Lorenzo Cavada Mat. 220502

Fourth exercise of OffTech about SYN Flooding. More information about the exercise can be found at this link. The code used in this exercise can be found here.

The attack.

The SYN Flood is an example of DoS attack. Basically in this attack, a malicious user bombards a server with connection requests without replying to the acknowledgements. The enormous number of open TCP connections that arise drain the server's resources, effectively crowding out legitimate traffic and making it difficult or impossible for authorized users to connect. This attack can occur in three main ways:

1. Direct attack:

Here the attacker uses its IP to send a huge amount of SYN request discarding the incoming SYN+ACK reply in order to achieve a "half-open" connection which consumes the resource of the webserver. By using a single source device with a real IP address to create the attack, the attacker is highly vulnerable to discovery and mitigation indeed is just needed to discard the packets coming from the malicious IP to stop the attack.

2. Spoofed Attack:

Here the attacker spoof the IP address on each SYN packet s/he send in order to inhibit mitigation efforts and make his/her identity more difficult to discover. Is important for the attacker to choose IP which does not reply in order to maximize the number of "half-open" connection on the server. Mitigating this type of attack by simply setting some firewall rules is hard and may require the help of the ISP.

3. Distributed Attack:

Here the attacker use a botnet to decrease the likelihood of tracking the attack back to its source. For an added level of obfuscation, an attacker may have each distributed device also spoof the IP addresses from which it sends packets.

The fix.

The main problem is related to the fact that the server allocates the resources needed for a connection as soon as it receives an SYN request. Due to the fact that the attacker is not answering to the SYN+ACK, the server will keep these resources allocated for some time finishing for end them becoming unable to serve other legitimate requests. An easy fix to this problem is by enabling **TCP cookies**. Basically, the idea behind this cookie is to craft a particular sequence number (usually randomly chosen) that encode some information about the connection. In this way the server will allocate the resource only after receiving the ACK message from the client, meaning that both the server and the client have completed the three-way handshake and are ready to start a communication. This fix works especially well due to the fact that does not require any change in the TCP protocol making it completely transparent for the client. A drawback coming from the use of TCP cookies is the loss of some TCP options which can not be encoded into the sequence number, some workaround is anyway still possible by using some encoding and the timestamp options. Is also important to notice that if the attacker has more bandwidth than the server an SYN flooding attack is still possible.

The client script.

The client script, called `traffic-gen.sh` is very simple, is indeed a `while true` in which, every second, is performed a `curl` request directed to the server (which IP is passed as a parameter) throw the default port 80. Example of use:

```
bash traffic-gen.sh 5.6.7.8 # This will start sending request to the specified IP every second
```

The attacker script.

As for the client also the attacker script, called `syn-dos.sh`, is quite simple. It takes in input 4 parameters, the first one is the target of the attack, then the second is the number of packets that the script should send every second, then there is the IP and the Netmask that the attacker should use for spoofing the source IP of the packets. Example of use:

```
bash syn-dos.sh 5.6.7.8 300 1.1.2.4 255.255.255.0 #The attacker will start sending 300 packets every second using addresses going from 1.1.2.1 to 1.1.2.254
```

The simulation of the attack.

