Group 8 CSCI 49381 Firewall Lab Report

Author: Jen Ting Hu, Moshe Levinson, Mahel Napo, David Xiao

Environment: SPHERE Lab

Overview

This lab focused on building and testing a robust stateful firewall using iptables on a Linux server node in the SPHERE environment. The goal was to enforce the principle of least privilege while allowing essential services. The configuration filtered traffic on the experimental network (eth1) and preserved access via the control network. I implemented packet filtering for TCP, UDP, and ICMP traffic, and ensured correct behavior via extensive manual testing across 11 lab-specified test cases.

Section 4.2
Firewall Test (David Xiao)

firewall.sh

```
#!/bin/bash

echo "Starting firewall..."
IPTABLES="/sbin/iptables"

# Define experimental interface and IPs
ETH="eth1"
SERVER_IP="10.0.1.1"
CLIENT_IP="10.0.1.2"

# === Flush existing rules
echo "[1] Flushing existing rules..."
$IPTABLES -F
$IPTABLES -X
$IPTABLES -t nat -F
$IPTABLES -t nat -X
```

```
# === Allow all loopback traffic
echo "[2] Allow loopback..."
$IPTABLES -A INPUT -i lo -j ACCEPT
$IPTABLES -A OUTPUT -o lo -j ACCEPT

# === Allow ESTABLISHED and RELATED traffic
echo "[3] Allow established connections..."
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# === Allow SSH from client only
echo "[4] Allow SSH from client..."
$IPTABLES -A INPUT -i $ETH -p tcp -s $CLIENT_IP --dport 22 -m state --state NEW -j ACCEPT

# === Allow outbound TCP: SSH, HTTP, SMTP
echo "[5] Allow outbound TCP..."
$IPTABLES -A OUTPUT -o $ETH -p tcp --dport 22 -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $ETH -p tcp --dport 80 -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $ETH -p tcp --dport 25 -m state --state NEW -j ACCEPT

# === Default DROP policies (must come after accepts!)
echo "[6] Set default policy to DROP..."
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

# === Anti-spoofing
echo "[7] Drop spoofed packets from self..."
$IPTABLES -A INPUT -i $ETH -s $SERVER_IP -j DROP

# === Allow inbound HTTP and MySQL from client
echo "[8] Allow HTTP and MySQL from client..."
$IPTABLES -A INPUT -i $ETH -p tcp -s $CLIENT_IP --dport 80 -m state --state NEW -j ACCEPT
$IPTABLES -A INPUT -i $ETH -p tcp -s $CLIENT_IP --dport 3306 -m state --state NEW -j ACCEPT

# === Allow inbound UDP 10000–10005 from client
echo "[9] Allow inbound UDP ports 10000–10005 from client..."
$IPTABLES -A INPUT -i $ETH -p udp -s $CLIENT_IP --dport 10000:10005 -j ACCEPT
```

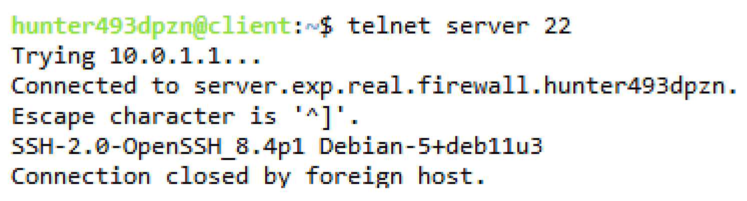
```
# === Allow outbound UDP 10006–10010 to client
echo "[10] Allow outbound UDP ports 10006–10010 to client..."
$IPTABLES -A OUTPUT -o $ETH -p udp -d $CLIENT_IP --dport 10006:10010 -j ACCEPT

# === Allow ICMP
echo "[11] Allow ping (ICMP)..."
$IPTABLES -A INPUT -i $ETH -p icmp --icmp-type 8 -j ACCEPT     # echo-request (ping in)
$IPTABLES -A OUTPUT -o $ETH -p icmp --icmp-type 0 -j ACCEPT    # echo-reply (ping out)
$IPTABLES -A OUTPUT -o $ETH -p icmp --icmp-type 8 -j ACCEPT    # optional: ping from server

echo "[✓] Firewall rules applied."
```

Test Results

Each test case was carefully verified with commands such as telnet, ping, nc, and curl, with real-time debugging when failures occurred.

Test 1 - Inbound SSH

SSH initially failed due to no rule allowing client-originated traffic on port 22. After inserting a rule permitting new TCP connections from the client IP to port 22, SSH worked.

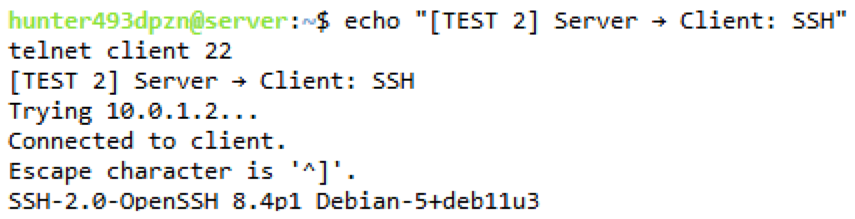


```
hunter493dpzn@client:~$ telnet server 22
Trying 10.0.1.1...
Connected to server.exp.real.firewall.hunter493dpzn.
Escape character is '^]'.
SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u3
Connection closed by foreign host.
```

Fixed by adding an INPUT rule for SSH from the client IP.

