#1 Simulating TCP Security Attacks - (Feb 27th)

In cybersecurity, most discussions—from textbooks to podcasts—are highly theoretical. Networking protocols and security mechanisms are often taught with diagrams but rarely applied hands-on. Seeing these attacks in action is rare, making hands-on experience crucial.

In this article, I will explore and simulate some common TCP-related attacks. Before diving into the attacks, I'll provide a brief background on the TCP protocol. Here are the attacks I will be simulating/looking into:

- TCP SYN flood attack - SYN cookies
- TCP reset attacks
- TCP session hijacking attacks
- Reverse shell attack

I will also automate these attacks in Python to demonstrate how efficiently they can be executed from an attacker's perspective.

What is TCP?

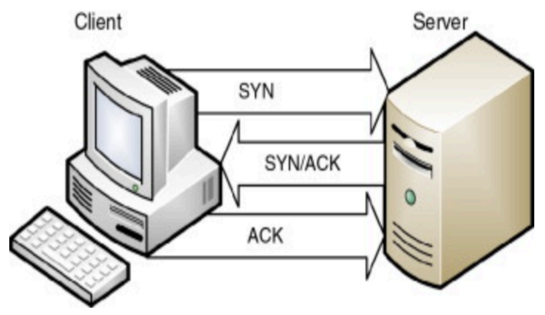
Imagine you send an Instagram post about a new Joe's Pizza location to your friends. This post is broken into small chunks (packets) and sent across the internet. The TCP protocol ensures these chunks arrive in the right order, error-free, and without missing data. Without TCP, your message might be incomplete or scrambled!

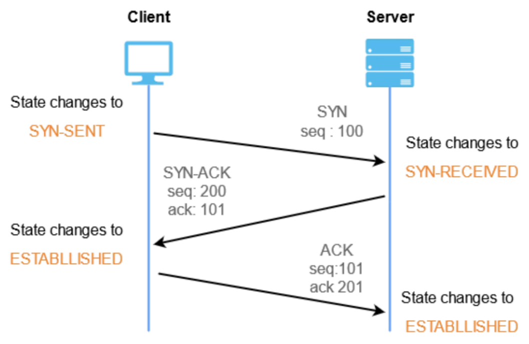
TCP ensures reliable communication through a process called the three-way handshake, where the sender and receiver establish a connection before data is transmitted. This mechanism helps prevent data loss and ensures packets arrive in order.

For those who want to dive deeper into TCP, here are some technical resources::

- https://www.fortinet.com/resources/cyberglossary/tcp-ip#:~:text=What%20does%20TCP%20mean%3F,data%20in%20digital%20network%20communications.
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- https://www.sciencedirect.com/topics/computer-science/three-way-handshake#:~:text=The%20TCP%20handshake,as%20shown%20in%20Figure%203.8.

Below is a diagram illustrating the TCP handshake process:





What are TCP Attacks?

Now that we understand how TCP ensures reliable communication, let's explore the dark side. Hackers exploit weaknesses in TCP to disrupt services, intercept sensitive data, and take over sessions. Understanding these attacks is crucial for building more secure networks.

Understanding TCP attacks isn't just about reading diagrams – it's about seeing how they play out in real time. In this article, I'm going to demonstrate how these attacks are executed and show how easily systems can be compromised if they're not properly secured.

To see these attacks in action, we need a controlled environment that allows us to simulate them safely. For this, we'll set up a **SEED Labs environment**, which provides a hands-on approach to cybersecurity experimentation.

Lab setup:

This project uses SEED Labs, a cybersecurity education framework that provides virtualized environments to explore topics like network security, web security, and operating systems.

We will set up a **Virtual Machine (VM)** using **VirtualBox** (for Windows) or **VMware Fusion** (for Apple Silicon Macs). The VM runs an Ubuntu ISO image pre-configured with SEED Labs, which includes a **docker-compose.yml** file. This file builds and runs **four containers** representing:

- Attacker
- User 1
- User 2
- Victim

This setup simulates multiple machines on the same **local area network (LAN)**, allowing us to observe and analyze attacks from different perspectives.

