Firewall as a Service

I. Introduction

Virtual Firewall provides similar protection as a traditional firewall, but the service is hosted in the cloud, meaning its available at all locations and on practically any device. Instead of existing as an appliance, the firewall exists as a virtual barrier.

In this project we are providing Firewall-as-a-Service which essentially means that we are not just providing a product (virtual firewall) to the customer, but we are also providing services such as Alert Logging, Connection Tracking, Scalability, python API, along with our virtual Firewall.

II. System Architecture

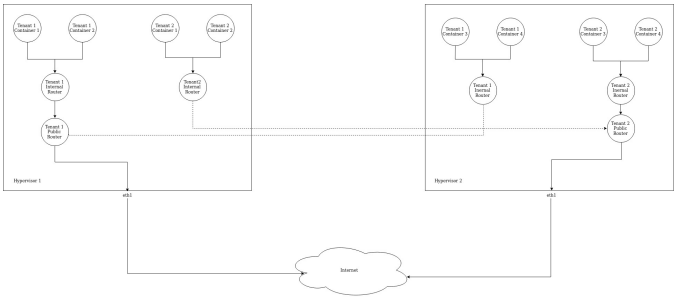

Fig. 1: System Architecture

In the Figure 1 we can see the system architecture of our project until Milestone 3. The key components of our system are as follows:

1) Internal Gateway: We created a container which is connected to each tenants container using veth pairs to provide the service to each tenant.
2) Public Gateway: As we can see in the above figure that each Tenant is assigned a container as a public gateway which will be used to reach the internet from each Tenant's container. Each public gateway is connected with the tenants respective internal gateways using a veth pair.
3) Tenant's Containers: Each tenant can create multiple containers using the automation tool and accordingly his/her topology will be created. He can specify the types of connections that he wants such as P2P, L3 tunnel, L2 tunnel, etc.
4) Firewalls: In our system we provide firewalls at three layers namely, tenants container and tenants internal gateway, and tenant's public gateway.
   (i) Firewall at internal gateway: It is an apt place for deploying a firewall for each tenant so that all the security subnet policies of a tenant which should be enforced on each container of that network of the tenant can be placed in his/her respective internal gateway.
   (ii) Firewall at public gateway: It is an apt place for deploying a firewall for each tenant so that all the security public network policies of a tenant which should be enforced on each container of that network of the tenant can be placed in his/her respective public gateway.
   (iii) Firewall at containers: If required by the tenant then we can also add custom firewall rules to a particular container.

Thus, the tenants container, internal gateway and public gateway are three layers in the system where the firewall will be deployed.

III. Service components and features of Firewall

A. Monitoring traffic

In this feature the customer gets the option of monitoring the incoming/outgoing traffic.This is based on the rules deployed on the firewall. Packets will be scanned at the network and also at the transport layer for suspicious activity. The customer can see the IP addresses(source and destination)- Layer 3 and Ports(source and destination). The customer can use this feature to filter traffic and rearrange his firewall rules to improve efficiency. The customer is provided with the data of number of packets and number of bytes hitting a particular rule in iptables rules. Now with this available information customer can reorder the firewall rules to improve CPU utilization.

we used this command for retrieving the number of packets and number of bytes hitting a particular rule by:

```
sudo iptables -L -v -n
```

In figure:7 below you can see the desired information about number of packets and number of bytes hitting a particular rule.

In figure 8 we can see the CPU utilization of a VM, which can be efficiently managed with re-ordering of firewall rules.

B. Alert logging

Enabling this feature would help in logging the packets which can be used for debugging and troubleshooting. In the basic firewall we have provided some rules to prevent malicious traffic. These malicious packets are logged first and later dropped. Logging feature helps a customer in better understanding the incoming traffic from malicious sources. These basic firewall rules prevents the customer from DOS attacks. We also provide the customization option for the customer in choosing necessary traffic to be logged. We use the INPUT chain to achieve this. Here are few commands