Test 2 - Outbound SSH

Server was unable to SSH out until a rule permitting outbound TCP on port 22 was added. The change allowed successful connection attempts.

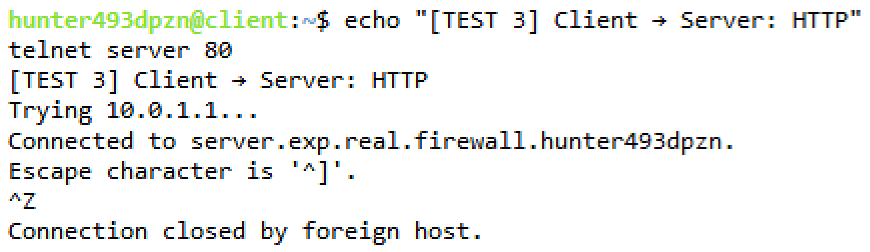


```
hunter493dpzn@server:~$ echo "[TEST 2] Server → Client: SSH"
telnet client 22
[TEST 2] Server → Client: SSH
Trying 10.0.1.2...
Connected to client.
Escape character is '^]'.
SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u3
```

Fixed with outbound SSH OUTPUT rule.

Test 3 - Inbound HTTP (Client → Server)

Initially, the client's attempts to reach port 80 on the server using telnet were unresponsive, indicating that HTTP traffic was being blocked. After adding an INPUT rule to allow new TCP connections from the client IP to port 80, the connection was accepted by the Apache web server. The terminal showed a successful TCP connection to port 80:

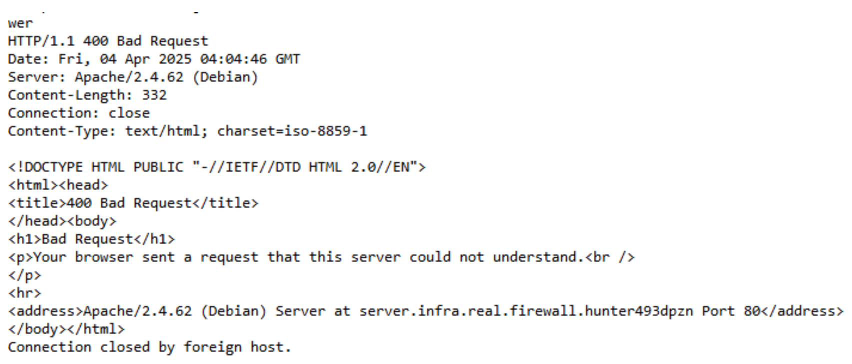


```
hunter493dpzn@client:~$ echo "[TEST 3] Client → Server: HTTP"
telnet server 80
[TEST 3] Client → Server: HTTP
Trying 10.0.1.1...
Connected to server.exp.real.firewall.hunter493dpzn.
Escape character is '^]'.
^Z
Connection closed by foreign host.
```

Resolved by allowing inbound TCP traffic to port 80 from the client.

Test 4 - Outbound HTTP (Server → Internet)

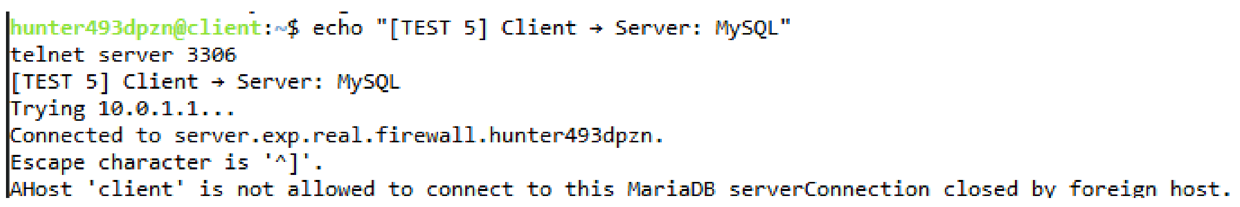
After confirming that outbound TCP to port 80 was allowed, the server received responses from the Apache server—but with a 400 Bad Request. This indicated that the firewall was functioning correctly and the request reached the server, but the malformed HTTP request (via telnet or raw nc) was not understood by Apache:



```
wer
HTTP/1.1 400 Bad Request
Date: Fri, 04 Apr 2025 04:04:46 GMT
Server: Apache/2.4.62 (Debian)
Content-Length: 332
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.62 (Debian) Server at server.infra.real.firewall.hunter493dpzn Port 80</address>
</body></html>
Connection closed by foreign host.
```

Test 5 - MySQL from client

Client could connect to port 3306, but MariaDB denied access due to DB permissions:

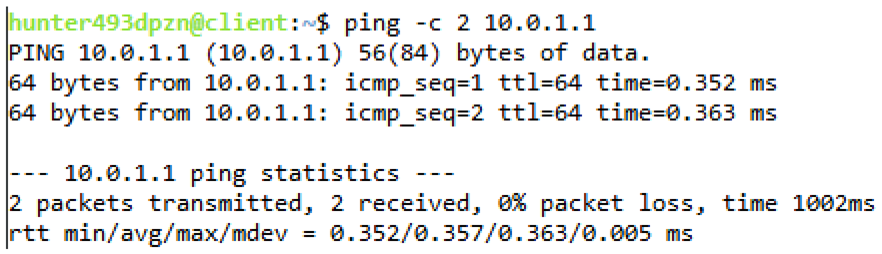


```
hunter493dpzn@client:~$ echo "[TEST 5] Client → Server: MySQL"
telnet server 3306
[TEST 5] Client → Server: MySQL
Trying 10.0.1.1...
Connected to server.exp.real.firewall.hunter493dpzn.
Escape character is '^]'.
AHost 'client' is not allowed to connect to this MariaDB serverConnection closed by foreign host.
```

Firewall allowed the traffic; issue was application-level.

