REPORT 1: BUILD WEBOS AND DEPLOYMENT FIREWALL

I. INTRODUCTION

This document serves as a guide to append the Simplified Mandatory Access Control Kernel feature onto webOS Open Source Edition (webOS OSE) version 2.20. The image of this version of webOS is supposed to be flashed and used on Raspberry Pi 4. The hardware specifications for building the machine will not be mentioned in this document. This document aims to fix the current issue with some fetch commands in the bitbake build files, along with some patches that were obsolete and thus cannot be used anymore, but persist on the remote repository.
This document was originally meant for the lecturers and educators at Da Nang University of Technology (DUT) to use as a guide, for setting up environments and building the image for Raspberry Pi 4.

II. PREREQUISITES

Before performing the firewall development and testing lab on webOS OSE, the following knowledge, tools, and setup are required:

1. Required Knowledge

- Basic understanding of the **protocols** in Networking.
- Familiarity with common network attacks such as **SYN flood**, **random source attack**, **port scanning**, and **smurf attacks**.
- Working knowledge of **iptables** command syntax and Linux command-line usage.
- Ability to analyze packets using tools like **Wireshark**.

2. Hardware

- A **Raspberry Pi 4 (8GB RAM)** with **webOS OSE 2.20** pre-flashed on a microSD card.
- An **attacker machine** (macOS or Linux) to simulate network attacks.
- *(Optional)* A **Layer 2 managed switch** for VLAN-based network isolation.

3. Software Tools

- **iptables** and **ipset**: installed on the Raspberry Pi to configure firewall rules.
- **hping3**: for generating crafted packets and simulating DoS-style attacks.
- **nmap**: for port scanning and probing the firewall.

- **Wireshark**: for packet capture and traffic inspection.

4. Initial Setup

- SSH access to the Raspberry Pi
- Necessary kernel modules (xt_hashlimit, xt_recent, xt_limit, etc.) are already built into the system or available as loadable modules.

III. CONFIGURATIONS

1. Build environment

1.1. Preparation :

Step 1: Update & Upgrade the system :

```
sudo apt update && upgrade -y
```

+ Install all dependence needed

```
sudo apt install build-essential cmake git curl

sudo apt install autoconf automake bison flex gawk libtool
pkg-config
```

+ Install git :

```
sudo apt install git-all
```

+ Clone webOS OSE from remote repository and checkout branch 2.20 :

```
git clone https://github.com/webosose/build-webos.git

cd build-webos
```

```
git checkout 2.20
```

+ Install and configure the build :

```
sudo scripts/prerequisites.sh

./mcf -p 3 -b 3 raspberrypi4-64 -p
```

- Specifications of the machine are as follows:
  - CPU: 13th Gen Intel(R) Core(TM) i5-13500 (20 number of logical CPU cores)
  - RAM: 16GB (8GBx2) DDR4 3200
  - Storage: WD PC SN740 SDDPNQD-512G-2006

+ Additional modifications :

In build-webos/meta-webososе/meta-webos-raspberrypi/recipes-kernel/linux/linux-raspberrypi :

- Append to bridge.cfg:

```
CONFIG_NETFILTER_XT_MATCH_CONNBYTES = y

CONFIG_NETFILTER_XT_MATCH_CONNLIMIT = y

CONFIG_NETFILTER_XT_MATCH_IPRANGE = y

CONFIG_NETFILTER_XT_MATCH_LENGTH = y

CONFIG_NETFILTER_XT_MATCH_LIMIT = y

CONFIG_NETFILTER_XT_MATCH_MAC = y

CONFIG_NETFILTER_XT_MATCH_QUOTA = y

CONFIG_NETFILTER_XT_TARGET_NFLOG = y

CONFIG_NF_LOG_IPV4 = y

CONFIG_NETFILTER_XT_MATCH_RECENT = y

CONFIG_NETFILTER_XT_MATCH_TCPMSS = y

CONFIG_NETFILTER_XT_TARGET_TCPMSS = y

CONFIG_NETFILTER_XT_MATCH_HASHLIMIT = y

CONFIG_NETFILTER_XT_MATCH_NFACCT = y

CONFIG_IP_SET=y

CONFIG_IP_SET_HASH_IP=y

CONFIG_IP_SET_HASH_NET=y

CONFIG_NETFILTER_XT_SET=y
```

- Append to security.cfg:

```
CONFIG_SECURITY_SMACK=y

CONFIG_DEFAULT_SECURITY=" smack "

CONFIG_DEFAULT_SECURITY_SMACK=y

CONFIG_TMPFS_XATTR=y

CONFIG_SECURITY_SMACK_BRINGUP=y

CONFIG_LSM= " lockdown , yama , loadpin , safesetid ,
integrity , smack , selinux , tomoyo , apparmor "

# kernel config to enable access to / proc/config. gz

CONFIG_IKCONFIG_PROC=y

CONFIG_IKCONFIG=y

CONFIG_PROC_FS=y

CONFIG_EXPERT=y
```

