create tap service and tap flow

Our System overview:

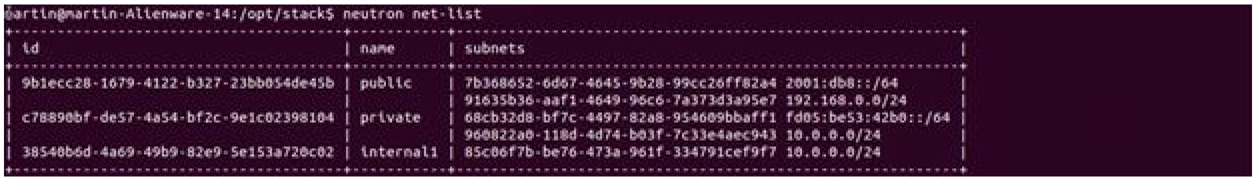


Instances

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
monitor	u14	10.0.0.5	m1.small	local2	Active	nova	None	Running	9 minutes	Create Snapshot
[illegible]	u14	10.0.0.4	m1.small	local2	Active	nova	None	Running	18 minutes	Create Snapshot
vm1	u14	10.0.0.3	m1.small	local2	Active	nova	None	Running	23 minutes	Create Snapshot

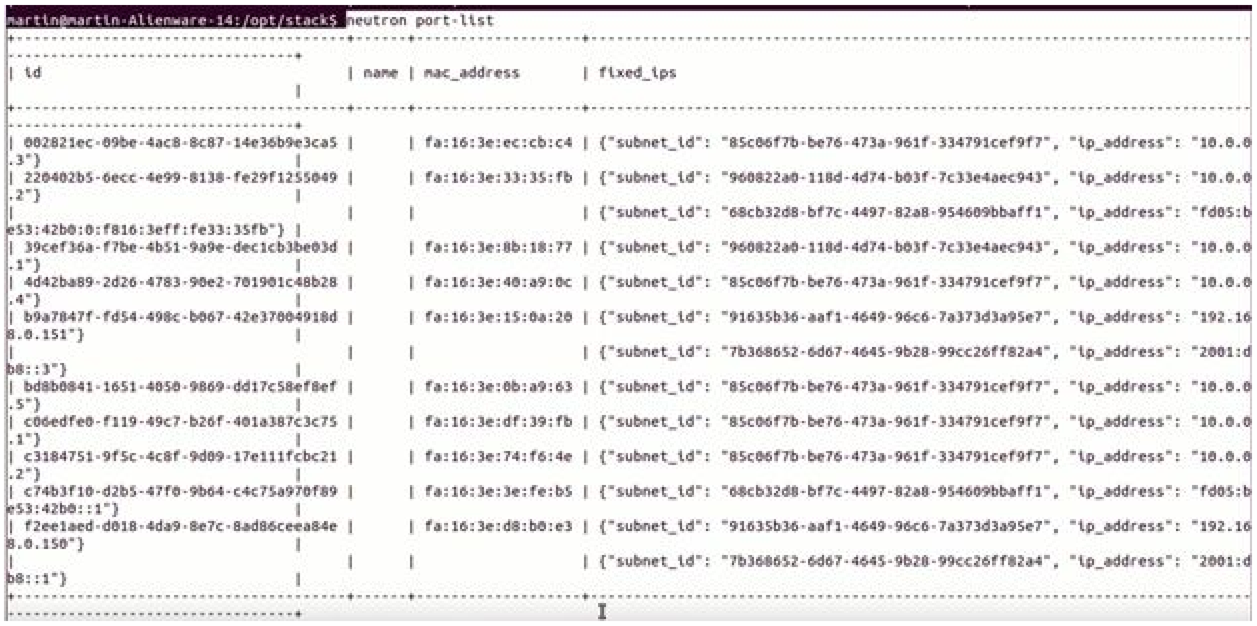
Displaying 3 items

Display networks in our devStack deployment:



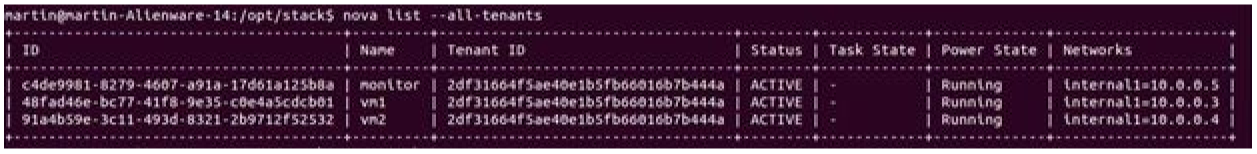
```
martin@martin-Alienware-14:/opt/stack$ neutron net-list
+--------------------------------------+-----------+------------------------------------------------------+
| id                                   | name      | subnets                                              |
+--------------------------------------+-----------+------------------------------------------------------+
| 9b1ecc28-1679-4122-b327-23bb054de45b | public    | 7b368652-6d67-4645-9b28-99cc26ff82a4 2001:db8::/64   |
|                                      |           | 91635b36-aaf1-4649-96c6-7a373d3a95e7 192.168.0.0/24  |
| c78890bf-de57-4a54-bf2c-9e1c02398104 | private   | 68cb32d8-bf7c-4497-82a8-954609bbaff1 fd05:be53:42b0::/64 |
|                                      |           | 960822a0-118d-4d74-b03f-7c33e4aec943 10.0.0.0/24     |
| 38540b6d-4a69-49b9-82e9-5e153a720c02 | internal1 | 85c06f7b-be76-473a-961f-334791cef9f7 10.0.0.0/24     |
+--------------------------------------+-----------+------------------------------------------------------+
```

Display ports information of openvSwitch:



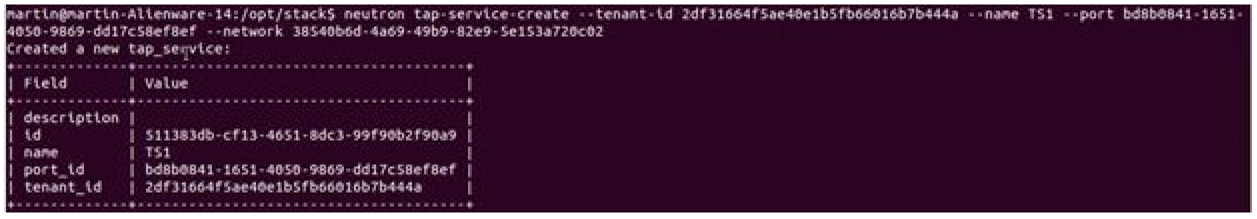
```
martin@martin-Alienware-14:/opt/stack$ neutron port-list
+--------------------------------------+------+-------------------+-----------------------------------------------------------------------------------
--------------------------------+
| id                                   | name | mac_address       | fixed_ips
                                |
+--------------------------------------+------+-------------------+-----------------------------------------------------------------------------------
--------------------------------+
| 002821ec-09be-4ac8-8c87-14e36b9e3ca5 |      | fa:16:3e:ec:cb:c4 | {"subnet_id": "85c06f7b-be76-473a-961f-334791cef9f7", "ip_address": "10.0.0
.3"}                            |
| 220402b5-6ecc-4e99-8138-fe29f1255049 |      | fa:16:3e:33:35:fb | {"subnet_id": "960822a0-118d-4d74-b03f-7c33e4aec943", "ip_address": "10.0.0
.2"}                            |
|                                      |      |                   | {"subnet_id": "68cb32d8-bf7c-4497-82a8-954609bbaff1", "ip_address": "fd05:b
e53:42b0:0:f816:3eff:fe33:35fb"} |
| 39cef36a-f7be-4b51-9a9e-dec1cb3be03d |      | fa:16:3e:8b:18:77 | {"subnet_id": "960822a0-118d-4d74-b03f-7c33e4aec943", "ip_address": "10.0.0
.1"}                            |
| 4d42ba89-2d26-4783-90e2-701901c48b28 |      | fa:16:3e:40:a9:0c | {"subnet_id": "85c06f7b-be76-473a-961f-334791cef9f7", "ip_address": "10.0.0
.4"}                            |
| b9a7847f-fd54-498c-b067-42e37004918d |      | fa:16:3e:15:0a:20 | {"subnet_id": "91635b36-aaf1-4649-96c6-7a373d3a95e7", "ip_address": "192.16
8.0.151"}                       |
|                                      |      |                   | {"subnet_id": "7b368652-6d67-4645-9b28-99cc26ff82a4", "ip_address": "2001:d
b8::3"}                         |
| bd8b0841-1651-4050-9869-dd17c58ef8ef |      | fa:16:3e:0b:a9:63 | {"subnet_id": "85c06f7b-be76-473a-961f-334791cef9f7", "ip_address": "10.0.0
.5"}                            |
| c06edfe0-f119-49c7-b26f-401a387c3c75 |      | fa:16:3e:df:39:fb | {"subnet_id": "85c06f7b-be76-473a-961f-334791cef9f7", "ip_address": "10.0.0
.1"}                            |
| c3184751-9f5c-4c8f-9d09-17e111fcbc21 |      | fa:16:3e:74:f6:4e | {"subnet_id": "85c06f7b-be76-473a-961f-334791cef9f7", "ip_address": "10.0.0
.2"}                            |
| c74b3f10-d2b5-47f0-9b64-c4c75a970f89 |      | fa:16:3e:3e:fe:b5 | {"subnet_id": "68cb32d8-bf7c-4497-82a8-954609bbaff1", "ip_address": "fd05:b
e53:42b0::1"}                   |
| f2ee1aed-d018-4da9-8e7c-8ad86ceea84e |      | fa:16:3e:d8:b0:e3 | {"subnet_id": "91635b36-aaf1-4649-96c6-7a373d3a95e7", "ip_address": "192.16
8.0.150"}                       |
|                                      |      |                   | {"subnet_id": "7b368652-6d67-4645-9b28-99cc26ff82a4", "ip_address": "2001:d
b8::1"}                         |
+--------------------------------------+------+-------------------+-----------------------------------------------------------------------------------
--------------------------------+
```

Display tenants information:



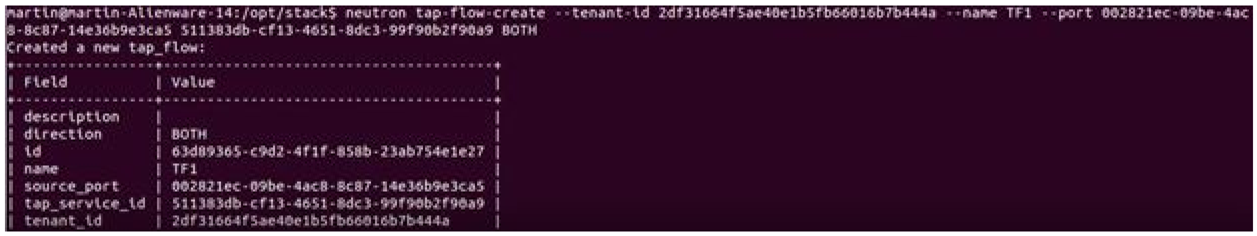
```
martin@martin-Alienware-14:/opt/stack$ nova list --all-tenants
+--------------------------------------+---------+----------------------------------+--------+------------+-------------+--------------------+
| ID                                   | Name    | Tenant ID                        | Status | Task State | Power State | Networks           |
+--------------------------------------+---------+----------------------------------+--------+------------+-------------+--------------------+
| c4de9981-8279-4607-a91a-17d61a125b8a | monitor | 2df31664f5ae40e1b5fb66016b7b444a | ACTIVE | -          | Running     | internal1=10.0.0.5 |
| 48fad46e-bc77-41f8-9e35-c0e4a5cdcb01 | vm1     | 2df31664f5ae40e1b5fb66016b7b444a | ACTIVE | -          | Running     | internal1=10.0.0.3 |
| 91a4b59e-3c11-493d-8321-2b9712f52532 | vm2     | 2df31664f5ae40e1b5fb66016b7b444a | ACTIVE | -          | Running     | internal1=10.0.0.4 |
+--------------------------------------+---------+----------------------------------+--------+------------+-------------+--------------------+
```

Create a tap service:



```
martin@martin-Alienware-14:/opt/stack$ neutron tap-service-create --tenant-id 2df31664f5ae40e1b5fb66016b7b444a --name TS1 --port bd8b0841-1651-
4050-9869-dd17c58ef8ef --network 38540b6d-4a69-49b9-82e9-5e153a720c02
Created a new tap_service:
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| description |                                      |
| id          | 511383db-cf13-4651-8dc3-99f90b2f90a9 |
| name        | TS1                                  |
| port_id     | bd8b0841-1651-4050-9869-dd17c58ef8ef |
| tenant_id   | 2df31664f5ae40e1b5fb66016b7b444a     |
+-------------+--------------------------------------+
```