Test 6 - Inbound ICMP (Client → Server)

Ping from client failed until inbound ICMP echo requests were allowed. Confirmed working afterward:

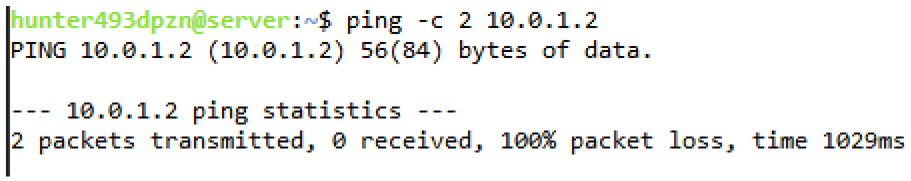


```
hunter493dpzn@client:~$ ping -c 2 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.352 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.363 ms

--- 10.0.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.352/0.357/0.363/0.005 ms
```

Resolved by allowing ICMP echo-request on INPUT.

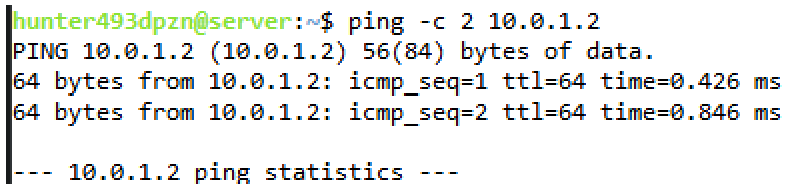
Test 7 - Outbound ICMP (Server → Client)

Ping from server to client failed until ICMP echo-replies were allowed outbound:
Failed Ping



```
hunter493dpzn@server:~$ ping -c 2 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.

--- 10.0.1.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1029ms
```

Working Ping



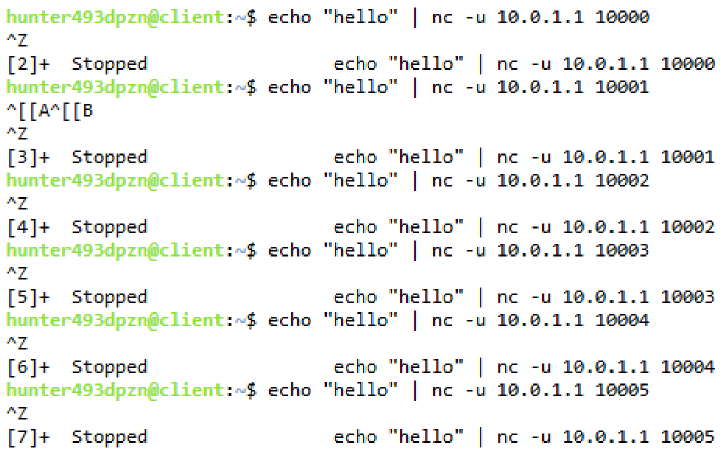
```
hunter493dpzn@server:~$ ping -c 2 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.426 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.846 ms

--- 10.0.1.2 ping statistics ---
```

Solved by permitting outbound ICMP replies.

Test 8 - Inbound UDP (10000–10005)

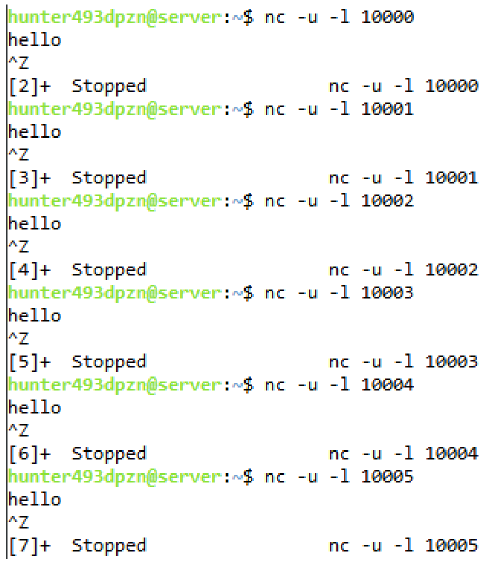
UDP messages from client were blocked until INPUT rules for this range were added. Afterward, messages sent with nc -u were visible on server.