In /build-webos/meta-webososе/meta-webos/recipes-core/images :

```
CONFIG_SECURITY_SMACK = y

CONFIG_DEFAULT_SECURITY = " smack "

CONFIG_DEFAULT_SECURITY_SMACK = y

CONFIG_TMPFS_XATTR = y

CONFIG_SECURITY_SMACK_BRINGUP = y

CONFIG_LSM = " lockdown , yama , loadpin , safesetid ,
integrity , smack , selinux , tomoyo ,

apparmor "

# kernel config to enable access to / proc / config . gz

CONFIG_IKCONFIG_PROC = y

CONFIG_IKCONFIG = y

CONFIG_PROC_FS = y

CONFIG_EXPERT = y
```

In /build-webos/meta-webososе/meta-webos/recipes-core/images :

- Append to webos-image.bb:

```
WEBOS_IMAGE_EXTRA_INSTALL : append = " \

htop \

tcpdump \

kernel-module-xt-hashlimit \

ipset \

apt \ ”
```

In /build-webos/conf :

- Append to local.conf:

```
DISTRO_FEATURES:append = " smack "

DISTRO_FEATURES:append = " smack-bringup "
```

In /build-webos/meta-webososе/meta-webos/recipes-webos/localization-tool :

- Replace SRC_URI of localization-tool-native.bb :

```
SRC_URI =
"git://github.com/iLib-js/ilib-loctool-webos-dist.git;branch=
main;protocol=https"
```

In /build-webos/meta-webososе/meta-webos/recipes-upstreamable/arm-compute-library

- Replace SRCBRANCH of arm-compute-library_22.08.bb :

```
SRCBRANCH = " main "
```

In /build-webos/oe-core/meta/recipes-support/bmap-tools :

- Replace SRC_URI of bmap-tools_git.bb :

```
SRC_URI =
"git://github.com/intel/${BPN};branch=main;protocol=https"
```

In /build-webos/meta-updater/recipes-sota/aktualizr :

```
${@d.expand("https://garage-sign.s3.eu-west-1.amazonaws.com/cl
i-${GARAGE_SIGN_PV}.tgz;unpack=0;name=garagesign") if not
oe.types.boolean(d.getVar('GARAGE_SIGN_AUTOVERSION')) else ''}
\
```

Step 2: Proceed with the Build Process

```
source oe-init-build-env
bitbake webos-image
```

Step 3: After a successful build, extract the wic.bz2 file

+ File wic.bz2 đặt trong thư mục BUILD/deploy/image/raspberrypi/

```
bunzip2 -d <wic.bz2 file>
```

Step 4: After a successful build, write the image to the SD card. Write to the SD card:

```
lsblk
```

Unmount the device to ensure a safe operation :

```
sudo umount /dev/sd<xN>
```

Enter fdisk.

```
sudo fdisk /dev/sd<x>
```

2. Flashing the Image

This section describes how to flash the webOS OSE image to a microSD card, for each host operating system.
Check the device name of the microSD card using the following command. Refer to the Preparing a microSD Card section for an explanation of the device name denotation.

```
lsblk
```

To flash the image to the microSD card, run the following commands.

```
sudo umount /dev/sd<xN>

sudo dd bs=4M if=<path to the webOS OSE image file> of=/dev/sd<x>

sudo umount /dev/sd<xN>
```

+ With path to the webOS OSE image file is .wic file

For dd command, you must pass sd<x> (without the suffix number) to the of operand. sd<x> indicates the mass storage device, not the partition.

- **Flashing Command Example for Linux**

```
sudo umount /dev/sdb1

sudo dd bs=4M if=./webos-image-raspberrypi4.rootfs.wic
of=/dev/sdb
```

```
sudo umount /dev/sdb1
```

3. Development firewall

1. Port Scanning Protection

Port scanning is an attacker's action to detect open ports on a server, thereby finding running services and potential vulnerabilities

➔ **Idea to block:** port scanning actions often generate a large number of packets with similar characteristics in a short time (for example, many RST packets from a server when scanning closed ports). We will limit the frequency of these packets, if it exceeds the threshold, it is a scanning action and will be blocked

Executable command line

```
Iptables -N port-scanning
```

➔ Create a separate chain to help manage the rules in a neat, organized and easier to read way. We will redirect suspicious packets into this port-scanning chain for separate processing.

```
Iptables -A port-scanning -p tcp –tcp-flags SYN,ACK,FIN,RST RST -m
limit –limit 1/s –limit-burst 2 -j RETURN
```

- tcp-flags SYN,ACK,FIN,RST RST: This is the core part to determine the packet type. This command tells iptables to only consider these 4 flags in the TCP header of the packet and the RST flag is required to be enabled

➔ This rule will match a TCP packet in which only the RST flag is set (and the SYN,ACK,FIN flags are not set). This packet is usually sent from your server to deny a connection to a closed port, when port scanning will cause many such packets. So this

rule will limit the frequency of RST packets, so that only valid RST packets can get through.

```
iptables -A port-scanning -j DROP
```

What the command means: If a packet is sent to the port-scanning chain and it does not match the above rule (frequency exceeds the allowed threshold), it will be pushed down to this rule and dropped.

➔ When an attacker scans quickly, they will generate a series of RST packets, the first few packets may go through but as soon as the threshold is exceeded, all subsequent packets will be DROP. The attacker will not receive a response, causing the scanner to hang or falsely report that the port is filtered, making the scan useless.