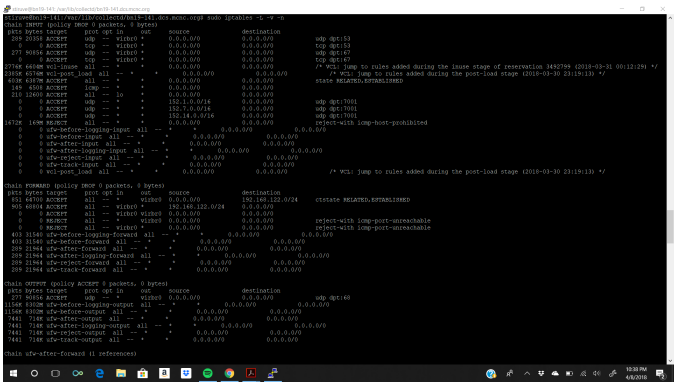

Fig. 2: Firewall rule Statistics:

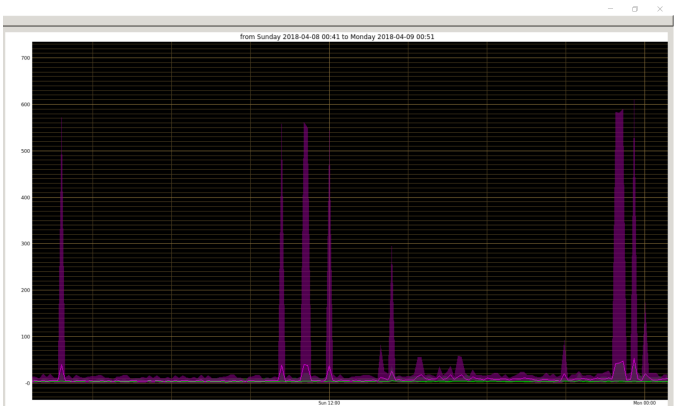

Fig. 3: CPU-Utilization

which we have used in our service by creating a new chain named:

```
LOG_AND_DROP
```

```
sudo iptables -N LOG_AND_DROP
sudo iptables -A LOG_AND_DROP -j LOG
--log-prefix "iptables dropped"
sudo iptables -A LOG_AND_DROP -j DROP

For example : sudo -A INPUT -m conntrack
--ctstate INVALID -j DROP
LOG_AND_DROP
```

In the above example we have used the INPUT chain to log any packets which have an INVALID connection state and later drop them. In figure:9 we can see the logs from /var/log/kern.log file indicating IPtables have dropped these packets.

C. Packet filtering

In this feature we provide some basic firewall rules which the customer can use them to prevent from various types of attacks like: DOS, RST scan, FIN Scan. Suspicious packets will be dropped based on the firewall rules provided. The customer is also provided with the service of adding customized

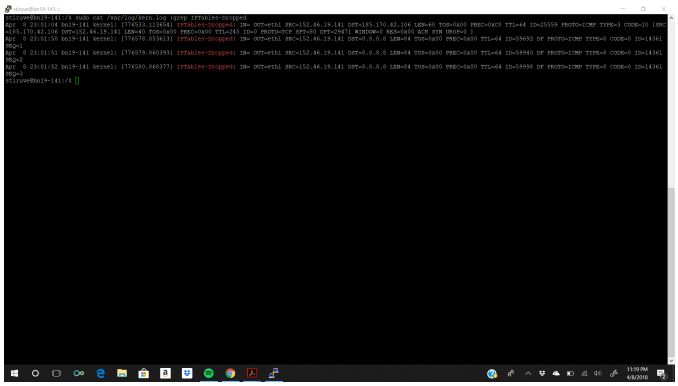

Fig. 4: LOG describing of dropped packets by IPtables

firewall rule.We will not perform deep packet inspection on the packets.

D. Connection tracking

In this feature the customer is able to track connections, this is achieved with the help of stateful firewalls. In this service, the client is able to view all his connections and can also terminate his connections with a source. we have used the conntrack to achieve this. The following commands were used:

```
.
conntrack -L

This command is used for displaying the
connections established.

conntrack -D -p tcp --dport 3389
```

Here, we are deleting the connection based on the user's input of destination port and the type of protocol. The customer has the flexibility to terminate any connection by specifying the protocol and source/destination port. In Figure:10 we can see available connections and the customer terminates one connection of tcp protocol with destination port 80.