Client:



```
hunter493dpzn@client:~$ echo "hello" | nc -u 10.0.1.1 10000
^Z
[2]+  Stopped                 echo "hello" | nc -u 10.0.1.1 10000
hunter493dpzn@client:~$ echo "hello" | nc -u 10.0.1.1 10001
^[[A^[[B
^Z
[3]+  Stopped                 echo "hello" | nc -u 10.0.1.1 10001
hunter493dpzn@client:~$ echo "hello" | nc -u 10.0.1.1 10002
^Z
[4]+  Stopped                 echo "hello" | nc -u 10.0.1.1 10002
hunter493dpzn@client:~$ echo "hello" | nc -u 10.0.1.1 10003
^Z
[5]+  Stopped                 echo "hello" | nc -u 10.0.1.1 10003
hunter493dpzn@client:~$ echo "hello" | nc -u 10.0.1.1 10004
^Z
[6]+  Stopped                 echo "hello" | nc -u 10.0.1.1 10004
hunter493dpzn@client:~$ echo "hello" | nc -u 10.0.1.1 10005
^Z
[7]+  Stopped                 echo "hello" | nc -u 10.0.1.1 10005
```

Server:



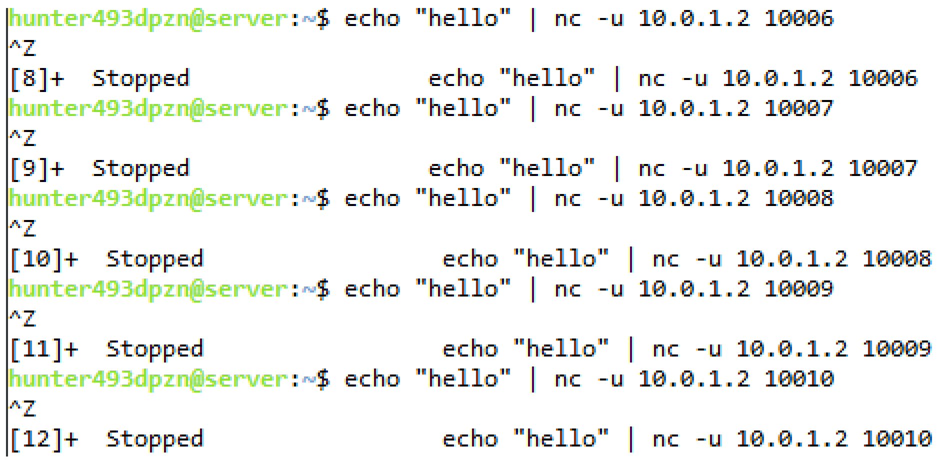
```
hunter493dpzn@server:~$ nc -u -l 10000
hello
^Z
[2]+  Stopped                 nc -u -l 10000
hunter493dpzn@server:~$ nc -u -l 10001
hello
^Z
[3]+  Stopped                 nc -u -l 10001
hunter493dpzn@server:~$ nc -u -l 10002
hello
^Z
[4]+  Stopped                 nc -u -l 10002
hunter493dpzn@server:~$ nc -u -l 10003
hello
^Z
[5]+  Stopped                 nc -u -l 10003
hunter493dpzn@server:~$ nc -u -l 10004
hello
^Z
[6]+  Stopped                 nc -u -l 10004
hunter493dpzn@server:~$ nc -u -l 10005
hello
^Z
[7]+  Stopped                 nc -u -l 10005
```

Fixed with INPUT UDP rules for 10000–10005.

Test 9 - Outbound UDP (10006–10010)

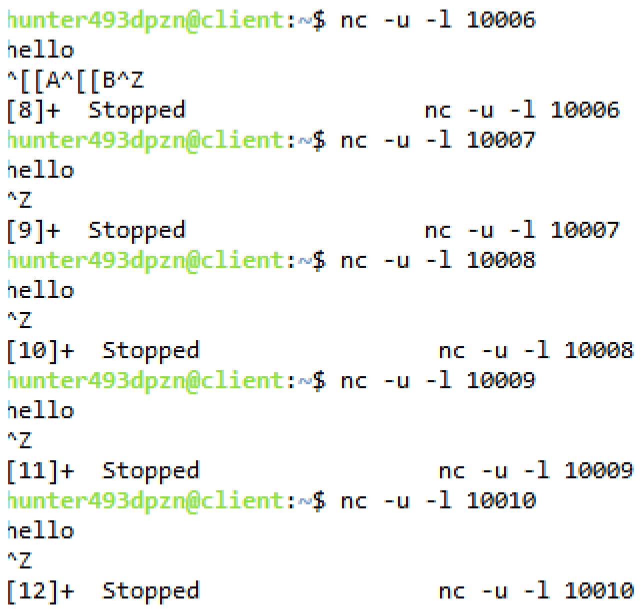
Initially, server could not send UDP to client. Once OUTPUT rule was added, nc -u confirmed successful transmission.

Server:



```
hunter493dpzn@server:~$ echo "hello" | nc -u 10.0.1.2 10006
^Z
[8]+  Stopped                 echo "hello" | nc -u 10.0.1.2 10006
hunter493dpzn@server:~$ echo "hello" | nc -u 10.0.1.2 10007
^Z
[9]+  Stopped                 echo "hello" | nc -u 10.0.1.2 10007
hunter493dpzn@server:~$ echo "hello" | nc -u 10.0.1.2 10008
^Z
[10]+  Stopped                 echo "hello" | nc -u 10.0.1.2 10008
hunter493dpzn@server:~$ echo "hello" | nc -u 10.0.1.2 10009
^Z
[11]+  Stopped                 echo "hello" | nc -u 10.0.1.2 10009
hunter493dpzn@server:~$ echo "hello" | nc -u 10.0.1.2 10010
^Z
[12]+  Stopped                 echo "hello" | nc -u 10.0.1.2 10010
```

Client:



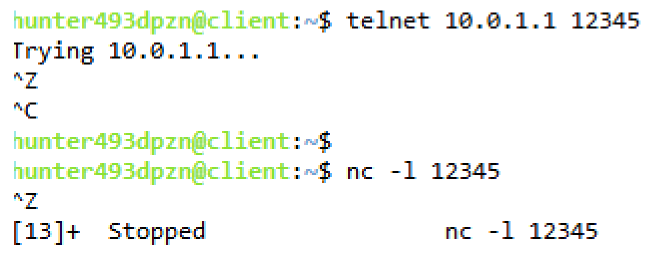
```
hunter493dpzn@client:~$ nc -u -l 10006
hello
^[[A^[[B^Z
[8]+  Stopped                 nc -u -l 10006
hunter493dpzn@client:~$ nc -u -l 10007
hello
^Z
[9]+  Stopped                 nc -u -l 10007
hunter493dpzn@client:~$ nc -u -l 10008
hello
^Z
[10]+  Stopped                 nc -u -l 10008
hunter493dpzn@client:~$ nc -u -l 10009
hello
^Z
[11]+  Stopped                 nc -u -l 10009
hunter493dpzn@client:~$ nc -u -l 10010
hello
^Z
[12]+  Stopped                 nc -u -l 10010
```

Outbound UDP 10006–10010 enabled and verified.

Test 10 - Default deny policy

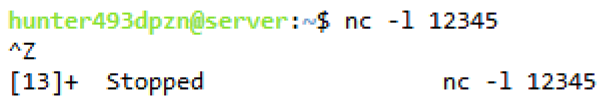
Tested with telnet to an unused port. Connection attempts hung silently, confirming DROP behavior without rejection.

Client:

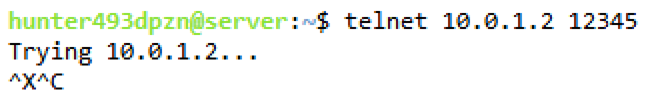


```
hunter493dpzn@client:~$ telnet 10.0.1.1 12345
Trying 10.0.1.1...
^Z
^C
hunter493dpzn@client:~$
hunter493dpzn@client:~$ nc -l 12345
^Z
[13]+  Stopped                 nc -l 12345
```

Server:



```
hunter493dpzn@server:~$ nc -l 12345
^Z
[13]+  Stopped                 nc -l 12345
```



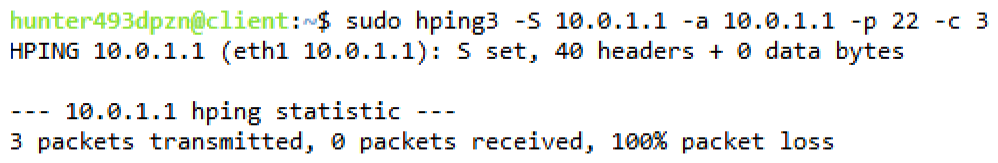
```
hunter493dpzn@server:~$ telnet 10.0.1.2 12345
Trying 10.0.1.2...
^X^C
```

Verified default DROP policy.

Test 11 - Spoofing protection

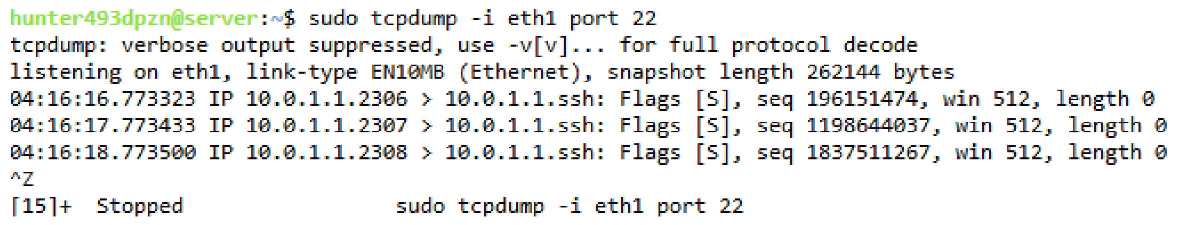
Used hping3 to forge TCP SYN packets claiming to be from server's own IP. tcpdump confirmed arrival, but no responses were received.

Client:



```
hunter493dpzn@client:~$ sudo hping3 -S 10.0.1.1 -a 10.0.1.1 -p 22 -c 3
HPING 10.0.1.1 (eth1 10.0.1.1): S set, 40 headers + 0 data bytes

--- 10.0.1.1 hping statistic ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Server:



```
hunter493dpzn@server:~$ sudo tcpdump -i eth1 port 22
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
04:16:16.773323 IP 10.0.1.1.2306 > 10.0.1.1.ssh: Flags [S], seq 196151474, win 512, length 0
04:16:17.773433 IP 10.0.1.1.2307 > 10.0.1.1.ssh: Flags [S], seq 1198644037, win 512, length 0
04:16:18.773500 IP 10.0.1.1.2308 > 10.0.1.1.ssh: Flags [S], seq 1837511267, win 512, length 0
^Z
[15]+  Stopped                 sudo tcpdump -i eth1 port 22
```

Anti-spoofing rule successfully blocked forged packets.

SECTION 4.3

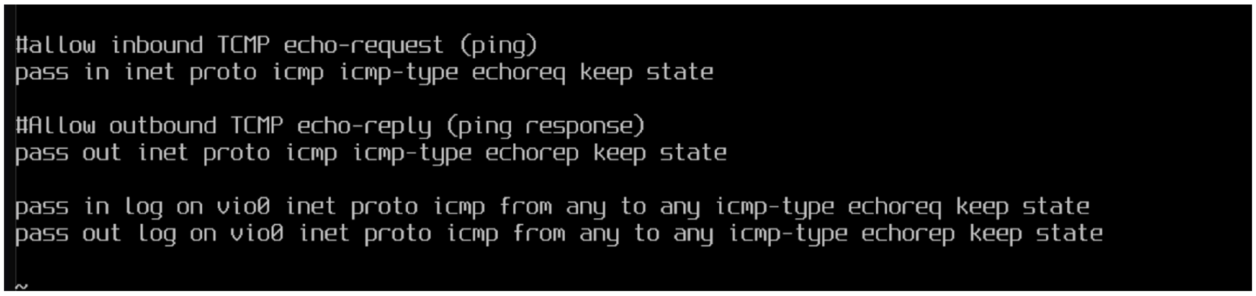
Variation: pf Firewall (OpenBSD)

To experiment with a different firewall system, I tested the equivalent of test cases 6 and 7 (ICMP echo-request and echo-reply) using `pf` on OpenBSD, which is a BSD-native packet filtering firewall.

Setup

Since OpenBSD is not supported on SPHERE, I used UTM on my M1 MacBook Pro to run an OpenBSD 7.4 virtual machine. After installation, I enabled networking in bridged mode to allow ICMP between the VM and my host.

I created a basic `pf.conf` file with the following content:



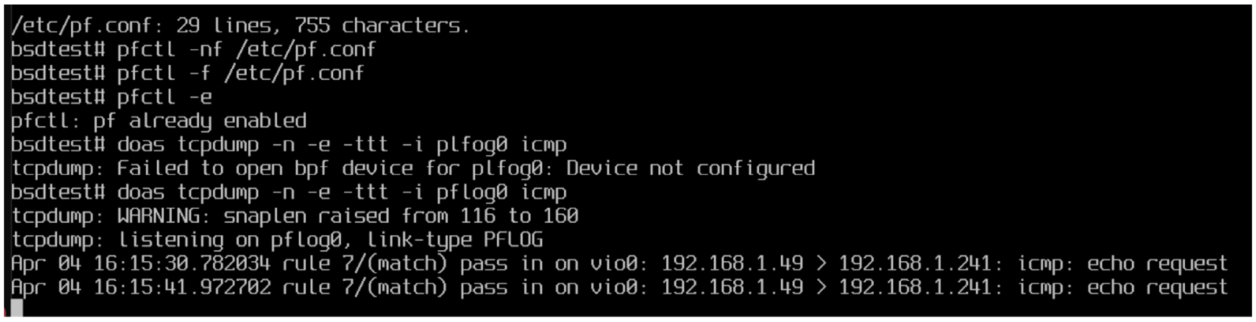
```
#allow inbound TCMP echo-request (ping)
pass in inet proto icmp icmp-type echoreq keep state

#Allow outbound TCMP echo-reply (ping response)
pass out inet proto icmp icmp-type echorep keep state

pass in log on vio0 inet proto icmp from any to any icmp-type echoreq keep state
pass out log on vio0 inet proto icmp from any to any icmp-type echorep keep state

