

Md Rony

Once the experiment is swapped in, we installed Apache on the server node by entering: `/share/education/TCPSYNFlood_USC_ISI/install-server`.

'/share/education/TCPSYNFlood_USC_ISI/install-flooder'.

```
processing triggers for libc-bin (2.23-0ubuntu9) ...
g++ celp.c -o celp
celpp.c: In function 'int main(int, char**)':
celpp.c:181:28: warning: format '%d' expects argument of type 'int', but argument 4 has type 'long unsigned int' [-Wformat=]
    argv[0], sizeof(buf));
                        ^
g++ floodor.c -lnet -lm -lpcap -fpermissive -o floodor
floodor.c: In function 'int flood_process(FloodParameters*)':
floodor.c:640:27: warning: invalid conversion from 'unsigned char*' to 'char*' [-fpermissive]
    thislen = sprintf(p->payload, "%c%c%c%c%c%c%c%c%c%cjoe%cexample%ccom
                          ^
In file included from /usr/include/pcap/pcap.h:52:0,
                 from /usr/include/pcap.h:43,
                 from floodor.c:17:
/usr/include/stdio.h:364:12: note:   initializing argument 1 of 'int sprintf(char*, const char*, ...)'
extern int sprintf(char *__restrict __s,
```

3.1 Generating Legitimate Traffic

```
#!/bin/bash

while true;

do wget http://5.6.7.8/index.html

sleep 1;

Done
```

With the above bash script ([http://5.6.7.8](#) describes the server node's IP address), we are able to download the index.html file once per second.

3.2 Turning off SYN Cookies

```
jjc400bj@server:~$ sudo sysctl net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 1
```

SYN Cookies are currently turned on, so we turn it off as shown in the following screenshot:

```
jjc400bj@server:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
jjc400bj@server:~$ sudo sysctl net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 0
```

3.3 Generating attack traffic

We use the following command to generate attack traffic:

‘Sudo flooder --dst 5.6.7.8 --srcmask 255.255.255.0’

3.4 Collecting traffic statistics

1. After entering ‘ip route 5.6.7.8’, we get:

```
jjc400bj@client:~$ ip route get 5.6.7.8
5.6.7.8 via 1.1.2.2 dev eth4 src 1.1.2.3
cache
jjc400bj@client:~$ █
```

This tells us that the interface leading to 5.6.7.8 is eth4. To see the traffic flowing, we type: ‘sudo tcpdump -nn -i eth4’.

```
jjc400bj@client:~$ sudo tcpdump -nn -i eth4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth4, link-type EN10MB (Ethernet), capture size 262144 bytes
12:28:05.154579 LLDP, length 232: EHP17e
12:28:35.154693 LLDP, length 232: EHP17e
12:29:05.154761 LLDP, length 232: EHP17e
12:29:35.154878 LLDP, length 232: EHP17e
12:30:05.154946 LLDP, length 232: EHP17e
12:30:35.155114 LLDP, length 232: EHP17e
12:31:05.155182 LLDP, length 232: EHP17e
12:31:35.155300 LLDP, length 232: EHP17e
12:32:05.155317 LLDP, length 232: EHP17e
12:32:35.155435 LLDP, length 232: EHP17e
12:33:05.155502 LLDP, length 232: EHP17e
12:33:35.155670 LLDP, length 232: EHP17e
```

To see IP traffic and save traffic to a file, we can enter: ‘sudo tcpdump -nn -v -i eth4 -w file.cap’. To see this file: ‘sudo tcpdump -r file.cap’.

2. Using a stopwatch, we perform the scenario:

We start tcpdump on the client node by entering: “sudo tcpdump -nn -v -i eth4 -w timedcapture-no-syn.txt”

- a. Start legitimate traffic (on client).

We run the bash script file on the client node, as described above.

- b. After 30 seconds start the attack.

On the attacker node, we enter “sudo flooder --dst 5.6.7.8 --srcmask 255.255.255.0”

- c. After 120 seconds stop the attack.
- d. After 30 seconds stop the legitimate traffic
- e. Stop the tcpdump on the client and save the file.