2. Stealth Scans

This technique uses specially modified, non-standard TCP packets to probe the system

Blocking idea: Blocking invalid TCP packets with common and common rules is often used in stealth scans

```
iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j
DROP
```

Command meaning: Block packets with connection status INVALID. These are packets that do not belong to any existing connection and have invalid structure

➔ Helps eliminate specially crafted packets that do not follow the standard from the beginning and does not give them a chance to consume system resources

```
iptables -A INPUT -p tcp --tcp-flags [MASK] [COMP] -j DROP
```

- NULL Scan Attack ('--tcp-flags ALL NONE'):
    - ALL: Mask is all flags (SYN, ACK, FIN, RST, PSH, URG)
    - NONE: Requires that no flags in the mask are set.
    - How it works: For an incoming TCP packet, check all flags. If all 6 flags are off, then the packet matches the rule condition and when it matches, a DROP (silently dropped) action is performed.
    - Blocking mechanism: When the attacker's NULL scan packet arrives at the server, it will be blocked by iptables and checked to see that no flag is enabled (NULL), it will DROP the packet.
- XMAS Scan Attack (`--tcp-flags ALL ALL`):
    - ALL: Mask is all flags
    - ALL: Requires that all flags in the mask be set
    - How it works: If a TCP packet has all 6 flags enabled at the same time, it matches the condition and is DROP
    - Blocking mechanism: XMAS packet will be blocked by iptables and checked if all 6 flags are enabled then match the condition and DROP
- FIN Scan Attack (`--tcp-flags FIN,ACK FIN`):
    - FIN,ACK: Mask là FIN flag and ACK flag
    - FIN: The test condition is that the FIN flag is on
    - How it works: For incoming TCP packets, only 2 flags FIN and ACK are checked. If FIN is on and ACK is off, then the packet meets the condition and is DROP.
    - Blocking mechanism: FIN Scan packet (only has FIN flag) arrives at server. Iptables checks and finds that FIN is on but ACK is off and matches the condition so DROP the packet

* The following commands will be similar to the above command only classifying abnormal packets in many cases:

```
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   FIN,SYN,RST,PSH,ACK,URG NONE -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   FIN,SYN FIN,SYN -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   SYN,RST SYN,RST -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   FIN,RST FIN,RST -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   FIN,ACK FIN -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   ACK,URG URG -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   ACK,FIN FIN -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags
   ACK,PSH PSH -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL
   ALL -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL
   NONE -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL
   FIN,PSH,URG -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL
   SYN,FIN,PSH,URG -j DROP
+  iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL
   SYN,RST,ACK,FIN,URG -j DROP
```

3. SYN Flood

A SYN Flood attack occurs when an attacker sends a large number of SYN packets (connection requests) but never completes the TCP three-way handshake. This exhausts

the server's resources, as it keeps sending SYN-ACK packets and waits for a response that never arrives, preventing it from accepting legitimate connections.

```
iptables -t mangle -A PREROUTING -p tcp ! --syn -m conntrack
--ctstate NEW -j DROP
```

The basic logic of TCP requires that a new connection must start with a SYN packet. Therefore, any new packet (requesting to establish a new connection) that does not contain a SYN flag is considered a forged packet.

If the system receives a non-SYN packet (such as ACK, FIN, etc.) and sees that this connection does not belong to any existing connection (ESTABLISHED) but is instead attempting to initiate a NEW connection, this violates TCP rules. Such packets will be immediately dropped to protect the integrity of the connection tracking table and to preserve system resources.

```
iptables -t mangle -A PREROUTING -p tcp -m conntrack --ctstate NEW
-m tcpmss ! --mss 536:65535 -j DROP
```

MSS (Maximum Segment Size) is the largest amount of data a machine can receive. Legitimate clients usually send a reasonable MSS value in the range of 536–65535, so packets with MSS values outside this range are likely to be forged.

➔ Poorly coded botnets or attack tools may send SYN packets with very low MSS values or even 0. This rule filters out these suspicious packets, considers them invalid, and drops them.

```
iptables -A INPUT -p tcp -m conntrack --ctstate NEW -m hashlimit
--hashlimit-name tcp_new_conn --hashlimit-above 20/second
--hashlimit-burst 20 --hashlimit-mode srcip -j DROP
```

When a valid user (from a given IP address) makes a new connection, hashlimit can create a "private counter" for that IP only. As long as the new connection rate from this IP is lower than or equal to 20 connections/second, the --hashlimit-bove (exceeded) condition will be false. Therefore, this rule does not match and their SYN packets are not dropped, allowing normal forum connections.

However, when a SYN Flood attack begins, thousands of SYN packets are sent every second. The first 20 packets will be accepted immediately, then the system will allow only 10 SYN packets per second to pass through. Any remaining SYN packets will no longer match Rule A and will fall through to Rule B and be **dropped**.

➔ This mechanism ensures that up to 60 legitimate connection requests per second are allowed, and the rest are blocked, helping to prevent the server from being overloaded.