Create a tap flow:



```
martin@martin-Alienware-14:/opt/stack$ neutron tap-flow-create --tenant-id 2df31664f5ae40e1b5fb66016b7b444a --name TF1 --port 002821ec-09be-4ac
8-8c87-14e36b9e3ca5 511383db-cf13-4651-8dc3-99f90b2f90a9 BOTH
Created a new tap_flow:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| description    |                                      |
| direction      | BOTH                                 |
| id             | 63d89365-c9d2-4f1f-858b-23ab754e1e27 |
| name           | TF1                                  |
| source_port    | 002821ec-09be-4ac8-8c87-14e36b9e3ca5 |
| tap_service_id | 511383db-cf13-4651-8dc3-99f90b2f90a9 |
| tenant_id      | 2df31664f5ae40e1b5fb66016b7b444a     |
```

P5) Detection

(1). Introduction:

The purpose of our program's collection of HTTP requests through the vRouter is to detect malicious traffic by applying our own behavioral model on all requests stored in InfluxDB. In this final version of our program, we have implemented rules that are capable of detecting nine different kinds of malware using different detection mechanisms. The results of detection are then finally written to a separate database in order to allow for further action to occur, such as blocking VM's that behave suspiciously. We will get in depth into each part of this process separately.

The 3rd party libraries that this script relies on are ciso8601 (https://pypi.python.org/pypi/ciso8601/1.0.1) for timestamp conversions and the influxdb python client (https://github.com/influxdata/influxdb-python).

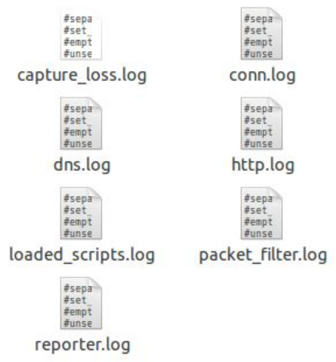
(2). Bro and the Malicious Pcap Samples:

We began our process of implementation by first attempting to identify patterns in the HTTP header of requests in order to make rules for their detection. In order to accomplish this, we began by using Bro. Bro is a robust network security monitor that enables us to easily parse pcap files and then produce logs that show each field in the HTTP header. You may use Bro by first downloading the newest version of Security Onion from: http://blog.securityonion.net/2014/02/security-onion-12044-iso-image-now.html

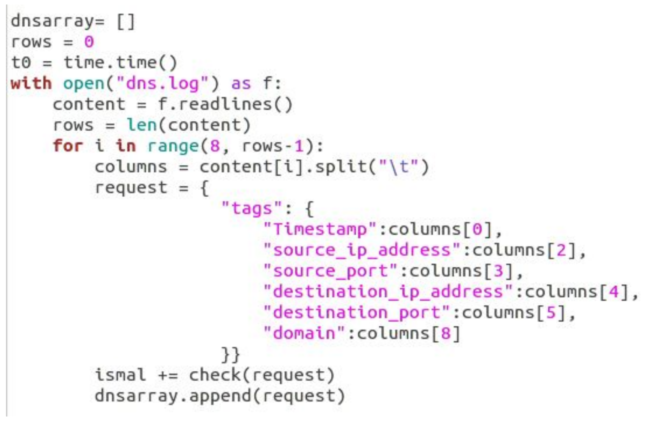
After running the ISO image with VirtualBox or an equivalent, you can run Bro by executing the following command:

bro –r sample.pcap local

Where sample.pcap is the name of the pcap file that you wish to analyze. Running this command produces the following logs, which can then be parsed separately in order to run different kinds of analyses and establish rules and thresholds for each kind of malware:



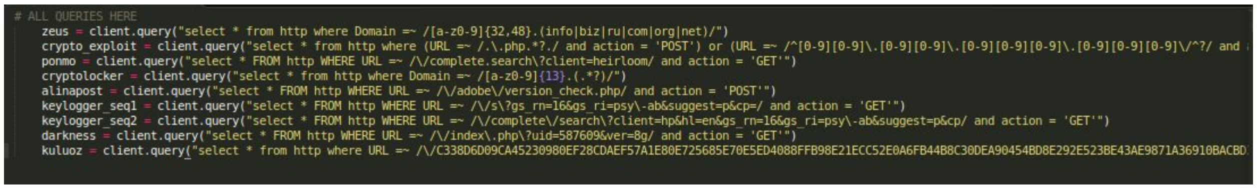
Here is a sample code snippet for parsing the dns.log file in order to look at strange behavior of malware in the Domain HTTP header field:


```
dnsarray= []
rows = 0
t0 = time.time()
with open("dns.log") as f:
    content = f.readlines()
    rows = len(content)
    for i in range(8, rows-1):
        columns = content[i].split("\t")
        request = {
                    "tags": {
                        "Timestamp":columns[0],
                        "source_ip_address":columns[2],
                        "source_port":columns[3],
                        "destination_ip_address":columns[4],
                        "destination_port":columns[5],
                        "domain":columns[8]
                    }}
        ismal += check(request)
        dnsarray.append(request)
```

Through the analysis of these logs, new rules and patterns can be found that can help increase the accuracy of the system. The behavioral model is structured in a way that allows for continuous updates that widen the scope of the program modularly in the future.
For an introduction to Bro that explains how the software can be used to find these rules and patterns, refer to the following link, which uses Zeus as an example:
http://blog.opensecurityresearch.com/2014/03/identifying-malware-traffic-with-bro.html

(3). Malware Types:

As previously stated, we have implemented detection mechanisms that can find 9 different malware types, some relying on methods more complicated than others. Below is a snapshot of the regexes that we used in our implementation. The results from these queries are used in order to perform a more in depth analysis on every suspicious request. All pcap traffic samples that we do not provide links to are available on our github repo in PCAP_SAMPLES, coded by number, along with the Bro logs for each pcap sample in a separate folder.