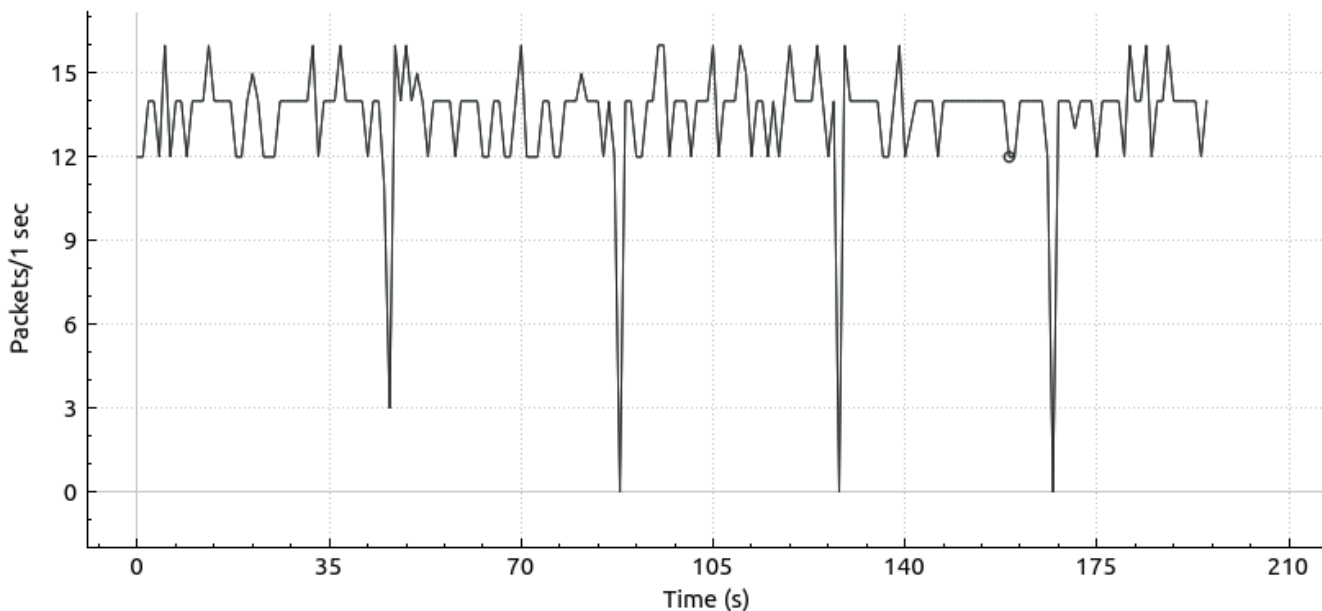
3. Turn SYN cookies on and repeat the above steps.

We turn SYN cookies on from the server node by entering: “sudo sysctl net.ipv4.tcp_syncookies=1”

