SYN flood exercise

Offensive Technologies 2021
Matteo Franzil <matteo.franzil@studenti.unitn.it>

October 16, 2021

1 Solution

This instruction file solely contains the information needed for replicating the experiment. In order to see the answers to the required questions, please see the `memo.pdf` file.

1.1 Setting up

In order to get started, i first opened a four-panel terminal session, one for each node in the experiment. This made easier the data collection later on.

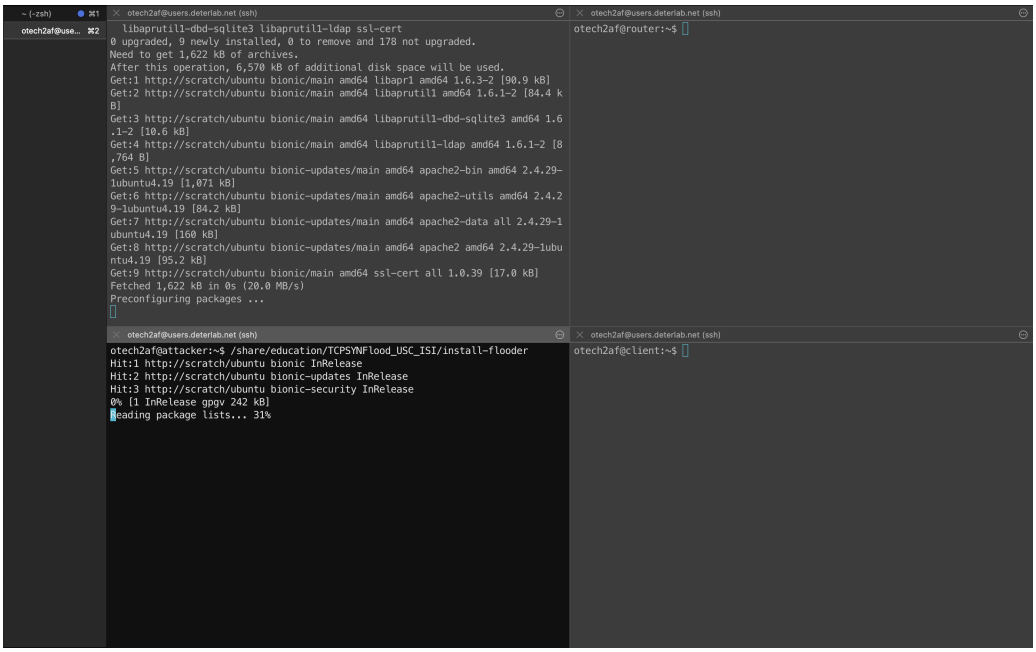


Figure 1 Setting up our environment.

For each of the nodes - bar the `router`, which I seldom used and only for quick `tcpdump` visits - I ran the following scripts, in order to set them up and later start the SYN flooding (or regular requests, for the client). For the client, I chose to keep `tcpdump` on the background in order to have focus on the script making requests, which could be then easily terminated with `^C`.

I decided to settle on `1000` packets per second for the DoS attack, a good compromise between crashing the webserver and having statistically insignificant results.

```
cat <<EOF > script.sh
#!/bin/bash
while sleep 1; do curl server/index.html >/dev/null 2>/dev/null; done;
EOF

# sudo tcpdump -nn -i eth1
```

```
chmod +x script.sh
sudo tcpdump -nn -v -s0 -i ${ETH} -w cookie.pcap &
./script.sh
```

Code 1 Code for the client

```
/share/education/TCPSYNFlood_USC_ISI/install-flooder

sudo flooder --dst 5.6.7.8 --src 1.1.2.0
--srcmask 255.255.255.0 --highrate 100
--lowrate 100 --proto 6 --dportmin 80
--dportmax 80
```

Code 2 Code for the attacker

```
SYNCOOKIES=$(sudo sysctl net.ipv4.tcp_syncookies | awk '{print $3}')
if [[ $SYNCOOKIES -eq 1 ]]; then
sudo sysctl -w net.ipv4.tcp_syncookies=0
sudo sysctl -w net.ipv4.tcp_max_syn_backlog=10000
fi
SYNCOOKIES=$(sudo sysctl net.ipv4.tcp_syncookies | awk '{print $3}')
if [[ $SYNCOOKIES -eq 0 ]]; then echo "OK"; else echo "KO"; fi

/share/education/TCPSYNFlood_USC_ISI/install-server

sudo tcpdump ip -i ${ETH}
```

Code 3 Code for the server

1.2 Gathering data

I gathered data for the following different setups:

- `REQ1`: syn-cookies: **on**, spoofing: **on**, network setup: **1**;
- `REQ2`: syn-cookies: **off**, spoofing: **on**, network setup: **1**;
- `EXT1`: syn-cookies: **off**, spoofing: **off**, network setup: **1** (extra point 1);
- `EXT2`: syn-cookies: **off**, spoofing: **off**, network setup: **2** (extra point 2);

Each setup was run with the following schedule (measured with a timer on my smartphone), for a total of 180 seconds each:

1. 30 seconds of client-only traffic;
2. 120 seconds of both attacker and client traffic;
3. 30 more seconds of client-only traffic.