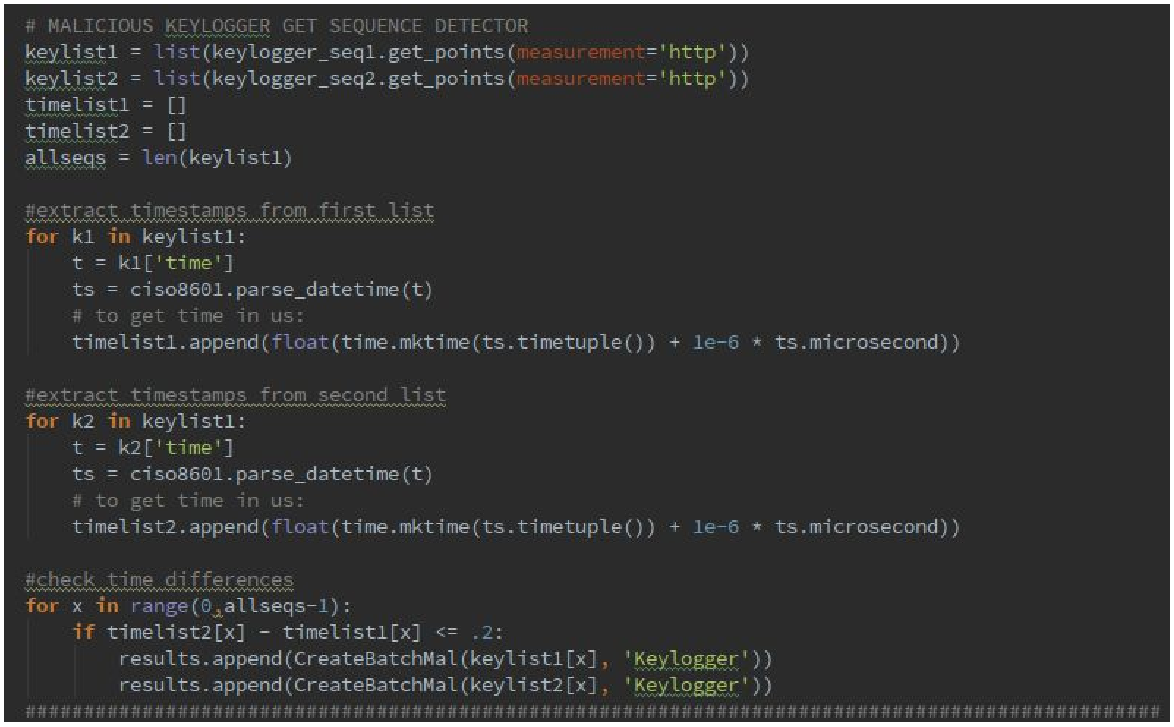
```
# ALL QUERIES HERE
    zeus = client.query("select * from http where Domain =~ /[a-z0-9]{32,48}.(info|biz|ru|com|org|net)/")
    crypto_exploit = client.query("select * from http where (URL =~ /.\.php.*?./ and action = 'POST') or (URL =~ /^[0-9][0-9]\.[0-9][0-9]\.[0-9][0-9][0-9]\.[0-9][0-9][0-9]\/^?/ and
    ponmo = client.query("select * FROM http WHERE URL =~ /\/complete.search\?client=heirloom/ and action = 'GET'")
    cryptolocker = client.query("select * from http where Domain =~ /[a-z0-9]{13}.(.*?)/")
    alinapost = client.query("select * FROM http WHERE URL =~ /\/adobe\/version_check.php/ and action = 'POST'")
    keylogger_seq1 = client.query("select * FROM http WHERE URL =~ /\/s\?gs_rn=16&gs_ri=psy\-ab&suggest=p&cp=/ and action = 'GET'")
    keylogger_seq2 = client.query("select * FROM http WHERE URL =~ /\/complete\/search\?client=hp&hl=en&gs_rn=16&gs_ri=psy\-ab&suggest=p&cp/ and action = 'GET'")
    darkness = client.query("select * FROM http WHERE URL =~ /\/index\.php\?uid=587609&ver=8g/ and action = 'GET'")
    kuluoz = client.query("select * from http where URL =~ /\/C338D6D09CA45230980EF28CDAEF57A1E80E725685E70E5ED4088FFB98E21ECC52E0A6FB44B8C30DEA90454BD8E292E523BE43AE9871A36910BACBD
```

1. **<u>Zeus</u>** : By observing the Domain name from the DNS bro logs, you can see that the TLD is always one of 5 different countries, and the second level domain is a randomly generated string of length between 32 and 48. In order to determine randomness, we calculate the Shannon Character Entropy of the string (as we will soon get into), and compare this float value against a threshold that was determined by calculating the entropies of the SLD's from the Bro logs generated by using a Zeus malware traffic sample. For Zeus, this value would be between **4.2195282823** and **4.55305590733.** As you can see in the image above, the regex that was used to find suspicious Zeus Domains is: **/[a-z0-9]{32,48}.(info|biz|ru|com|org|net)/ .** The pcap sample can be obtained from the Bro tutorial link.
2. **Exploit Kit and Cryptowall:** The rules for these two malware types depend on both the request "method" field and the detection of random parameters in the URL. The regex that pulls all GET (Exploit Kit) and POST (Cryptowall) traffic is:
   a. GET: URL=~ **/^[0-9][0-9]\.[0-9][0-9]\.[0-9][0-9][0-9]\.[0-9][0-9][0-9]\/^?/**
   b. POST: URL=~ **/.\.php.*?./**

   After identifying these patterns, the entropy calculator is used to identify randomness in the URL. The entropy thresholds were exactly the same for both Exploit Kit and Cryptowall (!) and lie between **3.7735572622751845** and **2.921928094887362.** The pcap traffic samples can be obtained from: http://www.malware-traffic-analysis.net/2015/07/24/index.html and http://www.malware-traffic-analysis.net/2015/05/08/index.html .
3. **Ponmocup:** This one was easier to implement than the rest, relying mainly on a regex that identifies a pattern in the URL of specific GET requests using the regex: **/\/complete.search\?client=heirloom/ . (sample4.pcap)**
4. **Cryptolocker:** This malware type works the same as Zeus, except the SLD is a randomly generated string with 13 characters in length and the entropy boundaries are

3.08505510276 and **3.3232314288**, based on calculations using the Bro DNS log file. (sample5.pcap)