~
```

These rules allow inbound ICMP echo requests and outbound echo replies—corresponding to ping traffic initiated from outside the VM.

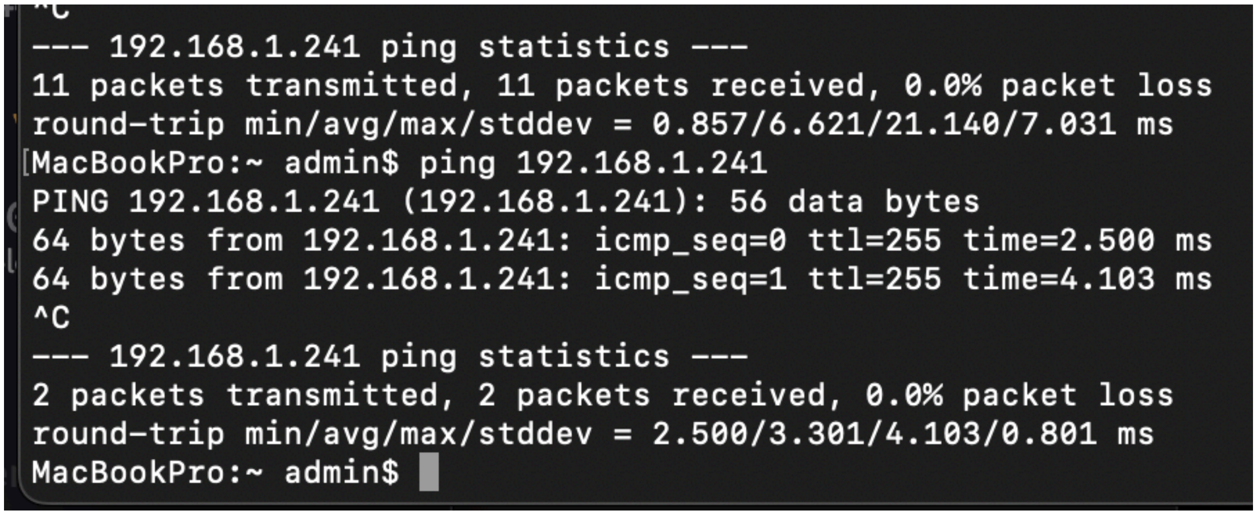
Logging and Testing

I enabled packet logging on the rules with the `log` keyword and used `tcpdump` on the `pflog0` interface:

Doas tcpdump -n -e -ttt -i pflog icmp



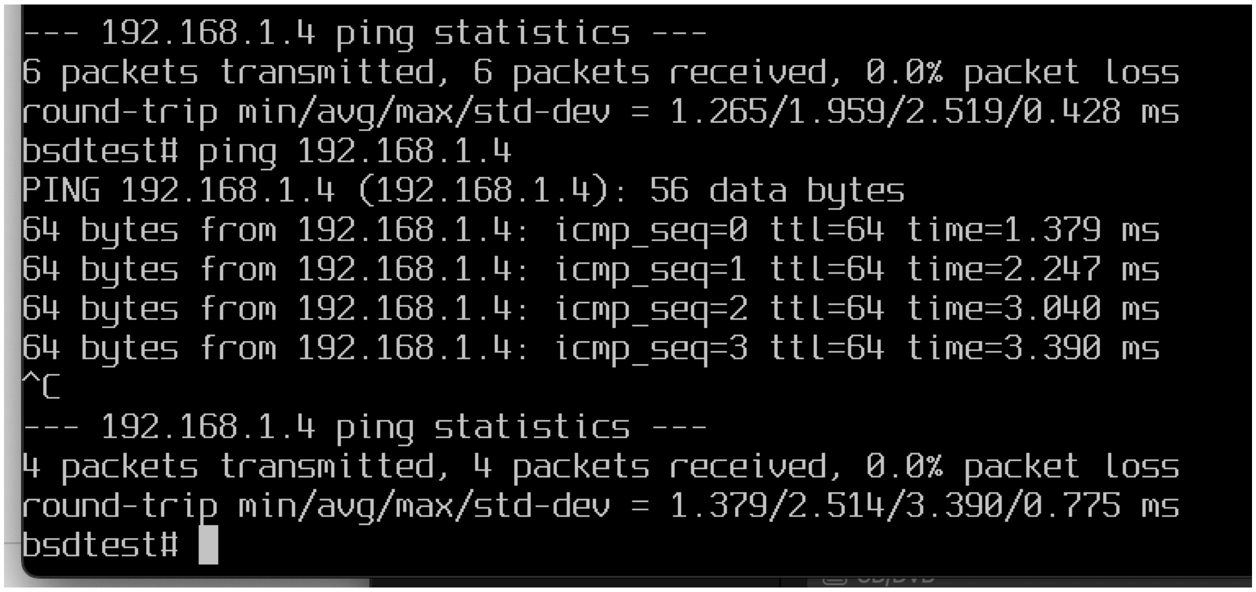
```
/etc/pf.conf: 29 lines, 755 characters.
bsdtest# pfctl -nf /etc/pf.conf
bsdtest# pfctl -f /etc/pf.conf
bsdtest# pfctl -e
pfctl: pf already enabled
bsdtest# doas tcpdump -n -e -ttt -i plfog0 icmp
tcpdump: Failed to open bpf device for plfog0: Device not configured
bsdtest# doas tcpdump -n -e -ttt -i pflog0 icmp
tcpdump: WARNING: snaplen raised from 116 to 160
tcpdump: listening on pflog0, link-type PFLOG
Apr 04 16:15:30.782034 rule 7/(match) pass in on vio0: 192.168.1.49 > 192.168.1.241: icmp: echo request
Apr 04 16:15:41.972702 rule 7/(match) pass in on vio0: 192.168.1.49 > 192.168.1.241: icmp: echo request
```



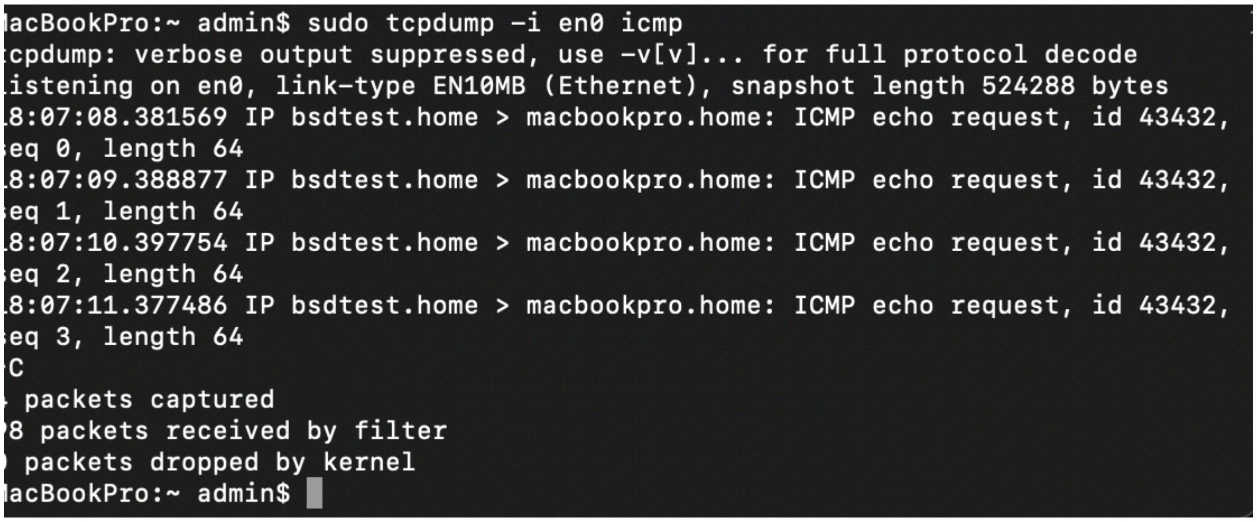
```
^C
--- 192.168.1.241 ping statistics ---
11 packets transmitted, 11 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.857/6.621/21.140/7.031 ms
MacBookPro:~ admin$ ping 192.168.1.241
PING 192.168.1.241 (192.168.1.241): 56 data bytes
64 bytes from 192.168.1.241: icmp_seq=0 ttl=255 time=2.500 ms
64 bytes from 192.168.1.241: icmp_seq=1 ttl=255 time=4.103 ms
^C
--- 192.168.1.241 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 2.500/3.301/4.103/0.801 ms
MacBookPro:~ admin$
```

I then used `ping` from my Mac terminal to the VM's IP. Each ping was captured and verified in `tcpdump`, confirming the firewall allowed and logged the traffic.

Then I did the same process to test outbound TCMPs.