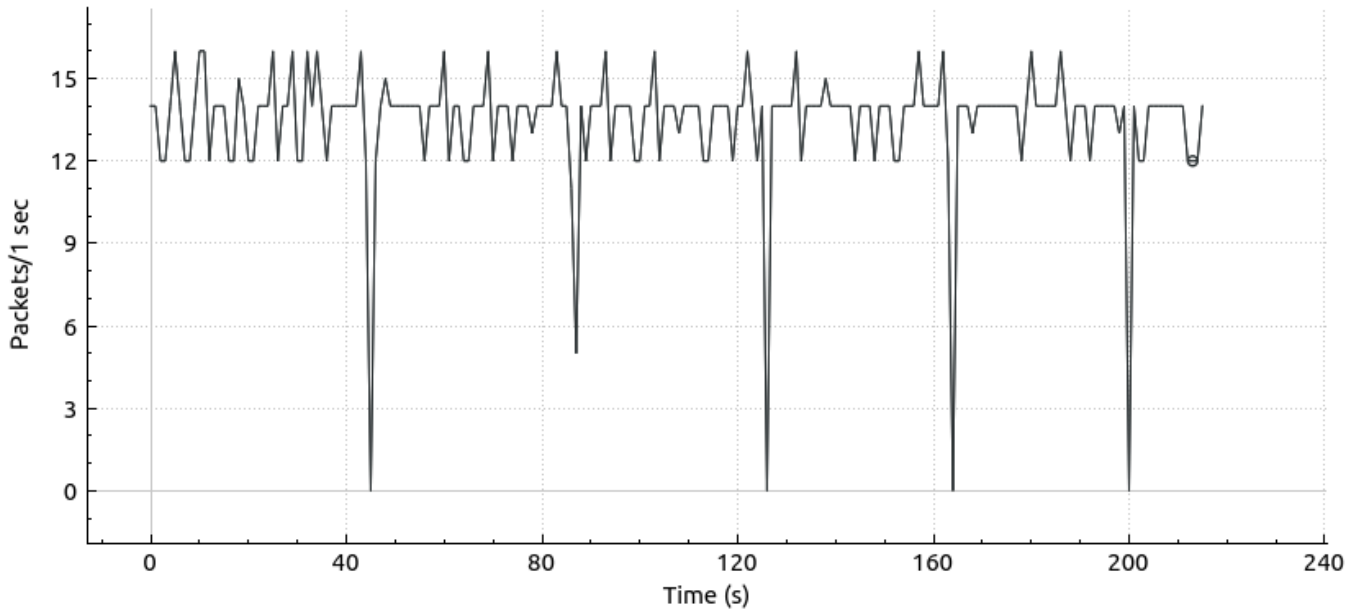
4. Using the recorded traffic files and tcpdump to read them, process the output and calculate connection duration for each TCP connection seen in the files. Connection duration is the difference between the time of the first SYN and of the ACK following a FIN-ACK (or between the first SYN and the first RESET) on a connection. Remind yourself what uniquely identifies a TCP connection, i.e. how to detect packets that belong to the same connection? If a connection did not end with an ACK following a FIN-ACK assign to it the duration of 200 s. Include two graphs in your submission, showing connection duration vs connection start time for the case without and with SYN cookies. Label the graphs so they can be distinguished and indicate on each graph using vertical lines or arrows the start and the end of the attack.

To provide answers to this prompt, we use Wireshark to view the tcpdump files, and create their respective graphs.

Wireshark IO Graphs: timedcapture-no-syn.txt



Wireshark IO Graphs: timedcapture-yes-syn.txt



3.5 No spoofing

What happens? It seems that by masking the IP address the user got more packets. This means the attack was successful.

Can you explain why this happens? by masking the IP address we got TCP packets and without only UDP.

Can you modify the attack so that it is effective without spoofing and how would you do this?

This attack could be effective if we were able to have the server send more packets.

The process for this section was to do everything in 3.1 to 3.4 except that in 3.3 we had to skip spoofing the IP address. The process was easy as the work and commands were already done by the first team member.

```
Processing triggers for systemd (215-1ubuntu2) ...  
Processing triggers for ureadahead (0.100.0-19) ...  
Processing triggers for ufw (0.35-0ubuntu2) ...  
Setting up apache2 (2.4.18-2ubuntu3.9) ...  
jjc400bh@server:~$
```

Screenshot of the installation of Apache2 in the server

```
g++ flooder.c -lnet -lm -lpcap -fpermiss
flooder.c: In function 'int flood_proces
flooder.c:640:27: warning: invalid conve
-fpermissive]
    thislen = sprintf(p->payload, "%c%
In file included from /usr/include/pcap/
    from /usr/include/pcap.
    from flooder.c:17:
/usr/include/stdio.h:364:12: note: ini
r*, const char*, ...)'
    extern int sprintf (char *__restrict __
jjc400bh@attacker:~$
```