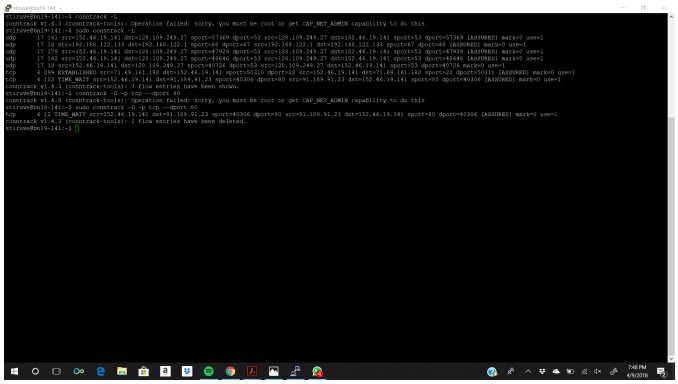

Fig. 5: Connection Tracking

E. Internet-Speed

In this service, the customer can test his internet speed the python script will provide ping, download and upload speeds. We have used the speedtest-cli tool in the python script to achieve this. In figure:11 we can see this data captured during every hour of the day.

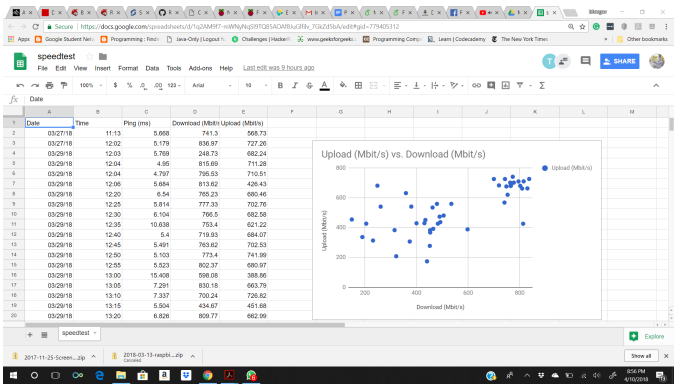

Fig. 6: Internet speed test

F. Isolation

As we can observe in the Figure that each tenant is allocated pure L2 networks which enables L2 isolation as well as L3 isolation among the tenants.

- L2 Isolation: The pure L2 networks enables multiple tenants to use the same MAC addresses for their VMs without any break in the communication within the network or with the outside world/Internet. This also enables a Tenant to use the same MAC for multiple VMs which are not part of the same network.
- L3 Isolation: Similar to the L2 isolation, in L3 isolation we can see that the tenants can use the same IP addresses for their network even if it is been used by another tenant. This is possible because each tenant is assigned a namespace through which all the tenants VMs packets will be transmitted after performing source NAT. And even though the tenants networks are part of the hypervisor but they are still pure L2 networks which dont have L3 interfaces associated unless manually configured.

Thus, assuming that enough resources are present, this feature of the system allows us as a service provider to connect as many VMs as possible with duplicate MAC and IP addresses which helps in scaling up the network. As mentioned in the Section **??** we are also planning to include VxLAN and GRE tunnels to provide scalability across multiple hypervisors.

G. Python API

We have provide a set of python scripts with functionalities such as createFirewall, deleteFirewall, listFirewall, createFirewallRule and deleteFirewallRule. The scripts could be used in provisioning firewall services and features. The scripts and their functionalities are more explored in the python API section.

IV. Python API

This section explains the scripts and their functionalities.

A. createFirewall

This script can be used to deploy some basic firewall rules on a VM. We have listed 30 'iptables' rules in the script, when the user runs this script these iptable rules would be added to the iptable of the VM. These rules can prevent DOS attacks on the VM. This is an added functionality for the user, it's up to their discretion to add these rules or not. The figure shows the output of the createFirewall script.

```
bsinha@bn17-163:~$ sudo python createFirewall.py
1
192.168.122.168
/usr/lib/python2.7/dist-packages/Crypto/Cipher/blockalgo.py:141: FutureWarning:
CTR mode needs counter parameter, not IV
  self._cipher = factory.new(key, *args, **kwargs)
/usr/lib/python2.7/dist-packages/paramiko/client.py:645: UserWarning: Unknown ss
h-rsa host key for 192.168.122.168: dee41b62aba08530e9569a17c7feb316
  (key.get_name(), hostname, hexlify(key.get_fingerprint()))
Default IP table rules have been added to the VM
```

Fig. 7: CreateFirewall output

B. deleteFirewall

This script can be used to dump all the iptable rules from a specific VM. This can be used by the user, if they want to dump all the rules from iptable of a VM.