This data was written in the `cookie.pcap` file, `scp`ied it to my PC, and opened with Wireshark. Then, it was further filtered and inspected (by removing runaway packets and observing the position of eventual ICMP error messages), and viewed with the Conversation view - which allows you to visualize the packets in a compact form, in which TCP sessions are condensed into a single line. Code 4 shows an example of a visualization provided by Wireshark:

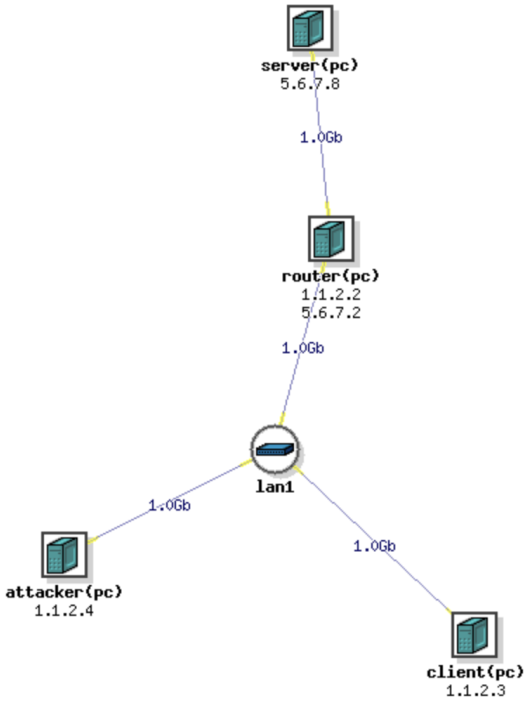


Figure 2 Network used for the first three setups.

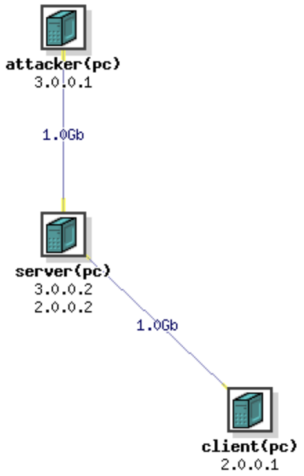


Figure 3 Network used for the last setup.

```
Address A----Port A----Address B---Port B----Packets---Bytes
1.1.2.3------58250-----5.6.7.8-----80--------14--------12193

Pk A>B---Bytes A>B---Pk B>A---Bytes B>A---Rel Start-----Duration
8--------616---------6--------11577-------1,031706------0,002221
```

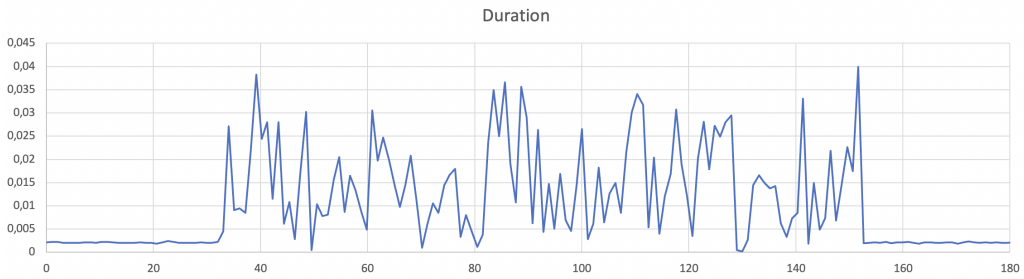
Code 4 Sample line (split in two) and key from the Wireshark conversation panel

For each of the setups, I exported the visualization for the whole 180 seconds of the setup to CSV and processed the data with Excel.

In the following tables, the *average duration* field refers to the total duration of the TCP session from the first `SYN` to the last packet (`RST/SYN`). As requested, timed out connections are assigned a duration of `200` seconds.

1.2.1 `REQ1` results

With this setup - syn-cookies on - we can see that the traffic kept flowing as expected, with connection duration only receiving a very small increase. This difference is amplified by the small scale of the graph, but still lies in the milliseconds and does not impact legitimate traffic at all.