Screenshot of flooder installation

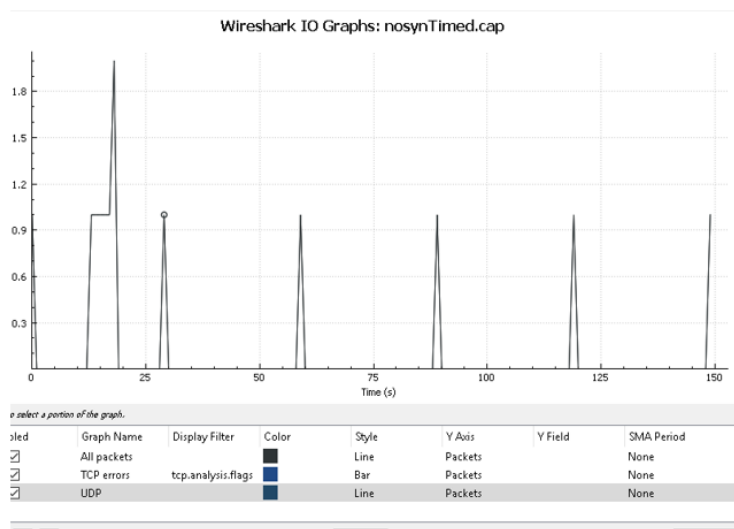
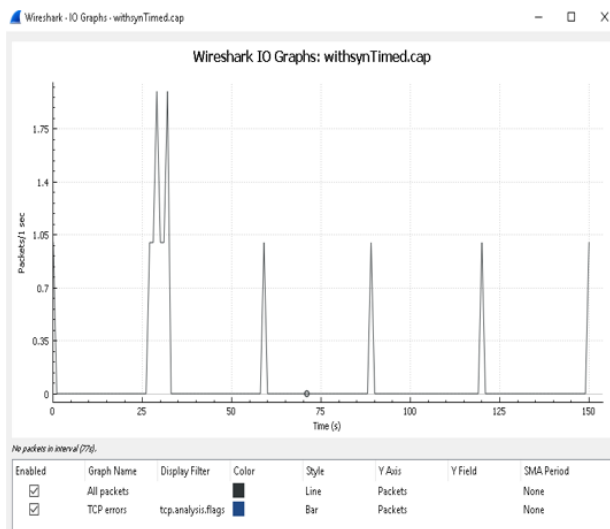
3.1 We used the same code as in the first part, implemented in nano editor.

```
root@jorgeLap: /mnt/c/Users/jorge
jjc400bh@server:/$ sudo sysctl net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 1
jjc400bh@server:/$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
jjc400bh@server:/$ sudo sysctl net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 0
jjc400bh@server:/$
```

3.2 Command for turning off SYN cookies

3.3 No need to mask IP address.

3.4 Using a watch and having command prompts connected to the server, client and attacker conducted transmission and attack.



After using tcpdump we used wireshark to provide IO graphs to compare with and without syncookies

nosynTimed.cap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|------------|-------------------|----------------|----------|--------|----------------------------------|
| 1 | 0.000000 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd Sy |
| 2 | 13.697815 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 24433 → 36622 Len=0 |
| 3 | 14.697784 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 31226 → 22007 Len=0 |
| 4 | 15.697804 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 49165 → 31759 Len=0 |
| 5 | 16.697773 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 57130 → 28386 Len=0 |
| 6 | 17.697792 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 39838 → 45262 Len=0 |
| 7 | 18.697813 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 18984 → 29827 Len=0 |
| 8 | 18.700457 | Intel_bb... | Intel_bb... | ARP | 60 | Who has 1.1.2.2? Tell 1.1.2.4 |
| 9 | 29.999785 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd Sy |
| 10 | 59.999624 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd Sy |
| 11 | 89.999462 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd Sy |
| 12 | 119.999348 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd Sy |
| 13 | 149.999036 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd Sy |

> Frame 1: 246 bytes on wire (1968 bits), 246 bytes captured (1968 bits) on interface 0

> Ethernet II, Src: HewlettP_32:b9:4a (2c:76:8a:32:b9:4a), Dst: LLDP_Multicast (01:00:00:00:00:00)