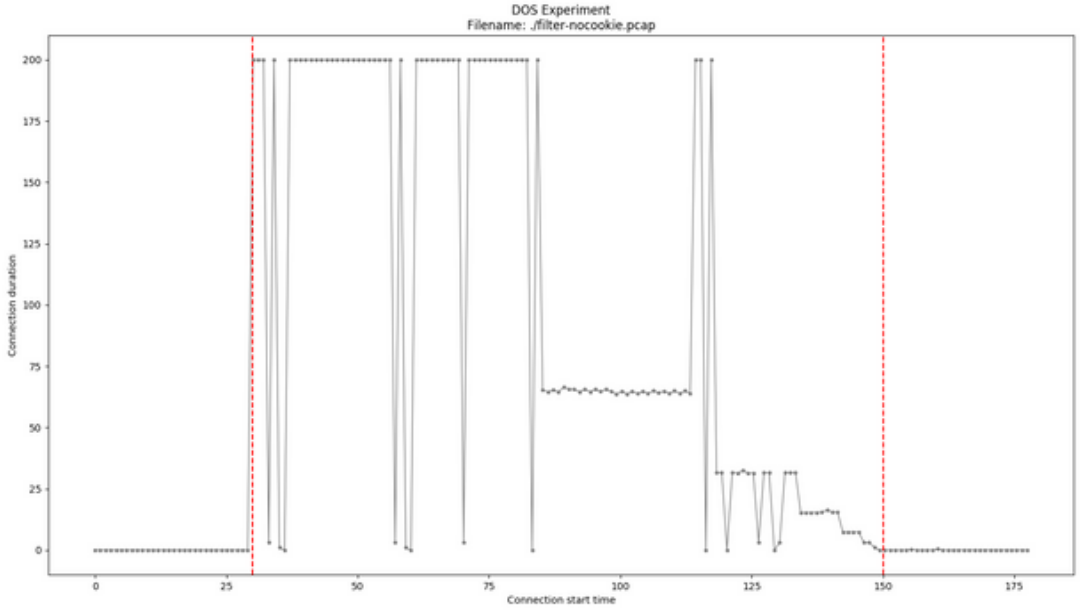
To replicate the attack first is needed to start the `setup.sh` script, this will download and install the necessary component and will disable the `TCP cookies` from the webserver. After that, the script is finished is possible to launch the `start_attack.sh` script which will first get the interface used from the client to connect to the server and will then perform the following action:

1. Start the `tcpdump` from the client listening for any tcp packets and saving the result of the filtering in a file called `filter.pcap`
2. Start the `traffic-gen.sh` script that will start generating new requests from the client to the webserver
3. Wait 30 seconds
4. Start the attacker script that will begin the SYN flooding toward the webserver
5. Wait 120 seconds
6. Stop the attacker script
7. Wait 30 seconds
8. Stop both the client script and the `tcpdump`

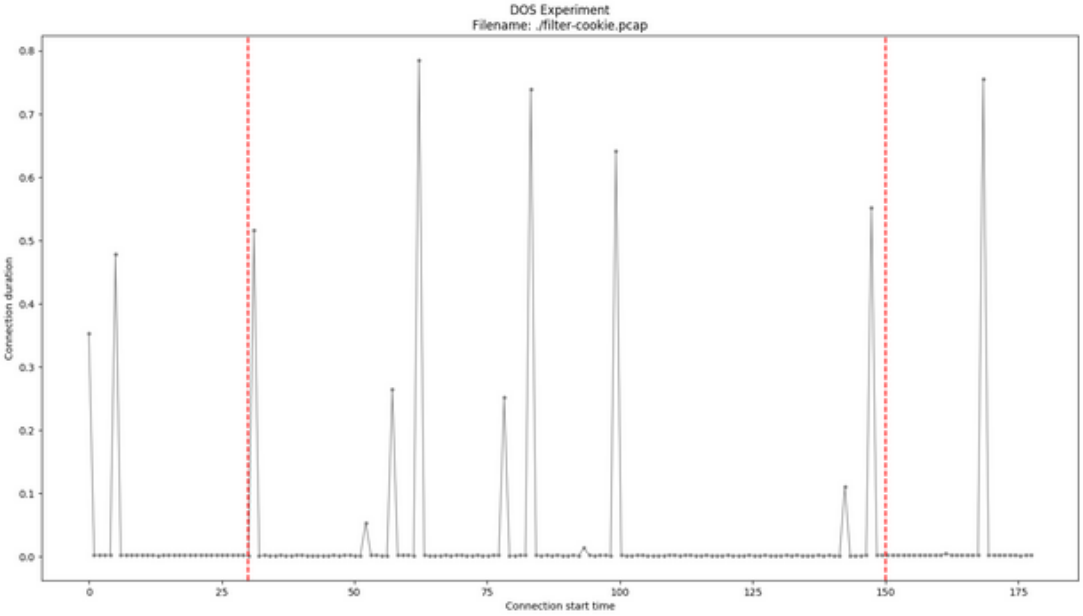
Is now possible to `scp` the result filtering from the Deterlab machine to a local one where can be checked. For parsing it can be used a python script called `wirefish.py` which require as argument the path to the `.pcap` file. What this script does is reading each packet and try to reconstruct the history of each connection. For understanding, if two packets belong to the same connection the `source port` of the client is used (Note that with big `.pcap` file this can lead to collision). Each connection is so reconstructed and is checked if it has been correctly closed (throw an RST or a four-way handshake), if this is the case the duration of the connection is calculated (time passed from the first message and the last) otherwise it is arbitrary set to 200s. This value is then used to plot a graph showing on the x-axis the time passed from the start of the experiment and the start of each connection while on the y-axis is shown the duration of each connection.