5. **Malicious Periodic Post Traffic:** This malware type is the first of our rules that takes advantage of our periodicity detector, generating periodic malicious POST traffic with the regex for the URL being: **/\/adobe\/version_check.php/.** Again, using the Bro logs, the variance threshold for the variance in the timing interval between each of these POST requests from the period is **0.030604491454.**
6. **Keylogger:** This malware type uses regexes to search for patterns in the URL's of a



```
# MALICIOUS KEYLOGGER GET SEQUENCE DETECTOR
keylist1 = list(keylogger_seq1.get_points(measurement='http'))
keylist2 = list(keylogger_seq2.get_points(measurement='http'))
timelist1 = []
timelist2 = []
allseqs = len(keylist1)

#extract timestamps from first list
for k1 in keylist1:
    t = k1['time']
    ts = ciso8601.parse_datetime(t)
    # to get time in us:
    timelist1.append(float(time.mktime(ts.timetuple()) + 1e-6 * ts.microsecond))

#extract timestamps from second list
for k2 in keylist1:
    t = k2['time']
    ts = ciso8601.parse_datetime(t)
    # to get time in us:
    timelist2.append(float(time.mktime(ts.timetuple()) + 1e-6 * ts.microsecond))

#check time differences
for x in range(0,allseqs-1):
    if timelist2[x] - timelist1[x] <= .2:
        results.append(CreateBatchMal(keylist1[x], 'Keylogger'))
        results.append(CreateBatchMal(keylist2[x], 'Keylogger'))
##################################################################################################
```

sequence of GET requests that are very close in time.. (this difference in time was found to be never more than .2 seconds, and thus time intervals between the two GETs that are larger are considered not malware according to our behavioral model. The regex of the URL for the first GET in the sequence is: **/\/s\?gs_rn=16&gs_ri=psy\-ab&suggest=p&cp=/** , while the regex for the second GET in the sequence is: **/\/complete\/search\?client=hp&hl=en&gs_rn=16&gs_ri=psy\-ab&suggest=p&cp/** .

7. **Darkness:** Also uses the periodicity detector for all GET requests with URLs that match the regex: **/\/index\.php\?uid=587609&ver=8g/**, with a calculated variance from the period of **0.0134380314051.**

8. **Kuluoz:** Also uses the periodicity detector for all GET requests with URLs that match the regex: **/\/C338D6D09CA45230980EF28CDAEF57A1E80E725685E70E5ED4088FFB98E21ECC52E0A6FB44B8C30DEA90454BD8E292E523BE43AE9871A36910BACBD3E09B23700FDE12BC8A5F54E0FB8BDC91E6D5B4/**, with a calculated variance from the period of **1.346600235.**

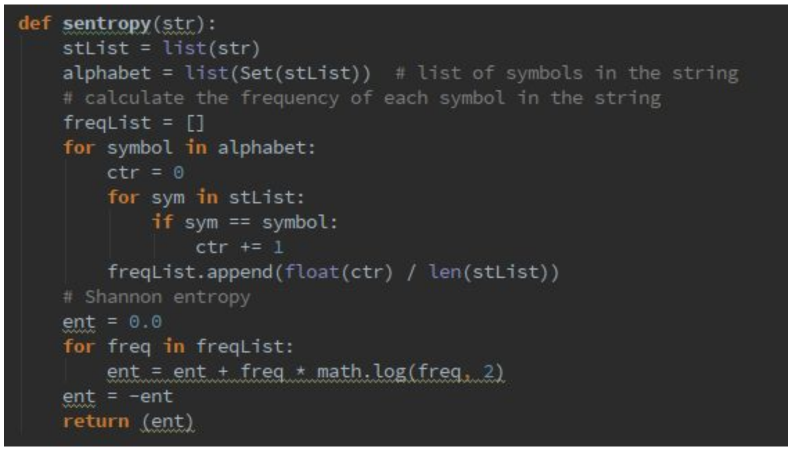
(4). Random Parameter Detection:

After some research, Shannon Entropy was picked out to be used in order to determine the randomness of a given parameter string in our behavioral model. According to (http://www.bearcave.com/misl/misl_tech/wavelets/compression/shannon.html), the Shannon entropy equation provides a way to estimate the average minimum number of bits needed to encode a string of symbols, based on the frequency of the symbols. In the below equation, "pi" represents the probability of any given character.

Shannon's entropy equation:

$$H(X) = -\sum_{i=0}^{N-1} p_i \log_2 p_i$$

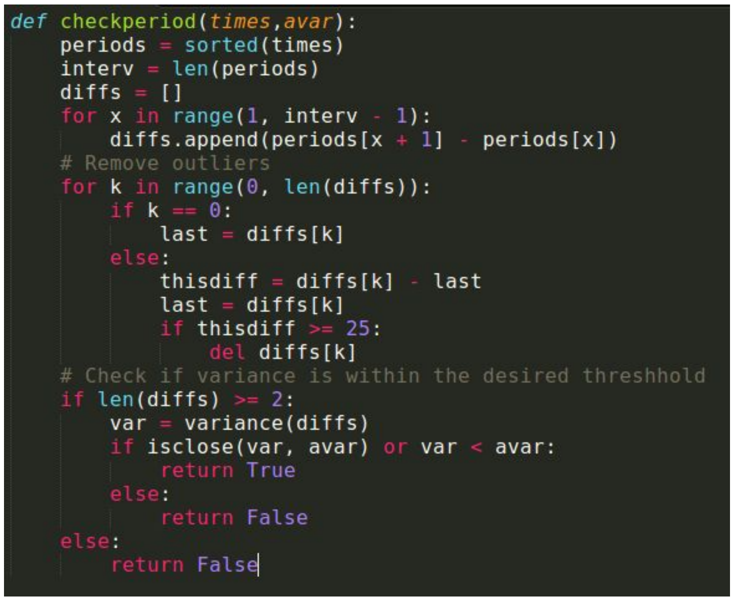
The following code snippet shows the implementation of this equation in our behavioral model, a function which receives a string as input and returns the Shannon entropy of the string. In the "System" folder in our github repo, you can see the "entropy_calc.py" script, which includes functions that were designed to find the range of entropies that any rule will allow based on the Bro logs extracted from that malware types' pcap traffic sample; one for the HTTP log and another from the DNS log.



```python
def sentropy(str):
    stList = list(str)
    alphabet = list(Set(stList))  # list of symbols in the string
    # calculate the frequency of each symbol in the string
    freqList = []
    for symbol in alphabet:
        ctr = 0
        for sym in stList:
            if sym == symbol:
                ctr += 1
        freqList.append(float(ctr) / len(stList))
    # Shannon entropy
    ent = 0.0
    for freq in freqList:
        ent = ent + freq * math.log(freq, 2)
    ent = -ent
    return (ent)
```

