– Project 6 –
SYN Flood Mitigation
Advanced Network Security

Mohammad Umeer – #4748549

Task 1

To test and quantify the size of the transmission control block I developed a script in python & scapy able craft and send TCP SYN packet very rapidly.
All the packets have common destination the web server and the source are randomly generated.
With a normal traffic all the SYN are acknowledged (SYNACK) by the server, but with the SYN injection script the server stopped replying to the request almost after 10 sec and the acknowledge packets were in total approx. 128 (average of multiple runs).
Before launching the attack it is necessary to disable the SYN cookies and it is possible by setting to 0 (zero) the following file: */proc/sys/net/ipv4/tcp_syncookies*.
In the same folder is also present a file named *tcp_max_syn_backlog* which contain the size of the backlog (structure used to handle incoming packets with the SYN flag set until the moment the three-way handshake process is completed), after decreasing the value form the default 128 to a smaller number I attempted another attack and as expected it took less to DDOS it.

To Analyse after which time the half-open connection is released, i reduced the flow to a single SYS injection and I analysed packet transmission on a network monitor and this is what I saw.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.022332158	10.16.1.1	192.168.1.113	TCP	60	49153 → 80 [SYN] Seq=0 Win=8192 Len=0
4	0.022369773	192.168.1.113	10.16.1.1	TCP	58	80 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
5	1.048834955	192.168.1.113	10.16.1.1	TCP	58	[TCP Retransmission] 80 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=146
6	3.065151345	192.168.1.113	10.16.1.1	TCP	58	[TCP Retransmission] 80 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=146
9	7.159876254	192.168.1.113	10.16.1.1	TCP	58	[TCP Retransmission] 80 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=146
10	15.352636757	192.168.1.113	10.16.1.1	TCP	58	[TCP Retransmission] 80 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=146
30	31.480323428	192.168.1.113	10.16.1.1	TCP	58	[TCP Retransmission] 80 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=146

Figure 1: Retransmission of SYNACK reply to an SYN request

Form the figure is visible how the kernel retransmits the SYNACK reply to an SYN request. In other words, this tells the system how many times to try to establish a passive TCP connection that was started by another host. And every retransmission is sent in an exponential amount of time.

In my case, the half-connection is released after 31 second and 5 retries approximately.

Form the file located in */proc/sys/net/ipv4/tcp_synack_retries* it is possible to change the number of retries and hence to increase or reduce the half-open connection time.

How to test it:

- Set this file to zero */proc/sys/net/ipv4/tcp_syncookies*
- On the server side you have to run the script called runSniffer.sh
- On the attacker host you have to execute the runSender.sh script
- On the attacker host and on the server you will the result of the attack as number of SYS injection acknowledged.
- You may need to change the network device name or/and the IP addresses of the server/host. Also the performance of a computer may not be sufficient to dos the server, it is highly suggested to use multiple computer.
- At the end of the test the attacker can't be closed with CTRL+C but you need to terminate the console.

Because the redirection of the packets on the server side was not working (I tried **everything for 1.5 weeks before giving up, even the notes) so to acknowledge the SYN-ACK request I'm sending back a UDP packet to the attacker which is then analysed.*

Task 2

To build an IPS DDoS protection against SYN attacks I create a rule in the iptables of the web server which redirected all the TCP in INPUT on the port 80 to the NFQUEUE list, and I build a program in python to manage these packet with opportune roles.

```
sudo iptables -A INPUT -p tcp –-dport 80 -j NFQUEUE --queue-num 1
```

When a SYN packet, it is received the program is logging its IP source and the number of times that it appeared and if that number is less than the threshold It is simply accepted and so it and sent to the web server. But when the quantity is higher the whole packet information is saved in a list (listPacket) and continue with the next packet in the queue(If is the first time it sends a RST packet to the server).
In the program, there is a thread which checks every 0.5 seconds if there are packets that have not been yet sent to the server and for every single IP it accepts and sends one packet (the oldest one for a specific IP), after doing this it decrements the SYN appearance counter by one unit.
If the packet that is analysed form the queue is a ACK directed from the host to the server, the program cancels all the info (counter, IP log and packet) associated with that IP.

By mean of all this the IPS is able to allow to every IP a rapid SYN packet transmission until a THRESHOLD and after that, the transmission is slowed down to 1 packet per IP per 0.5 seconds. After a "cooldown" time defined by the THRESHOLD value that IP will be able to operate again at full speed (until it hit the limit again). IP addresses that send the ACK (so complete the 3-way handshake) are never slowed down.