4. IP Spoofing/ Random Source

Hackers hide their real IP addresses by crafting packets with spoofed source IP addresses. They often use ranges known as **bogon IPs** — addresses that should never appear on the public internet. The purpose is to carry out reflection attacks such as **Smurf** or **DNS Amplification**, or simply to cover their tracks.

```
ipset create blacklist hash:net family inet maxelem 1000000
```

The command requests the creation of a new set named **blacklist** with **hash:ip** as the data structure, which is a hash table used to store IP addresses or IP ranges.

```
ipset add blacklist 192.168.0.0/16

ipset add blacklist 169.254.0.0/16

ipset add blacklist 172.16.0.0/12

for i in $(seq 0 16 240); do

  echo "ipset add blacklist 10.$i.0.0/12"

done

for i in $(seq 224 239); do

  ipset add blacklist $i.0.0.0/8

done

for i in $(seq 240 255); do

  ipset add blacklist $i.0.0.0/8

done
```

The above commands are used to add bogon IP ranges to the **blacklist**. This supports **iptables** by creating an optimized list for lookup with **O(1)** complexity.

Instead of **iptables** having to scan through multiple rules for each IP range, it can simply check against the **ipset**, which is much faster and more efficient.

```
iptables -t mangle -A PREROUTING ! -s 192.168.88.252 -m set
--match-set blacklist src -j DROP
```

It interacts with **ipset** to check whether the source IP address (**src**) of the incoming packet exists in the **blacklist**. If it does, the packet will be **dropped**. However, if the packet comes from an **exception IP** (e.g., **192.168.88.252**, a machine used for configuration), it will be **excluded** from this rule and allowed through.

➔ When a packet arrives, **iptables** retrieves its source IP and performs a lookup in the **ipset blacklist**. If the IP is found in the list, the packet is immediately discarded.

5. LAND Attack Protection

The hacker creates a special TCP SYN packet, in which the source IP address and source port are identical to the destination IP and destination port. When the server receives this packet, it will try to reply to itself and fall into an infinite loop causing the server to hang.

```
iptables -t mangle -A PREROUTING -s 192.168.88.252 -d
192.168.88.252 -j DROP
```

Command meaning: Block any packet whose source and destination are both your computer's IP

6. Smurf Attack Protection

The hacker finds a machine that allows replying to requests to the broadcast address, and sends a ping to the broadcast address of that network and the source address is spoofed to the victim's IP address, making all computers in the network reply to the ping packet by sending an ICMP echo reply packet to the victim machine, resulting in amplification into hundreds of packets attacking the victim with just 1 small request.

```
iptables -t mangle -A PREROUTING -p icmp -j DROP
```

➔ Here it simply blocks all ICMP commands but also makes the server unable to ping out and no one can ping it.

IV. PROBLEMS & SOLUTIONS

1. Problem 1 - 'Qtbase'

Cause: Lack of RAM during the build process.
Solution: Increase SWAP to 10G or 20G using the command:

```
sudo fallocate -l 20G /swapfile && sudo chmod 600 /swapfile &&
sudo mkswap /swapfile && sudo swapon /swapfile
```

This helps prevent RAM overflow.

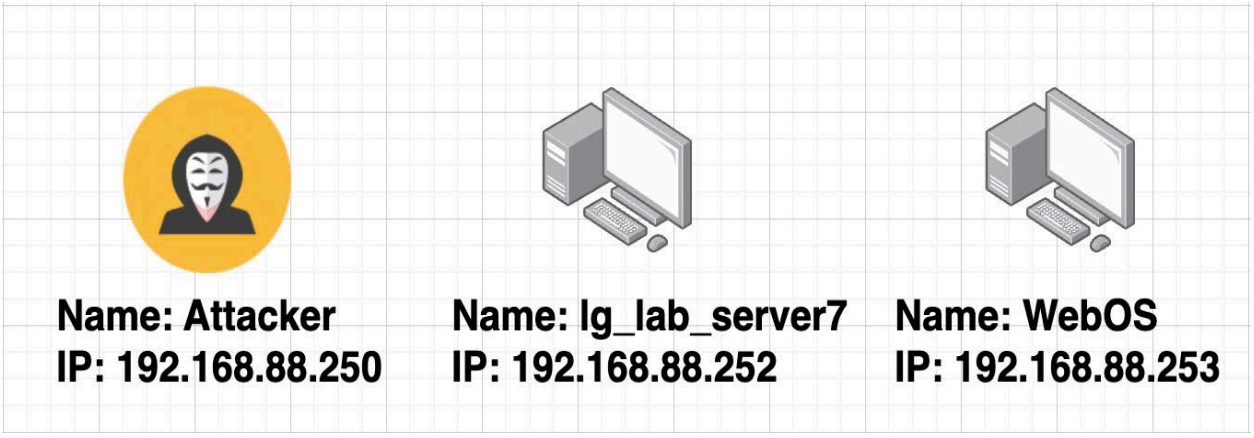
2. Problem 2 - 'Aktualize do_fetch'

Cause: The path points to an old CLI version and needs to be updated.

Solution: Update the version in the **GARAGE_SIGN_PV** variable to the latest version. Then, replace the checksum lines **SRC_URI[garagesign.md5sum]** and **SRC_URI[garagesign.sha256sum]** with the new hash values of the updated file to avoid **md5 checksum** and **sha256 checksum** errors.