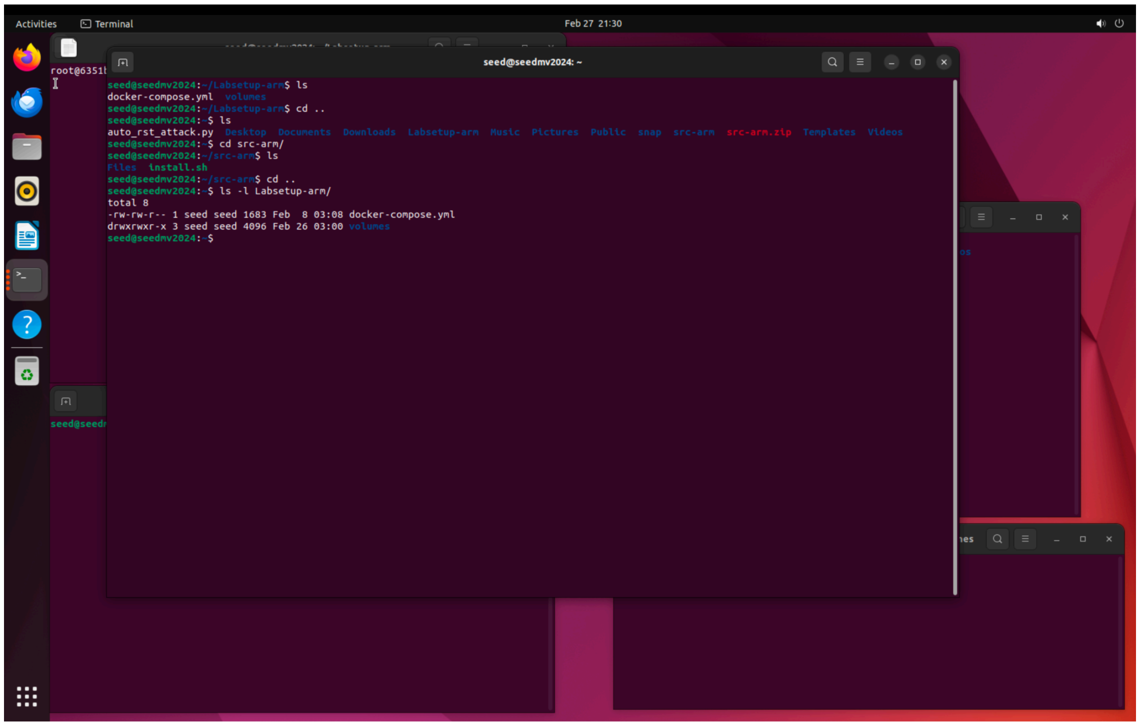
For a step-by-step guide on setting up the Ubuntu VM and SEED Lab environment, refer to the project's GitHub documentation: **[Insert GitHub link]**.

Let's set up the lab and look into the first attack: SYN Flood

Once the SEED Lab environment is installed and running, you should see a **Labsetup** folder in your local repository. If you are on **Apple Silicon**, extracting the **src-arm** installer creates a **Labsetup-arm** folder. For **Intel and Windows users**, download and extract **Labsetup.zip** from the SEED Labs website:

- https://seedsecuritylabs.org/Labs_20.04/Networking/TCP_Attacks/

At this point, your terminal should display the **Labsetup** folder structure, which includes the necessary files for running the simulations.



```
seed@seedmv2024:~/Labsetup-arm$ ls
docker-compose.yml  volumes
seed@seedmv2024:~/Labsetup-arm$ cd ..
seed@seedmv2024:~$ ls
auto_rst_attack.py  Desktop  Documents  Downloads  Labsetup-arm  Music  Pictures  Public  snap  src-arm  src-arm.zip  Templates  Videos
seed@seedmv2024:~$ cd src-arm/
seed@seedmv2024:~/src-arm$ ls
Files  install.sh
seed@seedmv2024:~/src-arm$ cd ..
seed@seedmv2024:~$ ls -l Labsetup-arm/
total 8
-rw-rw-r-- 1 seed seed 1683 Feb  8 03:08 docker-compose.yml
drwxrwxr-x 3 seed seed 4096 Feb 26 03:00 volumes
seed@seedmv2024:~$
```

Understanding the SEED Lab Environment

The **docker-compose.yml** file configures the containers, defines their properties, and establishes their network connections. If you want to learn more about its structure and components, check out:

- https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/compose-onelan.md

To start and manage Docker containers, refer to this command reference:

- https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/docker-commands.md

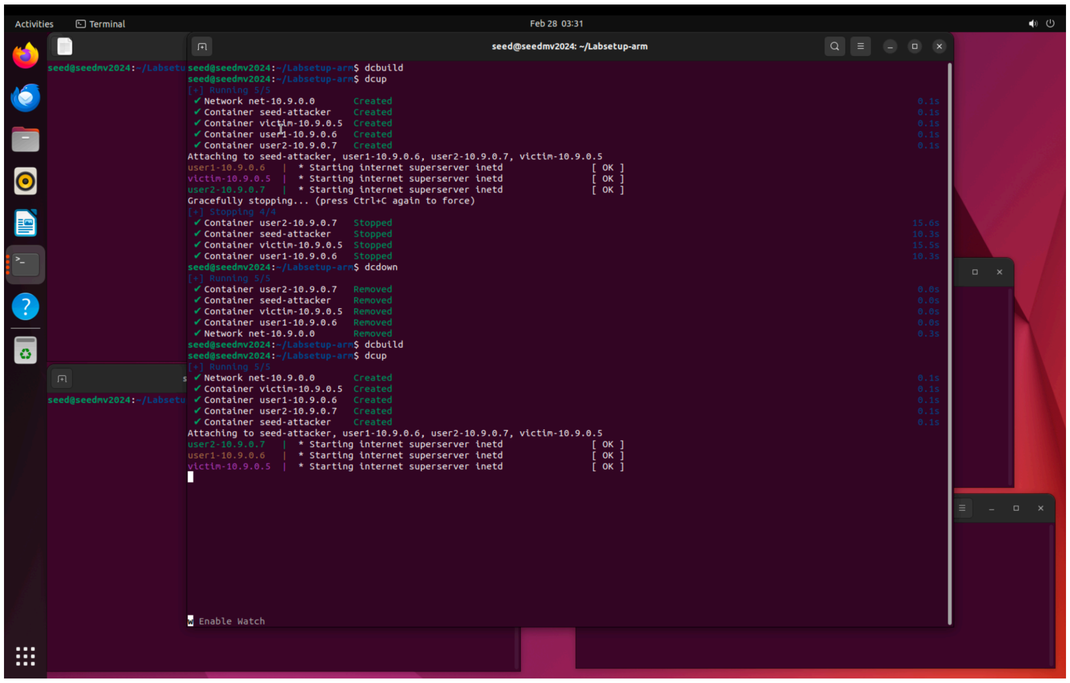
Shared Folder (Volumes):
The **attacker container** has a **shared folder** called `volumes` that allows code to be edited inside the VM while keeping attack scripts accessible. This makes it easier to modify and execute attack code between the VM and the containerized environment.

Starting up the Containers:

Now the way to start up the containers is to use the **dock-compose build** or **dcbuild** command in the host VM terminal. This builds the container images defined in the **docker-compose.yml** file. In order to start the containers to run, the command is **docker-compose up** or **dcup** for the aliases

- Note: aliases are defined from the **.bashrc** file

From there you will notice all the containers running in the background. If you want to close down the containers **control/command + c to** stop the containers, interrupting the containers for running, and use **docker-compose down or dcdown** to shut down the containers which will be removed, and there will be a need to **dcbuild** again to build the containers if used again. This is how the process looks like from the host VM.