> Link Layer Discovery Protocol

```

0000  01 80 c2 00 00 0e 2c 76 8a 32 b9 4a 88 cc 02 07 .....v-2-J...
0010  04 2c 76 8a 32 a9 00 04 04 07 31 38 32 06 02 00 .,v-2...-182...
0020  78 08 03 48 31 34 0a 05 42 68 70 6d 64 0c 93 48 x-H14...Bhpmd-H
0030  50 20 4a 38 36 39 38 41 20 53 77 69 74 63 68 20 P J8698A Switch
0040  35 34 31 32 7a 6c 2c 20 72 65 76 69 73 69 6f 6e 5412zl, revision
0050  20 4b 2e 31 36 2e 30 32 2e 30 30 32 31 2c 20 52 K.16.02..0021, R
0060  4f 4d 20 4b 2e 31 35 2e 33 30 20 28 2f 77 73 2f QN K.15. 30 (/ws/
0070  73 77 62 75 69 6c 64 6d 2f 72 65 6c 5f 73 70 6f swbuildm /rel_spo
0080  6b 61 6e 65 5f 71 61 6f 66 66 2f 63 6f 64 65 2f kane_qao ff/code/
0090  62 75 69 6c 64 2f 62 74 6d 28 73 77 62 75 69 6c build/bt m(swbuil
00a0  64 6d 5f 72 65 6c 5f 73 70 6f 6b 61 6e 65 5f 71 dm_rel_s pokane.q
00b0  61 6f 66 66 5f 72 65 6c 5f 73 70 6f 6b 61 6e 65 aoff_rel _spokane
00c0  29 29 0e 04 00 14 00 04 10 0e 07 06 2c 76 8a 32 )).....,v-2
00d0  a9 00 02 00 00 00 00 00 fe 06 00 80 c2 01 02 19 .....
00e0  fe 09 00 12 0f 01 03 00 01 00 1e fe 07 00 12 0f .....
00f0  02 07 01 01 00 00 .....

```

withsynTimed.cap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|------------|-------------------|----------------|----------|--------|-------------------------------|
| 1 | 0.000000 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd |
| 2 | 27.168001 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 23972 → 17617 Len=0 |
| 3 | 28.167969 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 23632 → 17277 Len=0 |
| 4 | 29.167938 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 52524 → 57988 Len=0 |
| 5 | 29.999733 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd |
| 6 | 30.167959 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 52964 → 6847 Len=0 |
| 7 | 31.167926 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 54602 → 59989 Len=0 |
| 8 | 32.167946 | 192.168.1.1 | 5.6.7.8 | UDP | 60 | 17896 → wscopy(3378) Len=0 |
| 9 | 32.172539 | Intel_bb... | Intel_bb... | ARP | 60 | Who has 1.1.2.2? Tell 1.1.2.4 |
| 10 | 59.999669 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd |
| 11 | 89.999355 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd |
| 12 | 120.049160 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd |
| 13 | 150.048948 | HewlettP_32:b9:4a | LLDP_Multicast | LLDP | 246 | TTL = 120 System Name = Bhpmd |

> Frame 12: 246 bytes on wire (1968 bits), 246 bytes captured (1968 bits) on interface 0

> Ethernet II, Src: HewlettP_32:b9:4a (2c:76:8a:32:b9:4a), Dst: LLDP_Multicast (01:00:00:00:00:00)

> Link Layer Discovery Protocol

```

0000  01 80 c2 00 00 0e 2c 76 8a 32 b9 4a 88 cc 02 07 .....v-2-J...
0010  04 2c 76 8a 32 a9 00 04 04 07 31 38 32 06 02 00 .,v-2...-182...
0020  78 08 03 48 31 34 0a 05 42 68 70 6d 64 0c 93 48 x-H14...Bhpmd-H
0030  50 20 4a 38 36 39 38 41 20 53 77 69 74 63 68 20 P J8698A Switch
0040  35 34 31 32 7a 6c 2c 20 72 65 76 69 73 69 6f 6e 5412zl, revision
0050  20 4b 2e 31 36 2e 30 32 2e 30 30 32 31 2c 20 52 K.16.02..0021, R
0060  4f 4d 20 4b 2e 31 35 2e 33 30 20 28 2f 77 73 2f QN K.15. 30 (/ws/
0070  73 77 62 75 69 6c 64 6d 2f 72 65 6c 5f 73 70 6f swbuildm /rel_spo
0080  6b 61 6e 65 5f 71 61 6f 66 66 2f 63 6f 64 65 2f kane_qao ff/code/
0090  62 75 69 6c 64 2f 62 74 6d 28 73 77 62 75 69 6c build/bt m(swbuil
00a0  64 6d 5f 72 65 6c 5f 73 70 6f 6b 61 6e 65 5f 71 dm_rel_s pokane.q
00b0  61 6f 66 66 5f 72 65 6c 5f 73 70 6f 6b 61 6e 65 aoff_rel _spokane
00c0  29 29 0e 04 00 14 00 04 10 0e 07 06 2c 76 8a 32 )).....,v-2
00d0  a9 00 02 00 00 00 00 00 fe 06 00 80 c2 01 02 19 .....
00e0  fe 09 00 12 0f 01 03 00 01 00 1e fe 07 00 12 0f .....
00f0  02 07 01 01 00 00 .....