V. TESTING

- Model of machines in the network:



- Attacker IP:

```
Wireless LAN adapter Local Area Connection* 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::7f56:c56c:c93e:66e9%19
   IPv4 Address. . . . . . . . . . . : 192.168.88.250
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.88.1

Ethernet adapter vEthernet (WSL (Hyper-V firewall)):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::c45b:3d0b:48f1:fa5b%55
   IPv4 Address. . . . . . . . . . . : 172.20.0.1
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . :
PS C:\Users\kieth>
```

- Host IP:

```
lg_lab_server7@lglabserver7:~/back/build/patch$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default q
len 1000
    link/ether 2c:58:b9:8b:51:f9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.88.252/24 metric 100 brd 192.168.88.255 scope global dynamic enp1s0
       valid_lft 570sec preferred_lft 570sec
    inet6 fe80::2e58:b9ff:fe8b:51f9/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group defau
lt
    link/ether 1a:4a:ac:8f:9a:31 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

- Server IP:

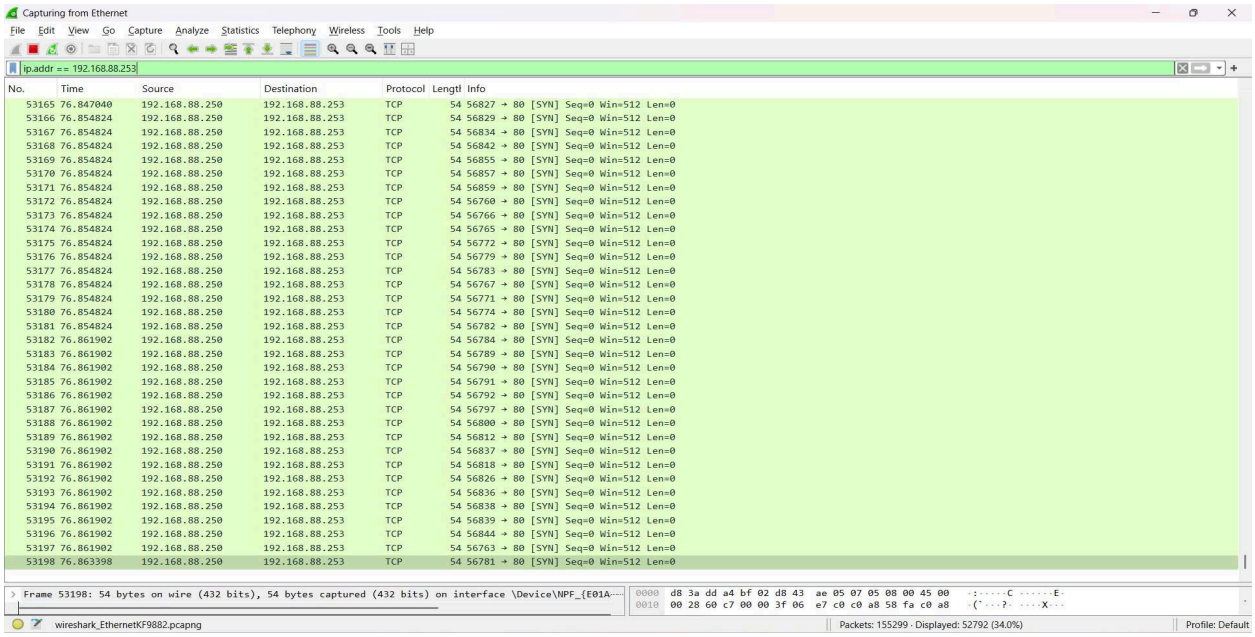
```
root@raspberrypi4-64:/var/rootdirs/home/root# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default q
len 1000
    link/ether d8:3a:dd:a4:bf:02 brd ff:ff:ff:ff:ff:ff
    inet 192.168.88.253/24 brd 192.168.88.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::da3a:ddff:fea4:bf02/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether d8:3a:dd:a4:bf:04 brd ff:ff:ff:ff:ff:ff
```

- Verify that the Server and Host are on the same network by pinging the host

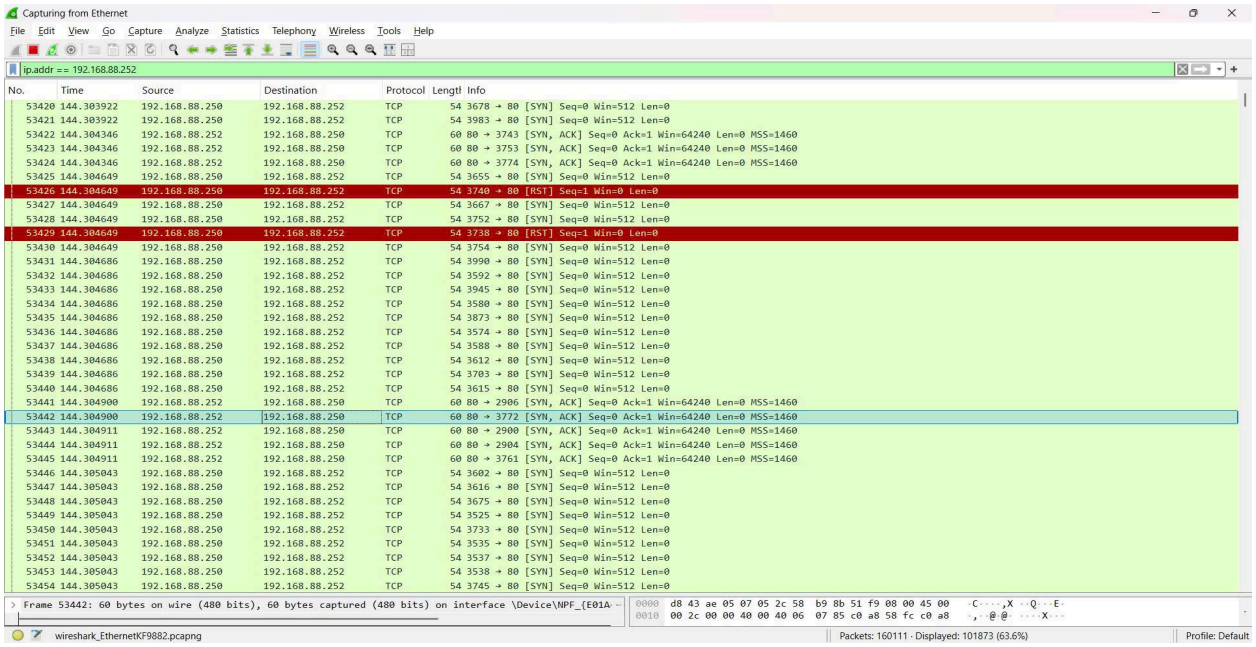
53329	129.467407	192.168.88.250	192.168.88.252	ICMP	98 Echo (ping) request id=0x03e8, seq=1/256, ttl=63 (reply in 53330)
53330	129.469117	192.168.88.252	192.168.88.250	ICMP	98 Echo (ping) reply id=0x03e8, seq=1/256, ttl=64 (request in 53329)
53331	130.468062	192.168.88.250	192.168.88.252	ICMP	98 Echo (ping) request id=0x03e8, seq=2/512, ttl=63 (reply in 53332)
53332	130.469537	192.168.88.252	192.168.88.250	ICMP	98 Echo (ping) reply id=0x03e8, seq=2/512, ttl=64 (request in 53331)