Simulation with and without TCP cookies.

Here is possible to see the result of the processing of a sniff obtained during a simulation of the attack with the TCP cookies disabled. The attacked started after 30 seconds and is possible to see that many connections did not conclude in a proper way due to the overload of the server which was not able to process all the incoming requests. As soon as the attack stops the client is again able to reach the server without any trouble.

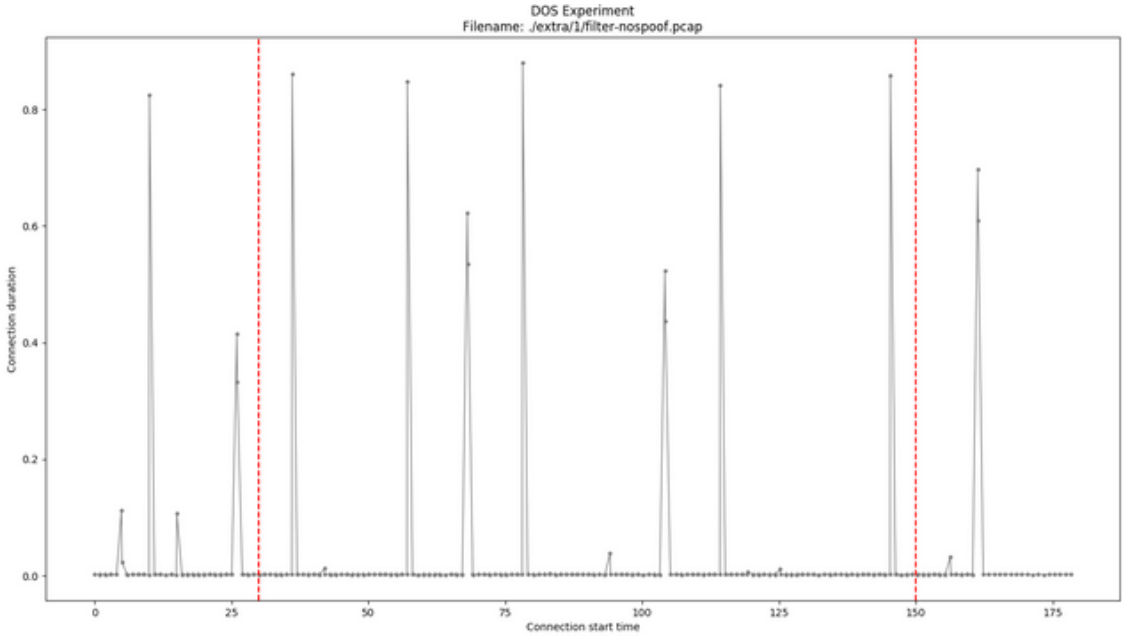


Here is instead possible to see the result of the processing of a sniff obtained during a simulation of the attack with the TCP cookies enabled. After that the attack starts is possible to notice some delay in some connection but in any case this does not affect the experience of the client (note that the longest connection duration is anyways under 1 second).