```

4. Word Problem:

1. Explain How TCP SYN flood attack works

SYN flooding Attack is a form of DOS Attack. A client uses many False IP addresses to send synchronizing (SYN) packets to a targeted port of server . The false IP addresses act like multiple user to the server and server interprets packets multiply. Server sends Acknowledge (ACK) back for each IP but in reality all one person .Thus it consume/attacks server's resources.

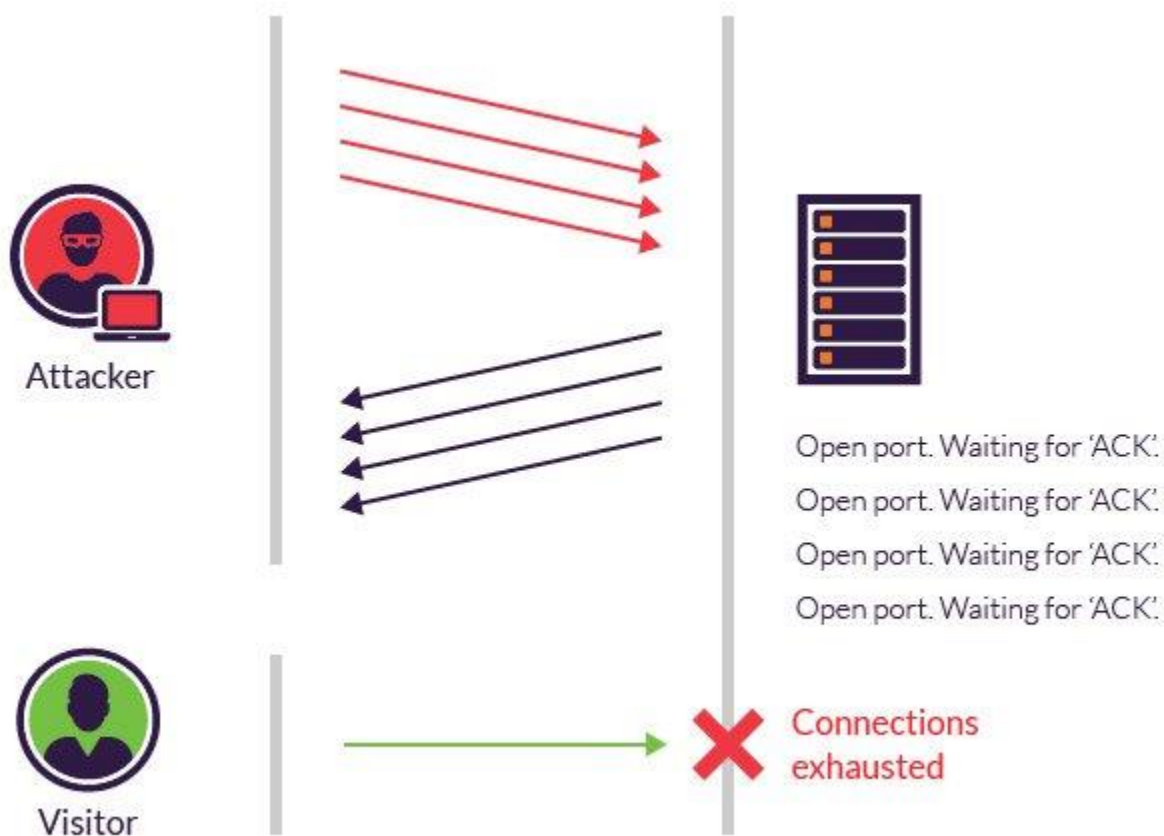
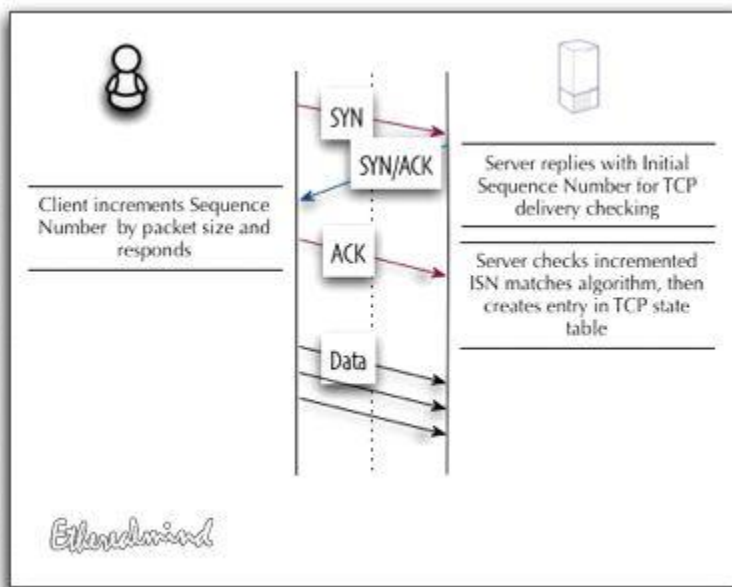