(5). Periodicity Detector:

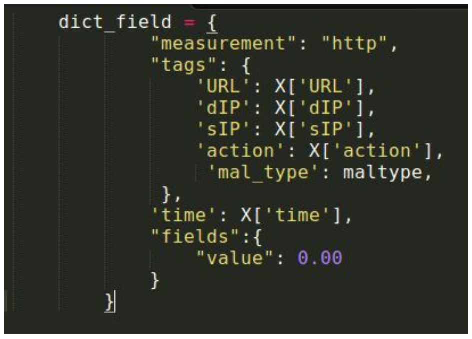
Below is our periodicity detector, which receives as arguments a list of times along with the threshold for variance from the period. The way this function is implemented is by calculated all of the time differences between every two timestamps in the list that is passed to it, and then calculating the variance of this new list of time differences. If the variance of these differences was less than or equal to the previously determined threshold, then all of the requests that these timestamps belong to are considered periodic. An addition was made that removed any requests that were huge outliers so that they would not also be considered malicious and so that they did not mess with the detection of periodicity for the rest of the traffic that could in fact be malicious.



```python
def checkperiod(times,avar):
    periods = sorted(times)
    interv = len(periods)
    diffs = []
    for x in range(1, interv - 1):
        diffs.append(periods[x + 1] - periods[x])
    # Remove outliers
    for k in range(0, len(diffs)):
        if k == 0:
            last = diffs[k]
        else:
            thisdiff = diffs[k] - last
            last = diffs[k]
            if thisdiff >= 25:
                del diffs[k]
    # Check if variance is within the desired threshhold
    if len(diffs) >= 2:
        var = variance(diffs)
        if isclose(var, avar) or var < avar:
            return True
        else:
            return False
    else:
        return False
```

(6). Results:

After all of the results of detection have been concluded, a function in the detection script called “CreateBatchMal” creates a dict_field for each request and appends it to the larger array of results. The formatting for this dictfield that contains the information about each malicious request can be seen below, where the “time” field is that of the original request, the “maltype” is designated in order to be used in any aggregator functions in the future for programs that build on this one, as well as the URL, destination IP, request type, and source IP of the original malicious request. This formatting can be seen below:



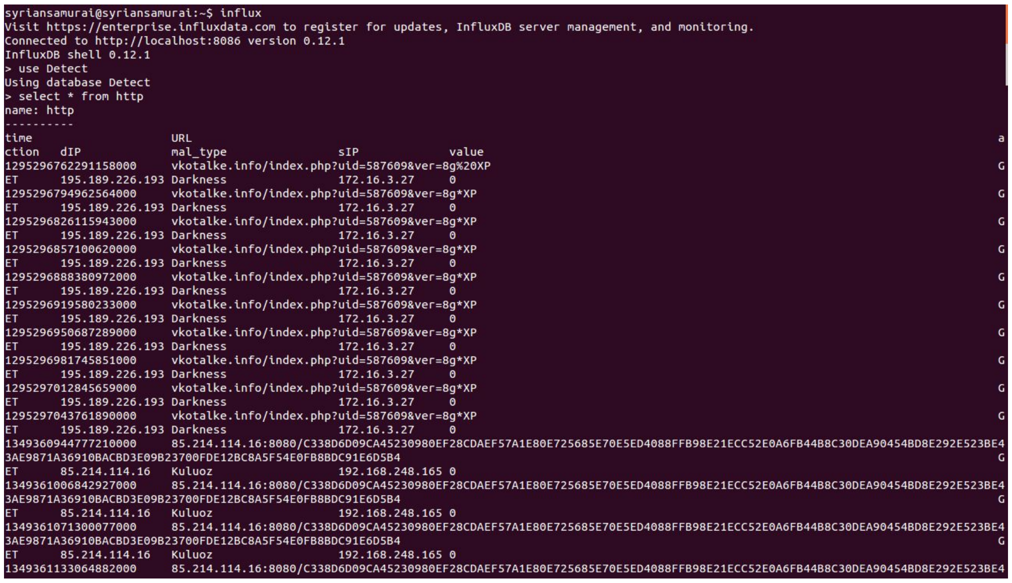
```
    dict_field = {
            "measurement": "http",
            "tags": {
                'URL': X['URL'],
                'dIP': X['dIP'],
                'sIP': X['sIP'],
                'action': X['action'],
                 'mal_type': maltype,
             },
            'time': X['time'],
            "fields":{
                "value": 0.00
            }
        }
```

The following image is a demo of InfluxDB running from the command line in order to show the results of the detection that were written to the database.

In order to do this, after the results of detection have been written, run the following commands:

- “influx”, in order to connect to InfluxDB
- “use DBNAME”, where DBNAME in this case is “Detect”
- “select * from http”, just a query that shows all of the series in the Detect DB.