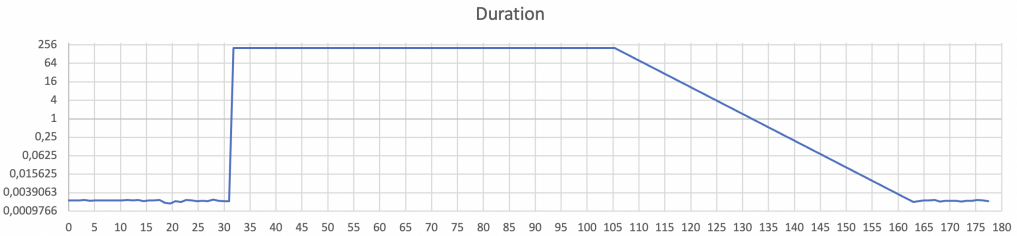
Traffic	Average duration (s)
regular traffic	`0.002105`
with DoS	`0,147`
overall	`0.01072`

1.2.2 `REQ2` results

In this setup - without syn-cookies - we can see that unlike in `REQ1`, during the DoS attack we have a complete loss of traffic. Indeed, during the 120-second window I recorded a single, clean packet originating from the client (which never received a reply), while I recorded `15` additional spoofed response packets, originating from the server. These packets were generated as a result of the spoofed attacker's request, and the client promptly replied with a `RST` - as the receiving port was not of course open.

I tried to tune down the amount of packets per second to 500, 300 and less, and I started having a rebound in traffic (i.e. some lucky session managing to close in 30 seconds or less) starting from 250 packets per second. Still, such numbers are irrealistic and too low for a proper DoS attack, even the weakest one.

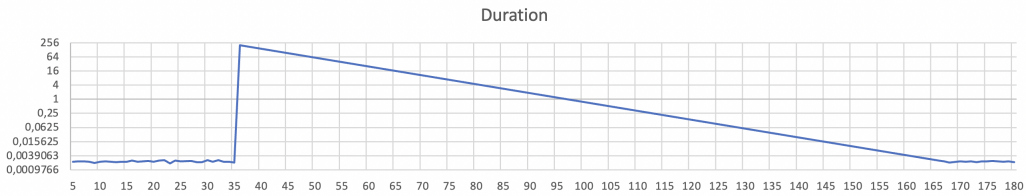
The graphic (and subsequent ones) has a logarithmic y-axis to account for the extreme difference between small fluctuations in the regular traffic (in milliseconds) and slowed down traffic (in seconds)



Traffic	Average duration (s)
regular traffic	`0.002138`
with DoS	`200`
overall	`19.60`

1.2.3 `EXT1` results

With this setup - syn-cookies off and spoofing off - we obtained similar result to `REQ2`, with the single difference that we obtained a single packet with a duration of 200 seconds. This happened since we deactivated spoofing, and so the attacker was no longer crafting packets whose response would be directed back at the client.

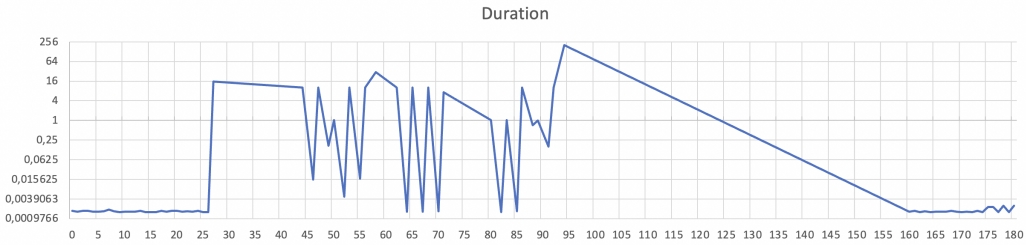


Traffic	Average duration (s)
regular traffic	`0.00218`
with DoS	`200`
overall	`5.08`

1.2.4 `EXT2` results

Finally, in this setup we drastically changed the layout of the network, getting rid of the router and instating point-to-point connections between the attacker and the server, and the client and the server. SYN cookies and spoofing were left off.

This had a drastic impact on the results. Firstly, the overall amount of traffic (and congestion) was reduced due to the segmentation of the routes (i.e. no overlap between client and attacker traffic, and no trash ARP traffic due to spoofing). Secondly, the removal of a hop halved the overall session duration, particularly during regular traffic windows. Thirdly and most importantly, even at 1000 packets per second some traffic was able to go through and come back unaffected, although slowly. This is reflected in the graph, showing an initial peak of unconcluded sessions and trickling down into a more acceptable - yet into the several seconds - level of duration.



Traffic	Average duration (s)
regular traffic	`0.00162`
with DoS	`6`
overall	`2.2087`