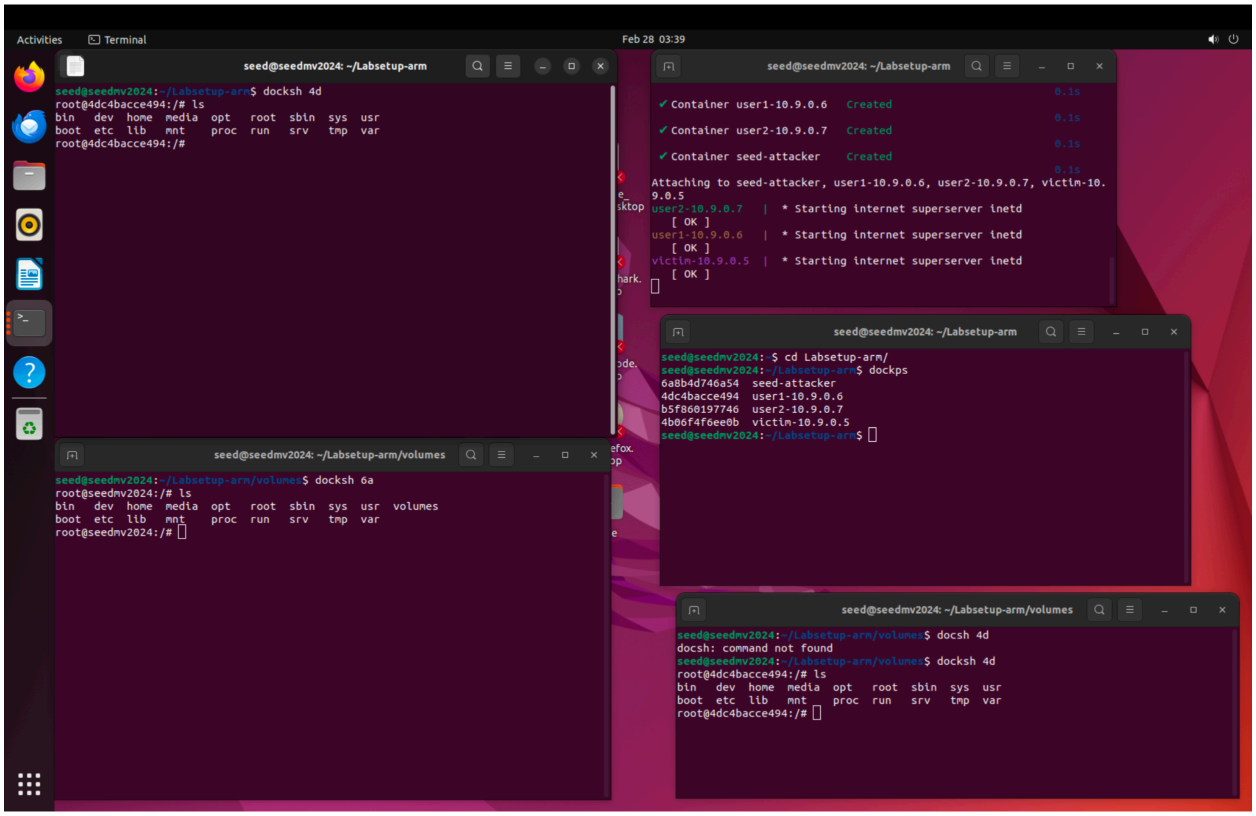


```
seed@seedmv2024:~/Labsetup-arm$ dcbuild
seed@seedmv2024:~/Labsetup-arm$ dcup
[+] Running 5/5
 ✓ Network net-10.9.0.0        Created                                    0.1s
 ✓ Container seed-attacker     Created                                    0.1s
 ✓ Container victim-10.9.0.5   Created                                    0.1s
 ✓ Container user1-10.9.0.6    Created                                    0.1s
 ✓ Container user2-10.9.0.7    Created                                    0.1s
Attaching to seed-attacker, user1-10.9.0.6, user2-10.9.0.7, victim-10.9.0.5
user1-10.9.0.6   |  * Starting internet superserver inetd              [ OK ]
victim-10.9.0.5  |  * Starting internet superserver inetd              [ OK ]
user2-10.9.0.7   |  * Starting internet superserver inetd              [ OK ]
Gracefully stopping... (press Ctrl+C again to force)
[+] Stopping 4/4
 ✓ Container user2-10.9.0.7    Stopped                                   15.6s
 ✓ Container seed-attacker     Stopped                                   10.3s
 ✓ Container victim-10.9.0.5   Stopped                                   15.5s
 ✓ Container user1-10.9.0.6    Stopped                                   10.3s
seed@seedmv2024:~/Labsetup-arm$ dcdown
[+] Running 5/5
 ✓ Container user2-10.9.0.7    Removed                                    0.0s
 ✓ Container seed-attacker     Removed                                    0.0s
 ✓ Container victim-10.9.0.5   Removed                                    0.0s
 ✓ Container user1-10.9.0.6    Removed                                    0.0s
 ✓ Network net-10.9.0.0        Removed                                    0.3s
seed@seedmv2024:~/Labsetup-arm$ dcbuild
seed@seedmv2024:~/Labsetup-arm$ dcup
[+] Running 5/5
 ✓ Network net-10.9.0.0        Created                                    0.1s
 ✓ Container victim-10.9.0.5   Created                                    0.1s
 ✓ Container user1-10.9.0.6    Created                                    0.1s
 ✓ Container user2-10.9.0.7    Created                                    0.1s
 ✓ Container seed-attacker     Created                                    0.1s
Attaching to seed-attacker, user1-10.9.0.6, user2-10.9.0.7, victim-10.9.0.5
user2-10.9.0.7   |  * Starting internet superserver inetd              [ OK ]
user1-10.9.0.6   |  * Starting internet superserver inetd              [ OK ]
victim-10.9.0.5  |  * Starting internet superserver inetd              [ OK ]

w Enable Watch
```

In order to log into the containers, I would recommend having 5 different tabs of the terminal open with 3 of them being the containers you will be signing into (attacker, user1/user2, and victim), 1 for commands from the host VM, and the last one that keeps the containers running.

To make the login process quicker, recommend using the not container running Host VM terminal window to do a **docker ps** command (this is the **dockps**) which finds the IDs of the container and to start the shell of the container is using **docksh <id>** (note the id doesn't have to be the full id, it need to be enough to distinguish between to connect to the container). So here is how I like to have it look virtually for me, but any way is fine as long as it makes sense to you when working through the attacks.