As you can see, all of the tags from the dict_field are column names for the detection results written to InfluxDB.



```
syriansamurai@syriansamurai:~$ influx
Visit https://enterprise.influxdata.com to register for updates, InfluxDB server management, and monitoring.
Connected to http://localhost:8086 version 0.12.1
InfluxDB shell 0.12.1
> use Detect
Using database Detect
> select * from http
name: http
----------
time                    URL                                                                                                                    a
ction   dIP             mal_type                sIP             value
1295296762291158000     vkotalke.info/index.php?uid=587609&ver=8g%20XP                                                                         G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295296794962564000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295296826115943000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295296857100620000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295296888380972000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295296919580233000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295296950687289000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295296981745851000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295297012845659000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1295297043761890000     vkotalke.info/index.php?uid=587609&ver=8g*XP                                                                           G
ET      195.189.226.193 Darkness                172.16.3.27     0
1349360944777210000     85.214.114.16:8080/C338D6D09CA45230980EF28CDAEF57A1E80E725685E70E5ED4088FFB98E21ECC52E0A6FB44B8C30DEA90454BD8E292E523BE4
3AE9871A36910BACBD3E09B23700FDE12BC8A5F54E0FB8BDC91E6D5B4                                                                                      G
ET      85.214.114.16   Kuluoz                  192.168.248.165 0
1349361006842927000     85.214.114.16:8080/C338D6D09CA45230980EF28CDAEF57A1E80E725685E70E5ED4088FFB98E21ECC52E0A6FB44B8C30DEA90454BD8E292E523BE4
3AE9871A36910BACBD3E09B23700FDE12BC8A5F54E0FB8BDC91E6D5B4                                                                                      G
ET      85.214.114.16   Kuluoz                  192.168.248.165 0
1349361071300077000     85.214.114.16:8080/C338D6D09CA45230980EF28CDAEF57A1E80E725685E70E5ED4088FFB98E21ECC52E0A6FB44B8C30DEA90454BD8E292E523BE4
3AE9871A36910BACBD3E09B23700FDE12BC8A5F54E0FB8BDC91E6D5B4                                                                                      G
ET      85.214.114.16   Kuluoz                  192.168.248.165 0
1349361133064882000     85.214.114.16:8080/C338D6D09CA45230980EF28CDAEF57A1E80E725685E70E5ED4088FFB98E21ECC52E0A6FB44B8C30DEA90454BD8E292E523BE4
```

After the end of this detection script, it is now possible to pull the IP’s of all VM’s that have suspicious malicious behavior straight from the Detect database and then block their traffic.

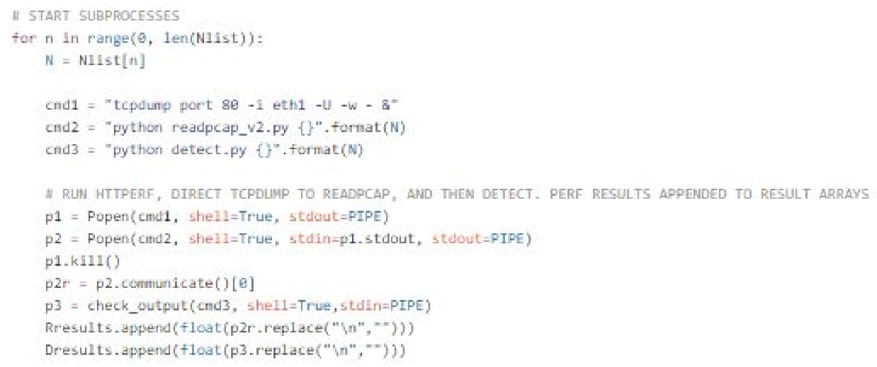
(7). Verdict:

At the end of our detection, and before writing the results to the Detection DB, we pull all of the IP's of the VM's that are responsible for malicious behaviour aside into a list. Then, we execute the following command on every IP in the list in order to drop all connections from that particular VM: **iptables -A FORWARD -s IP -j DROP ,** where IP is replaced by the actual IP of theVM. This concludes the detection portion of our project.

P6) Performance Testing

Stress Testing Automation Script:

In order to ease our performance testing, a separate special set of scripts was made that included edited versions of the Collection and Detection script. Refer to the folder "Performance-Testing" in our github repo. The perf_test.py script redirects the output of Tcpdump listening to Port 80 into the stdin of another process running readpcap_v2.py, which streams from Tcpdump and writes these requests to InfluxDB. Finally, another process runs the detection script, and the latency values for each of these runs for N requests is appended to a log file 'perf_log.txt'.
We recommend that you first run perf_test.py with "sudo python perf_test.py N", and then run Tsung to generate HTTP traffic that will be collected by Tcpdump.
Below is a part of the code for perf_test.py that shows the redirection of stdout to stdin of readpcap_v2.py:



```
# START SUBPROCESSES
for n in range(0, len(Nlist)):
    N = Nlist[n]

    cmd1 = "tcpdump port 80 -i eth1 -U -w - &"
    cmd2 = "python readpcap_v2.py {}".format(N)
    cmd3 = "python detect.py {}".format(N)

    # RUN HTTPERF, DIRECT TCPDUMP TO READPCAP, AND THEN DETECT. PERF RESULTS APPENDED TO RESULT ARRAYS
    p1 = Popen(cmd1, shell=True, stdout=PIPE)
    p2 = Popen(cmd2, shell=True, stdin=p1.stdout, stdout=PIPE)
    p1.kill()
    p2r = p2.communicate()[0]
    p3 = check_output(cmd3, shell=True,stdin=PIPE)
    Rresults.append(float(p2r.replace("\n","")))
    Dresults.append(float(p3.replace("\n","")))
```

What is Tsung?

The purpose of Tsung (http://tsung.erlang-projects.org/) is to simulate users in order to test the scalability and performance of IP based client/server applications. You can use it to do load and stress testing of your servers. Many protocols have been implemented and tested, and it can be easily extended. It virtualizes several clients in order to generate the high volume of requests.

Why do we need Tsung?

To do performance testing we initially started out with httperf. It was capable of generating around 200k packets/sec. It sounded impressive but when we actually started testing it, it struggled to generate 10k requests/sec. Remember, requests are different from packets as requests are composed of several packets. This is why we needed Tsung to mediate the matter so we could actually test our code. Tsung on first impression was able to generate 600k packets/sec, and with a few adjustments could easily achieve 1m packets/sec.

Tsung Installation and Execution Instructions:

Simple, just execute the commands below given you have ubuntu 14.04:

```
sudo apt-get install erlang #download erlang
wget http://tsung.erlang-projects.org/dist/tsung-1.4.2.tar.gz
tar zxfv  tsung-1.4.2.tar.gz
cd tsung-1.4.2
sudo apt-get install make
sudo ./configure
sudo make
sudo make install
```

Then you need to create an xml file to configure it. Refer to http://dak1n1.com/blog/14-http-load-generate/ for instructions on it's creation and configuration.