53334	131.481666	192.168.88.250	192.168.88.252	ICMP	98 Echo (ping) request id=0x03e8, seq=3/768, ttl=63 (reply in 53335)
53335	131.483218	192.168.88.252	192.168.88.250	ICMP	98 Echo (ping) reply id=0x03e8, seq=3/768, ttl=64 (request in 53334)
53336	132.530606	192.168.88.250	192.168.88.252	ICMP	98 Echo (ping) request id=0x03e8, seq=4/1024, ttl=63 (reply in 53337)
53337	132.532204	192.168.88.252	192.168.88.250	ICMP	98 Echo (ping) reply id=0x03e8, seq=4/1024, ttl=64 (request in 53336)
53341	133.580667	192.168.88.250	192.168.88.252	ICMP	98 Echo (ping) request id=0x03e8, seq=5/1280, ttl=63 (reply in 53342)
53342	133.582284	192.168.88.252	192.168.88.250	ICMP	98 Echo (ping) reply id=0x03e8, seq=5/1280, ttl=64 (request in 53341)
53348	134.630121	192.168.88.250	192.168.88.252	ICMP	98 Echo (ping) request id=0x03e8, seq=6/1536, ttl=63 (reply in 53349)
53349	134.632109	192.168.88.252	192.168.88.250	ICMP	98 Echo (ping) reply id=0x03e8, seq=6/1536, ttl=64 (request in 53348)
53350	135.680106	192.168.88.250	192.168.88.252	ICMP	98 Echo (ping) request id=0x03e8, seq=7/1792, ttl=63 (reply in 53351)
53351	135.682119	192.168.88.252	192.168.88.250	ICMP	98 Echo (ping) reply id=0x03e8, seq=7/1792, ttl=64 (request in 53350)
53368	144.302032	192.168.88.250	192.168.88.252	TCP	54 3711 → 80 [SYN] Seq=0 Win=512 Len=0

1. SYN Flood & Random Source Attack

- Verify the effectiveness of the attack by checking the packets sent to the server



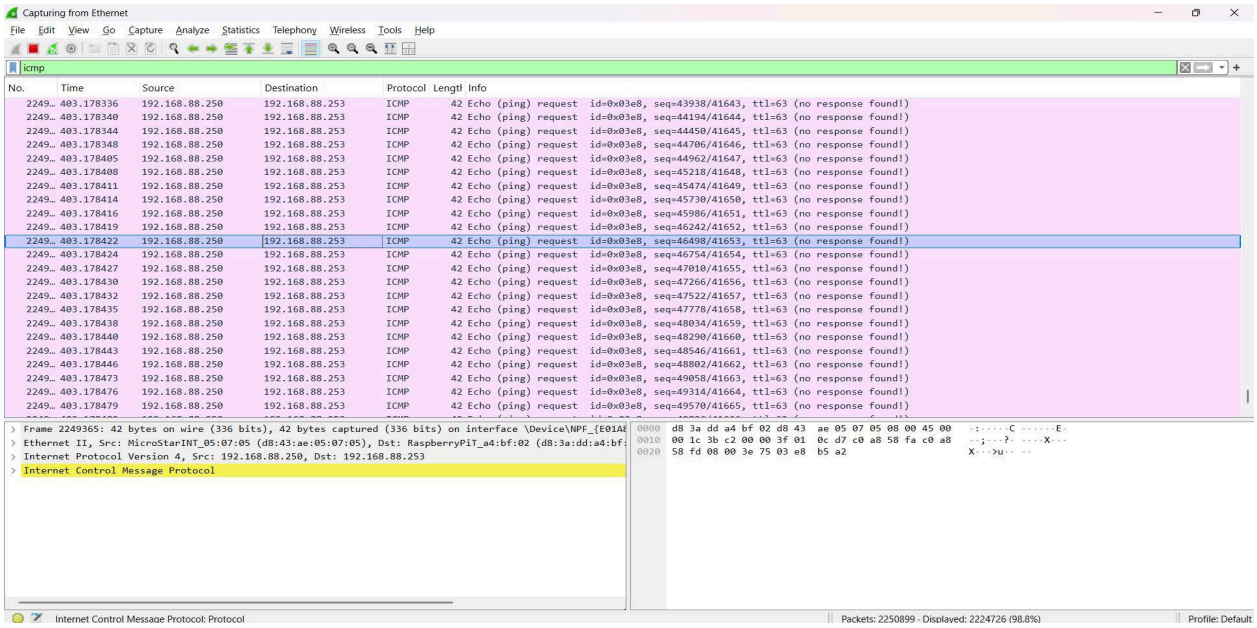
Here, the attacker has sent over 50,000 packets, but the server did not respond. Try verifying again by launching the attack on the host machine instead:



It was observed that the host machine still attempted to respond to the packets sent by the attacker.

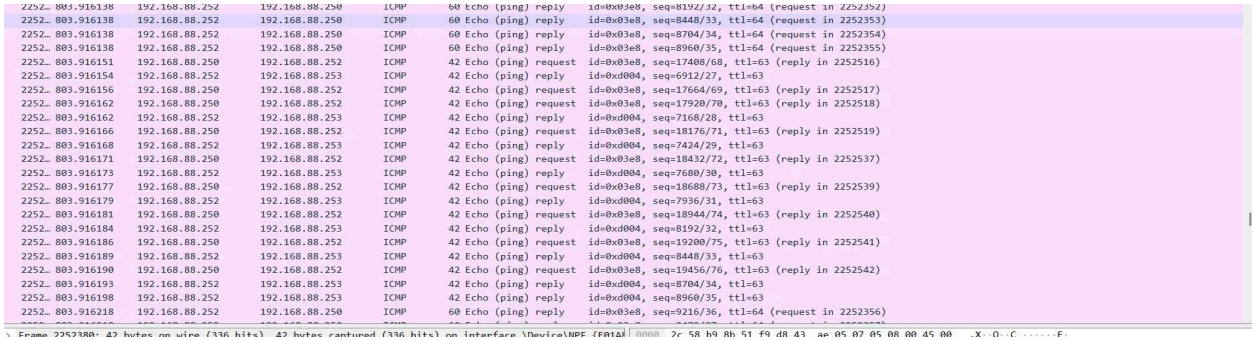
➔ Meanwhile, the server's firewall successfully blocked the attacker's packets and did not process or respond to them, thereby preventing resource consumption caused by handling fake packets

2. Smurf Attack

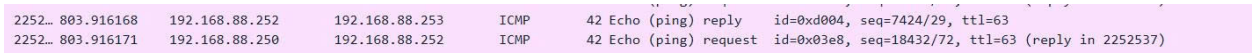


- The server successfully mitigated the attack by not responding to any ICMP packets.