Activities
Terminal
Feb 28 03:39
seed@seedmv2024: ~/Labsetup-arm
seed@seedmv2024: ~/Labsetup-arm/volumes

```
seed@seedmv2024:~/Labsetup-arm$ docksh 4d
root@4dc4bacce494:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@4dc4bacce494:/#
```

```
                                                        0.1s
 ✓ Container user1-10.9.0.6   Created
                                                        0.1s
 ✓ Container user2-10.9.0.7   Created
                                                        0.1s
 ✓ Container seed-attacker    Created
                                                        0.1s
Attaching to seed-attacker, user1-10.9.0.6, user2-10.9.0.7, victim-10.
9.0.5
user2-10.9.0.7   |  * Starting internet superserver inetd
   [ OK ]
user1-10.9.0.6   |  * Starting internet superserver inetd
   [ OK ]
victim-10.9.0.5  |  * Starting internet superserver inetd
   [ OK ]
```

```
seed@seedmv2024:~$ cd Labsetup-arm/
seed@seedmv2024:~/Labsetup-arm$ dockps
6a8b4d746a54  seed-attacker
4dc4bacce494  user1-10.9.0.6
b5f860197746  user2-10.9.0.7
4b06f4f6ee0b  victim-10.9.0.5
seed@seedmv2024:~/Labsetup-arm$
```

```
seed@seedmv2024:~/Labsetup-arm/volumes$ docksh 6a
root@seedmv2024:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr  volumes
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@seedmv2024:/#
```

```
seed@seedmv2024:~/Labsetup-arm/volumes$ docsh 4d
docsh: command not found
seed@seedmv2024:~/Labsetup-arm/volumes$ docksh 4d
root@4dc4bacce494:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@4dc4bacce494:/#
```

- Note that in the attacker shell, we see the `volumes` shared folder both the Host VM and attacker containers share.

Now we have the containers set up and machines started, we can now start the attacks process by simulating the first TCP attack, TCP SYN Flood attack.

TCP SYN Flood Attack:

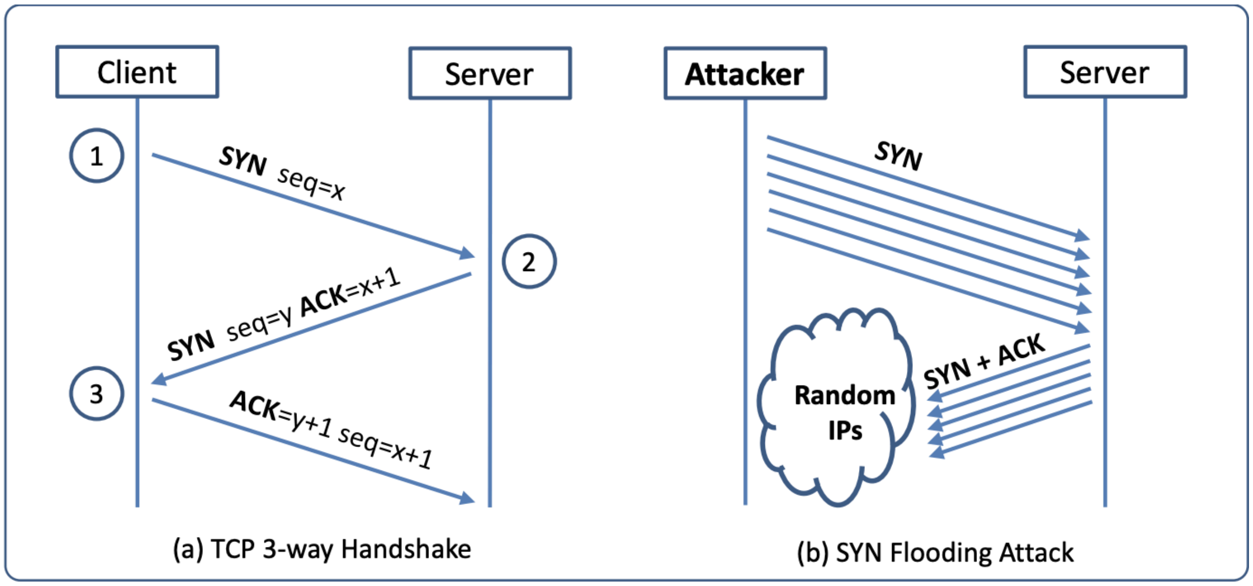
A TCP SYN Flood attack is a type of Denial-of-Service (DoS) attack that exploits the TCP three-way handshake to overwhelm a target server, making it unresponsive to legitimate users.

Here is how the attack works:

1. The attacker sends a large number of **SYN (synchronize) packets** to the victim's server, often using spoofed IP addresses.

2. The server, assuming these are legitimate connection requests, responds with SYN-ACK (synchronize-acknowledge) packets and reserves system resources to complete the handshake.
3. Instead of replying with the final ACK packet, the attacker never completes the handshake and continues to flood the server with more SYN requests.
4. Since the server keeps waiting for the missing ACK responses, it leaves connections in a "half-open" state and consumes memory, CPU, and available connection slots.
5. Eventually, the server runs out of resources, making it unable to accept new legitimate connections.