Extra 1: remove the spoofing

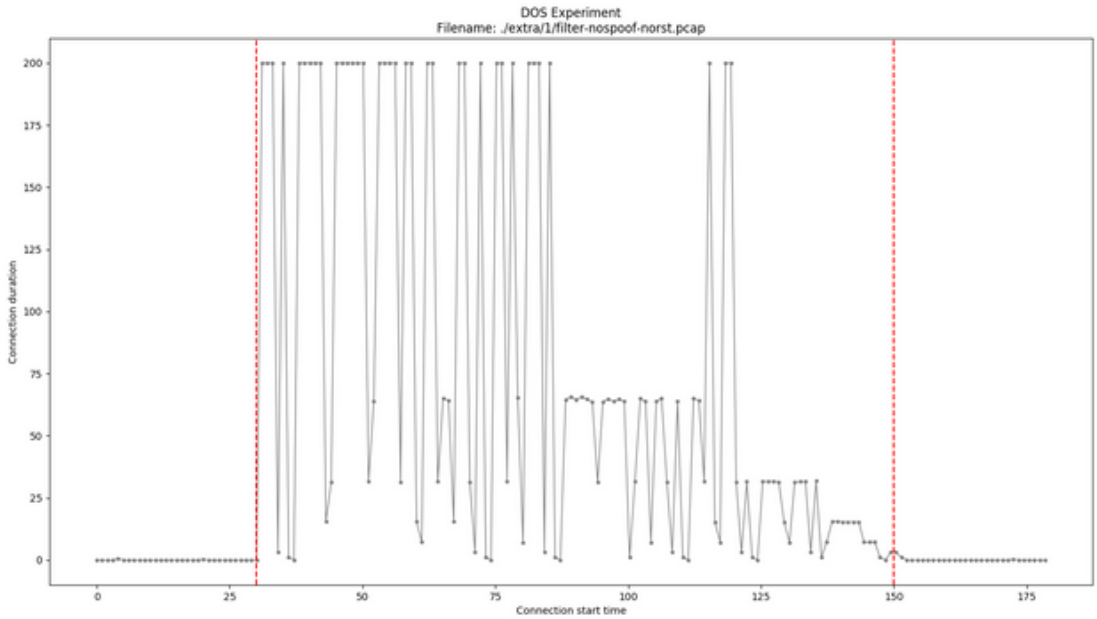
During the previous attack, the malicious user proceed to create some ad hoc requests having as source IP a random one chosen from `1.1.2.0/24`, in this way the server was allocating resources and sending responses to an unactive machine leading to the situation of "half-open" connection previously discussed. By removing the spoofing the attacker is now sending crafted packets having as source IP its own IP, `1.1.2.4`. When the attack starts the server will start receiving a huge amount of SYN requests from the attacker and will start answering with SYN+ACK messages. Due to the fact that the source IP is no longer spoofed this reply will come back to the attacker which will start receiving a lot of unexpected SYN+ACK replies. As from TCP specification the malicious machine will respond to this message with an RST packet which will reach the server closing the "half-open" connection and, so, freeing the previously allocated memory making the DoS attack ineffective. Here is instead possible to see the result of the processing of a sniff obtained during a simulation of the attack without the spoofing, is possible to see that, even without the TCP cookie enabled the attacker was not able to perform a DoS attack.



The DoS attack is in any case still possible by preventing the attacker to reply to the unexpected SYN+ACK messages with an RST. To do so is just needed to add a firewall rule that will drop all the packets directed to the server having the RST flag set to 1 (is also possible to drop the SYN+ACK packets coming from the server):

```
sudo iptables -I OUTPUT 1 -d 5.6.7.8 -p tcp --tcp-flags RST RST -j DROP
```

Here is instead possible to see the result of the processing of a sniff obtained during a simulation of the attack without the spoofing but with the RST packets dropped thanks to the new firewall rule. Is possible to see that the attacker is still able to perform a DoS attack.



Extra 2: change the topology

For answers to this question a change in the topology was needed, specifically was necessary to create 2 new links capable of connecting the server to the client and the server to the attacker throw a dedicated point-to-point connection. The new .ns file have this structure:

```
...
set link1 [$ns duplex-link $client $server 1000Mb 0ms DropTail]
set link2 [$ns duplex-link $attacker $server 1000Mb 0ms DropTail]

tb-set-ip-link $client $link1 1.1.1.2
tb-set-ip-link $server $link1 1.1.1.1

tb-set-ip-link $attacker $link2 2.2.2.2
tb-set-ip-link $server $link2 2.2.2.1
...
```

With this new topology the router was deleted and the attacker was no more able to reach the client and vice-versa, they are only able to talk with the server. Starting again the attack is interesting to see a dump of the traffic received from the server. By looking at the one intercepted in the interface connected with the attacker is possible to notice how the malicious packet sent by the attacker did not receive a proper SYN+ACK answer, this is due to the fact that the server does not know where to send those packets and proceed to drop them. Expecially, by looking into a dump of the connection made from e to the server is possible to see how, for each malicious packets, the server try to understand where to send the reponse throw an ARP request which will not get any response causing the drop of the connection by the server.