Our Configuration File:

```
<?xml version="1.0"?>
<!DOCTYPE tsung SYSTEM "/usr/share/tsung/tsung-1.0.dtd">
<tsung loglevel="notice" version="1.0">

<clients>
<client host="vrouter" weight="1" cpu="10" maxusers="40000">
<ip value="192.168.100.26"/>
</client>
</clients>

<servers>
<server host="www.google.com" port="80" type="tcp"/>
</servers>
```

```
<load>
<arrivalphase phase="1" duration="1" unit="minute">
<users maxnumber="15000" arrivalrate="8" unit="second"/>
</arrivalphase>

<arrivalphase phase="2" duration="1" unit="minute">
<users maxnumber="15000" arrivalrate="8" unit="second"/>
</arrivalphase>

<arrivalphase phase="3" duration="1" unit="minute">
<users maxnumber="20000" arrivalrate="3" unit="second"/>
</arrivalphase>

</load>

<sessions>
<session probability="100" name="ab" type="ts_http">
<for from="1" to="10000000" var="i">
<request> <http url="/test.txt" method="GET" version="1.1"/> </request>
</for>
</session>
</sessions>
</tsung>
```

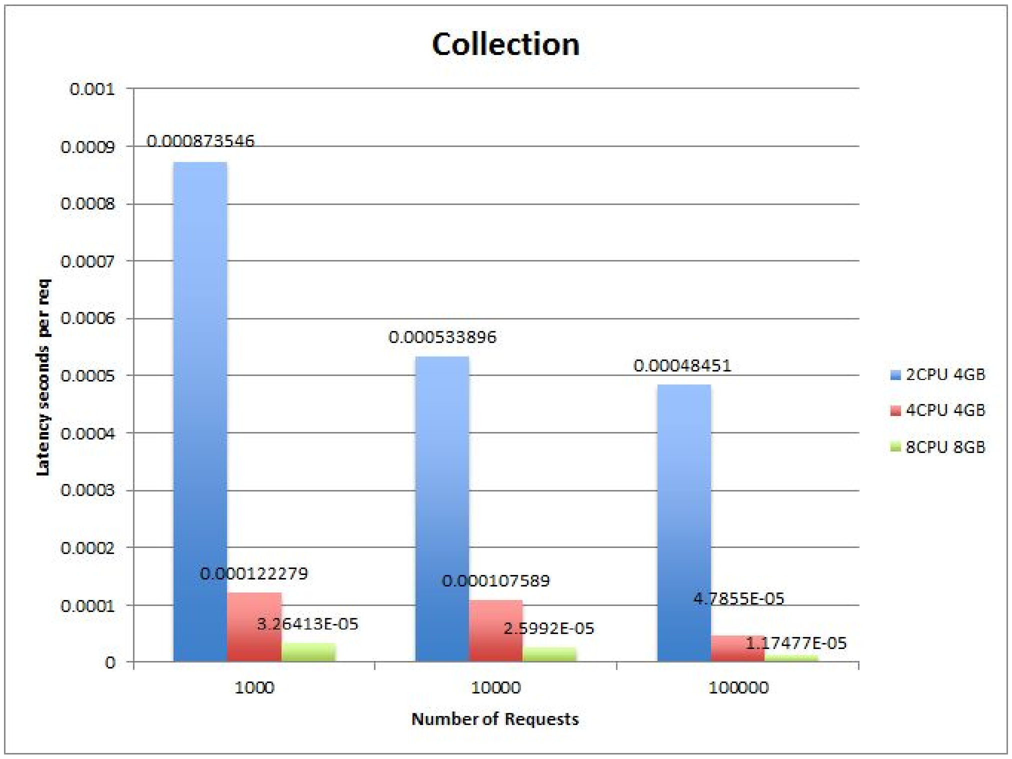
Finally, you may run Tsung by executing the following command:

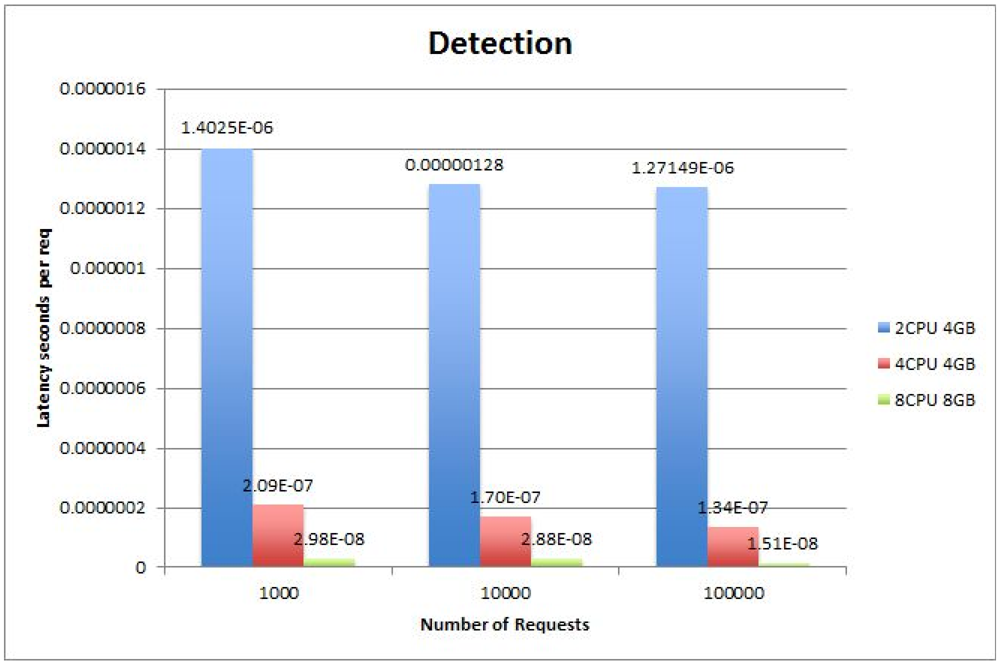
```
tsung -f filename.xml start
```

(Where “filename.xml” is the name of your config file)

Stress Testing Under Different vRouter Configurations:

In our performance testing, we set up the vRouter with 3 different configurations (2CPU/4GB RAM, 4CPU/4GB RAM, 8CPU/8GB RAM), ran the perf _test.py script for 3 values of N requests seperately (1k, 10k, and 100k), and then ran Tsung in order to generate HTTP traffic.





As you can see, time latency per request in seconds decreased slightly as the size of the batch of requests we were processing increased, and decreased dramatically with increases to #CPU’s and RAM memory available. Further improvements to performance can be made with additional resources allocated to the vRouter, but this was our limitation within the MOC for the purpose of this project.