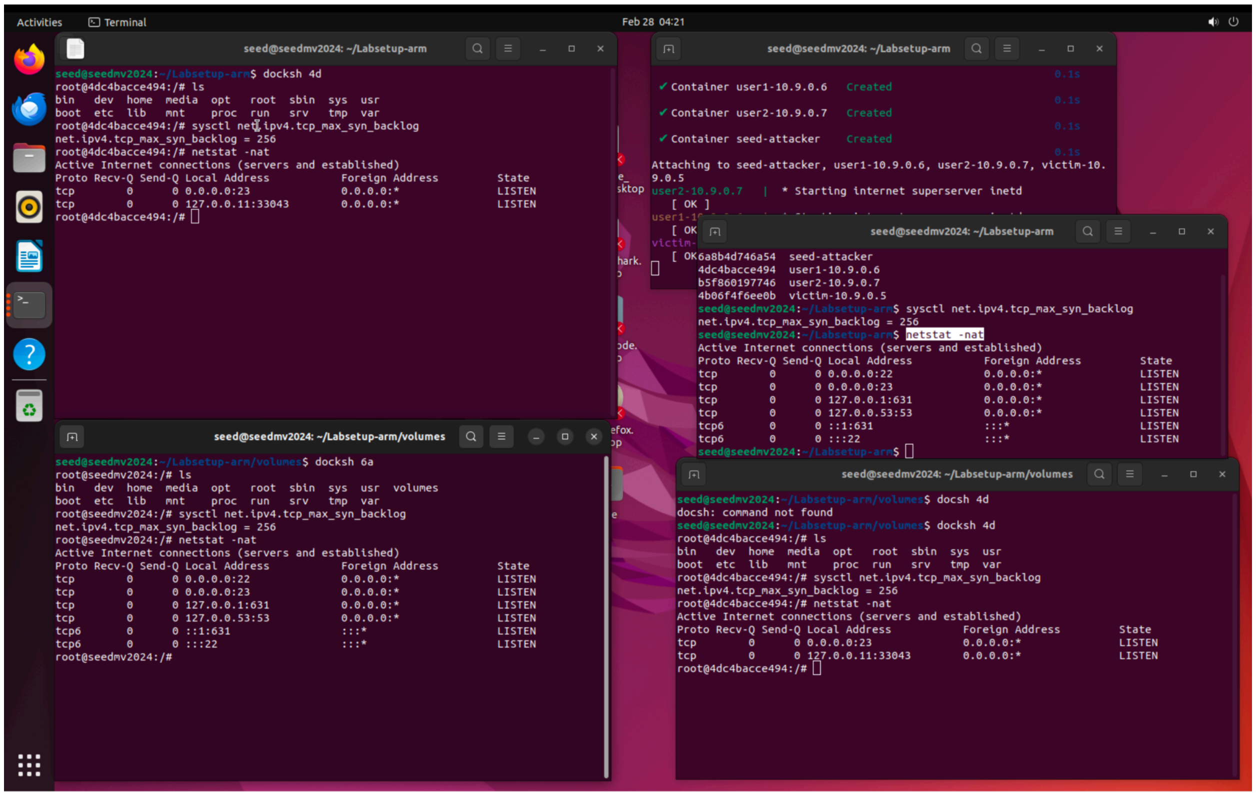
In normal TCP communication, a connection is fully established once the three-way handshake is complete. However, in a SYN Flood attack, the victim server keeps waiting for the final ACK from the attacker, holding open these incomplete connections indefinitely. Since these connections never finalize, they are referred to as half-open connections, which continuously drain system resources.



Since each half-open connection consumes memory and processing power, small-scale attacks can cripple a system when not protected. One of the largest DDoS attacks in history targeted GitHub in 2018, peaking at **1.35 Tbps**. While it primarily used Memcached reflection, SYN Floods are commonly used in similar large-scale attacks. Now these attacks affected US Government agencies in the election period in 2020, overloading hospital and healthcare providers during COVID-19 Pandemic, and even the New Zealand Stock exchange regarding trading disruptions in 2020.

For this lab, there is a way to check the amount of memory the Ubuntu OSes have with the following command **sysctl net.ipv4.tcp_max_synbacklog** . With OS the rule of thumb is that the more memory the machine has, the larger the value. Let's check this for all the systems we started (Host VM, Victim, User1, and Attacker). Lets also check the number of half open connections associated with **netstate -nat.**