- To confirm this behavior, we will now try the same command on the Host machine

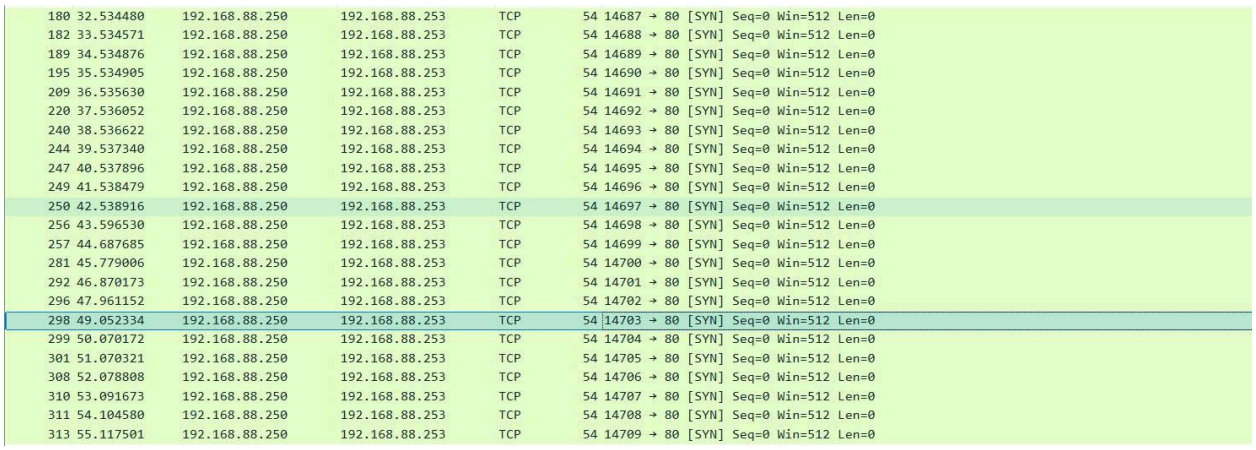


- As shown in the figure, the host machine still attempts to respond to the ICMP flood packets if the firewall is not configured



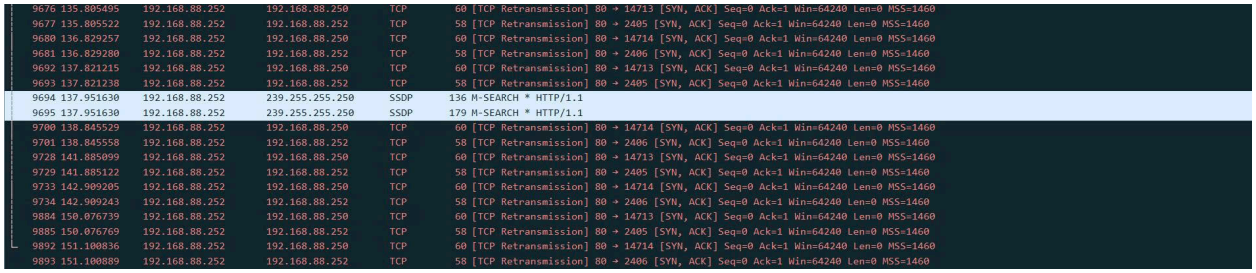
3. Land Attack

- In the case of a Land Attack, the server also does not respond and blocks any packets with both the source and destination IP addresses set to its own



  - Here, we can see that the server does not respond to itself due to the following rule: `iptables -t mangle -A PREROUTING -s 192.168.88.252 -d 192.168.88.252 -j DROP`

- Test with the host machine:



- As shown above, the host responds to its own packets where the source and destination IP addresses are the same.

VI. APPENDIX

Iptables Parameters Table

Parameter	Description
-N <--new-chain>	This parameter creates a new user-defined rule chain.
-A <chain>	Appends a rule to the specified chain (INPUT, PREROUTING, etc)
-t <table>	Specifies the iptables table (filter, magle, raw, etc)
-p <protocol>	Specifies the protocol of packet (tcp, icmp, udp, etc)
-dport <port>	Specifies the destination port of the packet
-syn	Matches TCP packets with the SYN flag (used to initiate connections)
! -syn	Matches TCP packets without the SYN flag
-tcp-flags <flags>	Checks TCP flag in packets (SYN, ACK, FIN, RST, PSH, URG)
-ctstate <state>	Checks connection state (NEW, ESTABLISHED, RELATED, INVALID)
-mss <range>	Checks the mss (Maximum Segment Size) value of TCP packets
-limit <value>	Limits the rate of packet processing (used with the limit module)
-limit-burst <value>	Maximum number of packets allowed before rate limiting applies
-hashlimit-name <name>	Assign a unique identifier to the hash table that will store the counters. All rules sharing the same name will share the same limit and configuration.
--hashlimit-abov	The rule will match when the rate of an object exceeds this value

e <value>	(e.g., 10/second). It is commonly used with the action -j DROP or -j REJECT in a single rule.