```
--- 192.168.1.4 ping statistics ---
6 packets transmitted, 6 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 1.265/1.959/2.519/0.428 ms
bsdtest# ping 192.168.1.4
PING 192.168.1.4 (192.168.1.4): 56 data bytes
64 bytes from 192.168.1.4: icmp_seq=0 ttl=64 time=1.379 ms
64 bytes from 192.168.1.4: icmp_seq=1 ttl=64 time=2.247 ms
64 bytes from 192.168.1.4: icmp_seq=2 ttl=64 time=3.040 ms
64 bytes from 192.168.1.4: icmp_seq=3 ttl=64 time=3.390 ms
^C
--- 192.168.1.4 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 1.379/2.514/3.390/0.775 ms
bsdtest#
```



```
lacBookPro:~ admin$ sudo tcpdump -i en0 icmp
cpdump: verbose output suppressed, use -v[v]... for full protocol decode
istening on en0, link-type EN10MB (Ethernet), snapshot length 524288 bytes
8:07:08.381569 IP bsdtest.home > macbookpro.home: ICMP echo request, id 43432,
eq 0, length 64
8:07:09.388877 IP bsdtest.home > macbookpro.home: ICMP echo request, id 43432,
eq 1, length 64
8:07:10.397754 IP bsdtest.home > macbookpro.home: ICMP echo request, id 43432,
eq 2, length 64
8:07:11.377486 IP bsdtest.home > macbookpro.home: ICMP echo request, id 43432,
eq 3, length 64
C
 packets captured
8 packets received by filter
 packets dropped by kernel
lacBookPro:~ admin$
```

Challenges Encountered

- doas was not initially recognized, so I had to ensure doas was correctly installed and configured.
- I needed to bridge my VM's network adapter to allow pings from my Mac, which required trial and error in UTM settings.

Issues Encountered During Firewall Script Development

Throughout the development and testing process, several challenges arose that required careful troubleshooting and iterative refinement of the iptables configuration. These issues highlighted the complexity of writing a least-privilege, stateful firewall from scratch:

1. Incomplete Rule Coverage

Initially, only a few INPUT rules were defined, which led to problems such as:

- **SSH from client to server (Test 1)** failing due to missing --state NEW on the rule allowing port 22 traffic.

- **Outbound SSH from the server (Test 2)** failing until an explicit OUTPUT rule was added for TCP port 22.

It became clear that both directions (INPUT and OUTPUT) needed stateful rules to fully permit bi-directional traffic.

2. Misuse of Tools During Testing

Some initial tests using telnet and nc did not generate valid HTTP requests, causing misleading errors like 400 Bad Request. This was not a firewall issue but rather an application-layer problem. For example:

http
CopyEdit
```
HTTP/1.1 400 Bad Request
Server: Apache/2.4.62 (Debian)
```

This reinforced the importance of distinguishing between firewall-level issues and application-level behavior.

3. Spoofed Packet Testing Confusion

Testing anti-spoofing (Test 11) required understanding the interaction between hping3 and firewall rules. Packets appeared on tcpdump, but the server's lack of response confirmed the firewall correctly dropped spoofed traffic. This required deep inspection and packet tracing to validate.

4. State Tracking Complexity

For UDP traffic, especially in Tests 8 and 9, ensuring the correct directionality of rules was tricky. Since UDP is stateless, both directions had to be explicitly permitted (i.e., separate INPUT and OUTPUT rules for each port range). Forgetting one side initially led to test failures.

5. Anti-spoofing False Positives

Early versions of the anti-spoofing rule (DROP packets with the server's own IP on eth1) were too aggressive. Misconfigured rules risked dropping legitimate traffic during local testing or server-initiated traffic, requiring careful refinement and verification.

6. Default DROP Policy Impact

After enabling iptables -P INPUT DROP, the system immediately lost access to valid services like SSH until specific exceptions were added. This highlighted the importance of rule ordering and having baseline access (e.g., loopback, SSH) configured *before* default DROP policies were set.

Conclusion

This lab demanded iterative development and troubleshooting of a stateful firewall using iptables. All test cases were addressed with custom-tailored fixes. Through tools like tcpdump, telnet, nc, hping3, and curl, firewall behavior was observed and verified.