- If the status is LISTEN, then it means listing for incoming IP packets on network
- If the state has connections for half open is SYN-RECV
- If 3-way handshake is finished, state is ESTABLISHED



seed@seedmv2024: ~/Labsetup-arm

```
seed@seedmv2024:~/Labsetup-arm$ docksh 4d
root@4dc4bacce494:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@4dc4bacce494:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
root@4dc4bacce494:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:33043        0.0.0.0:*               LISTEN
root@4dc4bacce494:/#
```

seed@seedmv2024: ~/Labsetup-arm

```
 ✓ Container user1-10.9.0.6   Created                                  0.1s
 ✓ Container user2-10.9.0.7   Created                                  0.1s
 ✓ Container seed-attacker    Created                                  0.1s
                                                                       0.1s
Attaching to seed-attacker, user1-10.9.0.6, user2-10.9.0.7, victim-10.
9.0.5
user2-10.9.0.7    |  * Starting internet superserver inetd
   [ OK ]
user1-1
   [ OK
victim-
   [ OK
```

seed@seedmv2024: ~/Labsetup-arm

```
6a8b4d746a54  seed-attacker
4dc4bacce494  user1-10.9.0.6
b5f860197746  user2-10.9.0.7
4b06f4f6ee0b  victim-10.9.0.5
seed@seedmv2024:~/Labsetup-arm$ sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
seed@seedmv2024:~/Labsetup-arm$ netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
seed@seedmv2024:~/Labsetup-arm$
```

seed@seedmv2024: ~/Labsetup-arm/volumes

```
seed@seedmv2024:~/Labsetup-arm/volumes$ docksh 6a
root@seedmv2024:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr  volumes
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@seedmv2024:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
root@seedmv2024:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
root@seedmv2024:/#
```

seed@seedmv2024: ~/Labsetup-arm/volumes

```
seed@seedmv2024:~/Labsetup-arm/volumes$ docsh 4d
docsh: command not found
seed@seedmv2024:~/Labsetup-arm/volumes$ docksh 4d
root@4dc4bacce494:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@4dc4bacce494:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
root@4dc4bacce494:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:33043        0.0.0.0:*               LISTEN
root@4dc4bacce494:/#
```

Now there are several mitigations that can occur as a countermeasure for SYN flooding, one that we are going to discuss is the **SYN Cookie**. Here are some of the mitigations, while not conversed, have a unique way of trying to solve the problem.

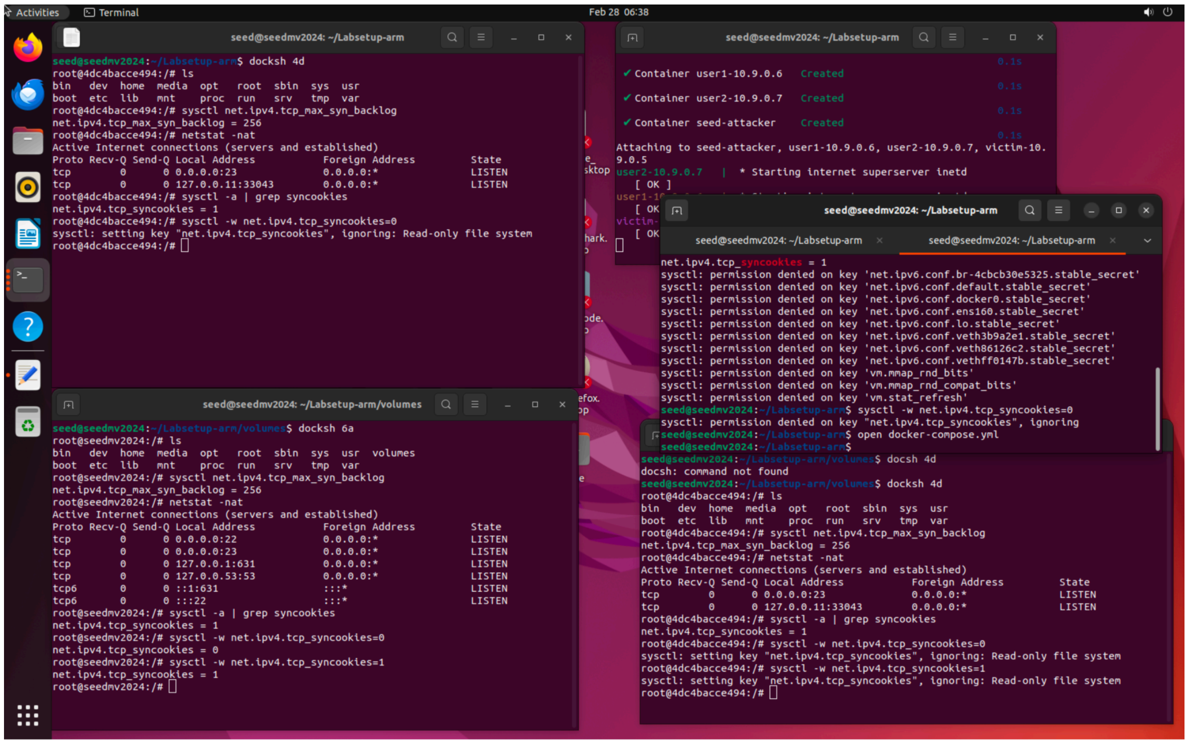
- Rate limiting: involving setting a threshold on the number of requests to a server from a source IP within a given time.
- Firewalls and Intrusion Detection Systems (IDS): the purpose of detecting and blocking traffic patterns which are in DoS and DDoS attacks.