Photo source : <https://www.incapsula.com/ddos/attack-glossary/syn-flood.html>

2. Explain how SYN cookies work to prevent denial-of-service effect from SYN flood Attacks.

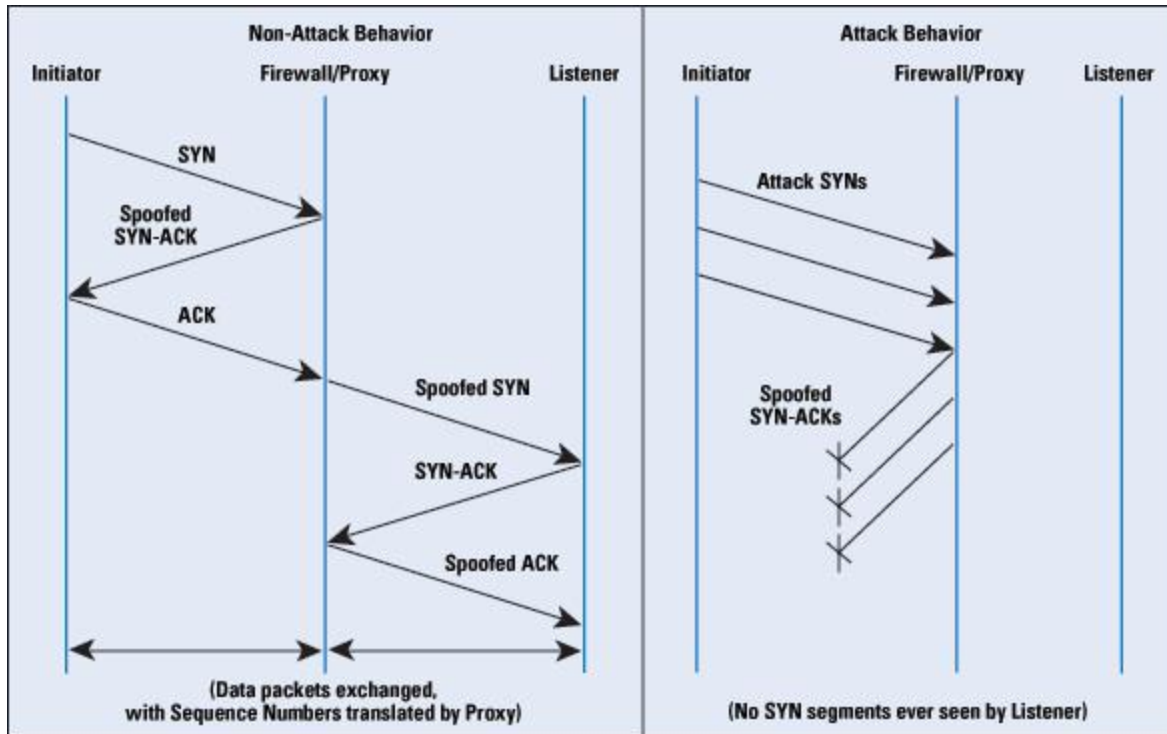
SYN cookies are a specific choice of initial TCP sequence numbers by the TCP software which prevent SYN flood attacks. When A client sends a SYN and the server replies with an SYN And ACK- response, Then the server will hold the informations in the TCP stack while it waits for the client's acknowledgement ACK-message. The SYN flood will then operate the SYN packets which, in return, will consume all TCP memory. Since the state table has limitation of memory , the server can not store any new TCP network connections which cause it will be a failed , denial or even worse response to the attacker.