--hashlimit-burst <value>	The initial number of packets allowed to pass through without being subject to the rate limit. Once this number is exceeded, the rate limit begins to take effect.
-connlimit-above <n>	Limits concurrent connections from a single source IP
-match-set <set> src	Matches packet source against an IP set (used with ipset)
-j <target>	Action to take on that packet (ACCEPT, DROP, EJECT, etc)
-reject-with <type>	Specifies the rejection message type when using REJECT
-second <value>	Time windows for tracking packets (used with recent module)
-hitcount <value>	Maximum number of hits within time window (used with recent module)

Iptables Module Tables

Module	**Description**
conntrack	Tracks connection state (NEW, ESTABLISHED, RELATED, INVALID)
limit	Limit the rate of packet or connection processing to prevent flooding
recent	Tracks recent packet to detect repetitive behavior (brute-force, scanning,..)
tcpmss	Checks the MSS of TCP packets to detect anomalies
connlimit	Limits the number of concurrent connections from a single source IP
set	Matches packet source or destination against an IP set

Attack Type Table

Attack Type	Description	Attack Mechanism	How iptables rules Mitigate
Port Scanning	Scan open ports to identify system vulnerabilities	Send packets (SYN, RST, etc) to multiple ports to checks responses	Block packets with abnormal TCP flags, detect scanning via recent
SYN Flood	Sends a large volume of SYN packets to exhaust server resources	Creates many half-open connections by sending SYN without completing the handshake	Limits new connections, validates SYN flags and MSS
Random Source Attack	Sends packets with spoofed source IPs to hide the attacker's identity	Uses random private sources IPs to flood the target	Blocked spoofed IPs ranges use ipset.
Smurf Attack	Sends ICMP packets with spoofed source IPs to amplify attack traffic	Spoofs victim's IP, send ICMP to networks to elicit large responses	Blocked all ICMP packets
LAND Attack	Sends packet with identical source and destination IPs to cause loop	Sends SYN packets with source and destination as IP server , causing system loops	Block packets with matching source and destination IPs checks for invalid IP ranges