To test the IPS I built a DDoS attack script that sends a burst SYN connection request (without never closing the handshake) to the web server. And after running it for a couple of seconds I tried to access a website hosted on the server and it worked every time with no substantial additional delays. The page request builds up a complete TCP handshake with the web server hence is never taken into consideration by the IPS.

However, this solution is far from perfection. Because as any resource-limited defence it can be broken with a resource depletion attack, by sending thousands of different IPs connection request per second it is possible to overload the table and eventually to crash the IPS, although it will be

more resilient because can be build the IPS with custom hardware specialized for this task. Also, by using a spoofed IP address, it is possible to slow down the access to the server to the original owner just by injecting enough SYN to keep the IP over the threshold.

How to test it:

- On the server side run the runIPS.sh, this script will install all the library the set the iptables rule before running the actual program.
- On the attacker side run the runAttackSimulation.py script to simulate an attack
- On a third machine you can try to build up TCP connection by requesting the webpage.
- You may need to change the server IP address with your server.

Task 3

This mitigation technique is very effective against web bots that have a basic logic but a complex system will be able to overcome this protection very easily. This solution is a good way to protect the server against a DDoS attack but is not perfect, by implementing it the sensible resources is now changed to the IPS which is a finite resource machine just like the server, a possible attack to it will have definitely small consequence and not the same effect if done directly to the server.

This solution will create a lot of false positive caused by IoT devices that want to legitimately connect to the server to upload/download data using API, which doesn't have a full HTTP stack and hence not able to follow redirection or run JavaScript.

Task 4

For this task, I used the same iptables role as for the TASK 2 and I developed a python application to manage the packets.
The application is waiting for a GET request to happen and when this occurs for the first time for an IP it is saving the IP and the destination requested and send back a redirect request to on fly crafted destination.
The destination is requested is built by a hash of the IP address and a HASH_SEED (randomly generated at the boot). So, the page requested has an URL similar to this host/?354239058032948 and it is different per for each IP.
If the client now replies with a GET request to that destination the program moves the IP to the whitelist and send back a redirection to the originally requested destination, which this time will be accepted because the IP is now trusted.

I tested the IPS using web browser generated connection and it is working every time, similar happen with WGET which is able to overcome the protection, but the simpler and “dumb” CURL is get blocked every time. The SYN flood attack test is able to effectively DDoS the countermeasure because this mitigation works on a higher layer.

How to test it:

- On the server side run the run.sh, this script will install all the library the set the iptables rule before running the actual program.
- Now you can test to access the server using a browser, curl or the use runAttackSimulation.sh to see how the code work upon receiving a SYN and use the runSender.sh of the Task1 to simulate a SYN attack)
- If you look at the terminal running the script you will see the redirection requested to the client

- When an IP is placed inside the whitelist is always trusted, if a device fails the redirection test the IP is “blacklisted” until it sends a GET with the test address.

Task 5

In the SYN+ACK crafting, the creation of the cookie number by mean of hash (and a successive normalization) of the following components to make it efficient and resilient.

- TCP socket information: In the hash function are included the IP address and port of both client and server, and this information is given to us by the client with the previous SYN packet. I included this component because are unique per each client, so it is possible to have multiple concurrent pending connections.
- Timestamp: A coarse version of the timestamp is also being used, it is coarse because during the ACK validation the cookie (+1) is compared with a timestamp (+-1 minute) this to allow flexibility for the connection but at the same time a small vulnerability windows.
- A random secret: The last component is generated one time at boot and it is composed of a random array of 16 bytes, this component allows us to have a differentiation factor for each machine running this software, and also an incrementation of the final entropy.

By mean of this cookie, a client can be easily verified without any drop of legitimate connection attempts.

Before building the IPS software I inserted the two following roles in the iptables. These roles allow the IPS to intercept the ingoing and outgoing TCP packet on the server.

```
sudo iptables -A INPUT -p tcp —-dport 80 -j NFQUEUE --queue-num 1

sudo iptables -A OUTPUT -p tcp --sport 80 -j NFQUEUE --queue-num 1
```

How to test it:

- On the server side run the run.sh, this script will install all the library the set the iptables rule before running the actual program.
- Now you can test to access the server using a browser for example and you will see on the standard output the TCP cookie creation and test.
- You may need to modify the SERVER_IP, INTERFACE_NAME or SERVER_PORT in order to run the IPS on you machine.

**When an IP is placed inside the whitelist is always trusted.*