C. listFirewall

This script lists all the rules in the iptable with line number, of a VM in that instance. This can be used with scripts like deleteFirewallRule to see all the rules and then delete a specific rule by line number.

The figure shows the output of the script.

```
bsinha@bn17-163:~$ sudo python createFirewall.py
1
192.168.122.168
/usr/lib/python2.7/dist-packages/Crypto/Cipher/blockalgo.py:141: FutureWarning:
CTR mode needs counter parameter, not IV
  self._cipher = factory.new(key, *args, **kwargs)
/usr/lib/python2.7/dist-packages/paramiko/client.py:645: UserWarning: Unknown ss
h-rsa host key for 192.168.122.168: dee41b62aba08530e9569a17c7feb316
  (key.get_name(), hostname, hexlify(key.get_fingerprint()))
Default IP table rules have been added to the VM
```

Fig. 8: listFirewall output

D. addFirewallRule

This script can be used to add a new rule to the iptable by the user. The user doesn't need to write the iptable command to be added, they just need to put in parameters like destination port, protocol, etc and the script dynamically creates the command for them and adds the command to the iptable of the VM.

E. deleteFirewallRule

This script can be used to delete a specific iptable rule for a VM. This script can be used in tandem with the listFirewall script, as the user can fist list out the rules for a VM and then give the line number of the rule to this script and the script will delete that rule form the table.

F. blockIP

This script can be used to block a specific IP from the VM. This would be useful in a scenario where the user notices some unusual traffic from an IP address, they could then execute this script and then block the IP address. The following images show this script in action.

Fig. 9: BlockIP output

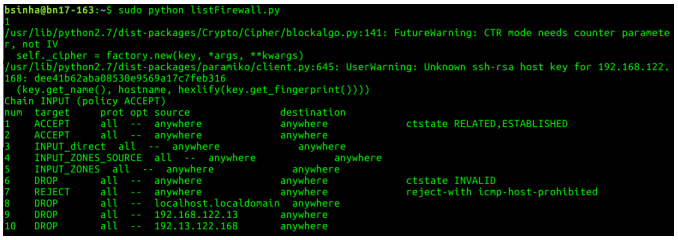

Fig. 10: List after block output

As it is shown, a specific IP is blocked and that ruke is populated in the IPtable by the script.

G. unblockIP

This script can be used to unblock a specific IP from the VM. This would be useful in a scenario where the user wants to unblock an IP address which was blocked previously. In the following example an IP address is provided to the script by the user and the script then removes the block command from the iptable of the VM.

Fig. 11: UnBlockIP output

Fig. 12: List after block output

H. blockMAC

This script can be used to block a specific MAC address from the VM. This would be useful in a scenario where the user notices some unusual traffic from a MAC address, they could then execute this script and then block the MAC address. The following images show this script in action.

Fig. 13: BloackMAC output

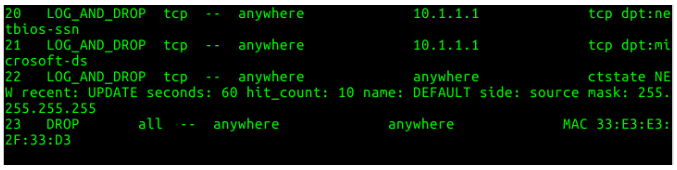

Fig. 14: List after block output

I. connectionTracking

This script shows the list of the connections being tracked on a VM. The output of the scriptis shown in the figure below.