<https://etherealmind.com/tcp-syn-cookies-ddos-defence/>

3. Would changing the network buffers in the OS (e.g. as available FreeBSD) have any impact on attack effectiveness?

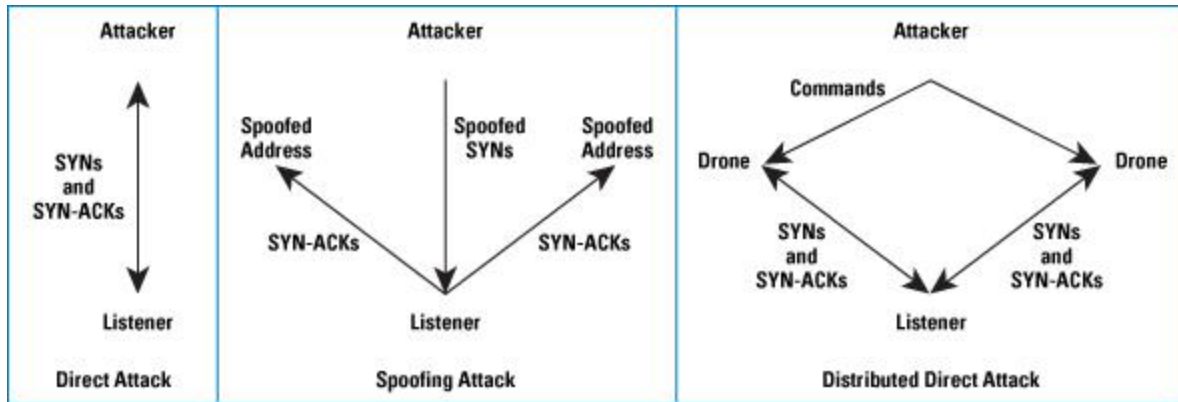
Yes, By changing the network buffer in OS (e.g. as available FreeBSD) could build an effective attack creating additional loopholes that a client get it hardship to crack. Exploiting the behavior of the buffer overflow become secure the OS.



https://www.cisco.com/c/dam/en_us/about/ac123/ac147/images/ipj/ipj_9-4/94_syn_fig6_lg.jpg

4. How would you defend against these attacks other than with SYN cookies?

- Hardening tcp/IP stack against SYN floods
- Changing the network buffer in OS as question 3.
- Implanting proper firewall policies are certainly first line of defense
- however the Linux kernel can also be hardened against these types of attacks.
- A spoofing attack when executed confuses the client/attacker with false or “spoofed” IP addresses that resemble the legitimate address. It is often used to overload network and initial devices to appear as the real deal, when in fact, it is a clone or similar to the real thing but not quite it.



<https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-34/syn-flooding-attacks.html>