SYN Cookie countermeasure

SYN cookies are a technique used to handle SYN flooding attacks by being able to encode connection info in the sequence number of SYN-ACK response which prevents allocation of resources until handshake is complete. This is accessible through Ubuntu and by default it is turned on. Now the way to check is a host, or machine/container, has this turned on, the way to do this is by displaying the cookie. Here are the common commands for ubuntu OS.

- **sysctl -a | grep syncookies**
  - Which is uses to display the SYN cookie flag
- **sysctl -w net.ipv4.tcp_syncookies=0**
  - This turns off the SYN cookie flag
- **sysctl -w net.ipv4.tcp_syncookies=1**

- This is to turn on the SYN cookie flag


Activities
Terminal
Feb 28 06:38
seed@seedmv2024: ~/Labsetup-arm
seed@seedmv2024: ~/Labsetup-arm/volumes

```
seed@seedmv2024:~/Labsetup-arm$ docksh 4d
root@4dc4bacce494:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@4dc4bacce494:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
root@4dc4bacce494:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:33043        0.0.0.0:*               LISTEN
root@4dc4bacce494:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 1
root@4dc4bacce494:/# sysctl -w net.ipv4.tcp_syncookies=0
sysctl: setting key "net.ipv4.tcp_syncookies", ignoring: Read-only file system
root@4dc4bacce494:/# 
```

```
                                                      0.1s
 ✓ Container user1-10.9.0.6   Created
                                                      0.1s
 ✓ Container user2-10.9.0.7   Created
                                                      0.1s
 ✓ Container seed-attacker    Created
                                                      0.1s
Attaching to seed-attacker, user1-10.9.0.6, user2-10.9.0.7, victim-10.
9.0.5
user2-10.9.0.7   |  * Starting internet superserver inetd
   [ OK ]
user1-1
   [ OK
victim-
   [ OK
```

```
net.ipv4.tcp_syncookies = 1
sysctl: permission denied on key 'net.ipv6.conf.br-4cbcb30e5325.stable_secret'
sysctl: permission denied on key 'net.ipv6.conf.default.stable_secret'
sysctl: permission denied on key 'net.ipv6.conf.docker0.stable_secret'
sysctl: permission denied on key 'net.ipv6.conf.ens160.stable_secret'
sysctl: permission denied on key 'net.ipv6.conf.lo.stable_secret'
sysctl: permission denied on key 'net.ipv6.conf.veth3b9a2e1.stable_secret'
sysctl: permission denied on key 'net.ipv6.conf.veth86126c2.stable_secret'
sysctl: permission denied on key 'net.ipv6.conf.vethff0147b.stable_secret'
sysctl: permission denied on key 'vm.mmap_rnd_bits'
sysctl: permission denied on key 'vm.mmap_rnd_compat_bits'
sysctl: permission denied on key 'vm.stat_refresh'
seed@seedmv2024:~/Labsetup-arm$ sysctl -w net.ipv4.tcp_syncookies=0
sysctl: permission denied on key "net.ipv4.tcp_syncookies", ignoring
seed@seedmv2024:~/Labsetup-arm$ open docker-compose.yml
seed@seedmv2024:~/Labsetup-arm$
```

```
seed@seedmv2024:~/Labsetup-arm/volumes$ docksh 6a
root@seedmv2024:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr  volumes
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@seedmv2024:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
root@seedmv2024:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
root@seedmv2024:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 1
root@seedmv2024:/# sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
root@seedmv2024:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@seedmv2024:/# 
```

```
seed@seedmv2024:~/Labsetup-arm/volumes$ docsh 4d
docsh: command not found
seed@seedmv2024:~/Labsetup-arm/volumes$ docksh 4d
root@4dc4bacce494:/# ls
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
root@4dc4bacce494:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
root@4dc4bacce494:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:33043        0.0.0.0:*               LISTEN
root@4dc4bacce494:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 1
root@4dc4bacce494:/# sysctl -w net.ipv4.tcp_syncookies=0
sysctl: setting key "net.ipv4.tcp_syncookies", ignoring: Read-only file system
root@4dc4bacce494:/# sysctl -w net.ipv4.tcp_syncookies=1
sysctl: setting key "net.ipv4.tcp_syncookies", ignoring: Read-only file system
root@4dc4bacce494:/# 
```

As you notice on the results of the commands that there is a strange behavior of **systctl: setting key “net.ipv4.tcp_synchookies”: Read-only file system** . This means that the container needs to be configured with **privileged: true** entry (which is in the victim server) in the