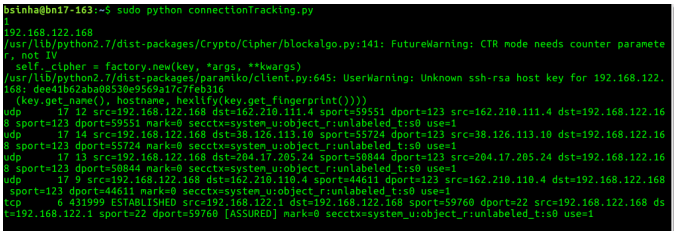

Fig. 15: Connection tracking

J. connectionTrackingDeleteConnection

This script can be used to delete a connection being tracked. This script can be used by the connectionTracking script to delete a specific connection being tracked after vieing the list of connections being tracked.

K. listRuleTraffic

This script can be used to list the traffic(in packets and bytes) that each rule is getting. This can be used to increase performance, for example, in a case where the user executes this script and notices that a specific rule in the bottom stack of Iptable is getting a lot of traffic, then the user can move that rule to the top of IPtable by useing the deleteRule and addRule scripts. The screen shot below shows the output of this script.

L. blockWebsite

This script can be used to block outgoing traffic to a specific URL. For example, if the URL facebook.com is given as an input to this script, it will block all outgoing traffic to facebook.com. This functionality can be used by the customer in case they want to block outgoing traffic to particular URLs.

In the following example, we have blocked instagram.com from the VM. We first tried pinging instagram.com and then executed the script and tried pinging it again. The VM was not able top ping after the block.

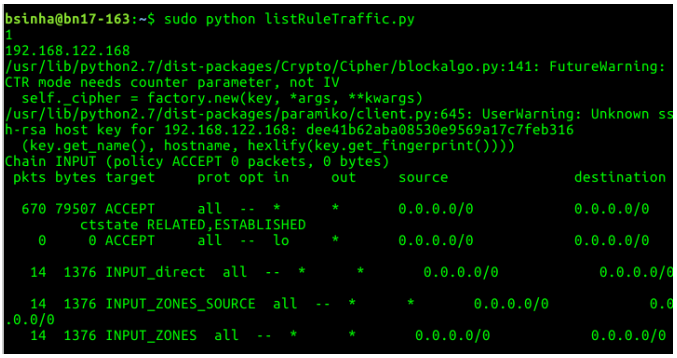

Fig. 16: Traffic on Rules

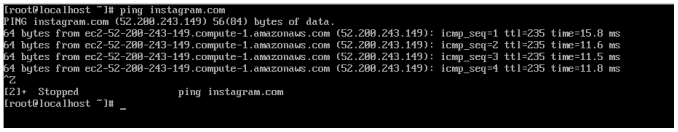

Fig. 17: Ping before block

V. Features to work on in Future

A. Scalability

We are planning to include VxLAN and GRE tunnels to provide scalability across multiple hypervisors.

B. High Availability

Availability will be ensured with two firewalls such that if the primary goes down then the secondary gets notified and then takes over. we are facing complexity in managing the states of the firewalls, which we are working on.

VI. Future Automation Plan

A. Creation of network for tenant

In this milestone we have manually created a network topology for multiple tenants whilst insuring isolation between them. The commands used for this network topology could be made into a bash script, but instead we intend to use containers in the future. So in the upcoming milestone we will create a bash script which will use containers and create an almost similar topology for a new tenant when they enter the cloud.

B. Scalability

We intend to use tunnels to give a single user VMs on multiple hypervisors. We are going to fix a number (For ex. 5), which would be the total VMs running on a hypervisor, since we have limited resources on a hypervisor. So whenever any user requests to make a new VM and the total VMs running are equal to the max VMs, we will spawn a new VM in another hypervisor for that tenant and provision the firewall on the new VM.

Fig. 18: Blocking instagram

Fig. 19: Pinging instagram after block

C. Python Package

In this milestone, we have created python scripts for each functionality. For the next milestone we intend to transform these scripts into one Python package and we will include more functions (like createVM). This package could be easily imported in a python script and the functions could be used directly. This package is going to be a wrapper for all the operations being performed and would enable a developer to create VMs in the cloud and provision the firewall directly while maintaining isolation between tenants.