

# ITS Assignment 5

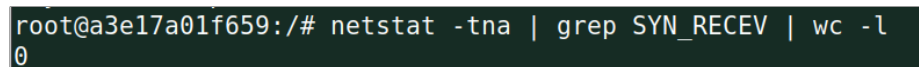
David hbd126, Rasmus dht579

15. October 2023

## Task 1: SYN flooding attack

In this task we are attacking a victim using a SYN flood attack, where the intention is opening a number of half-open connections, but never finishing the three-way-handshake. This can take up all available connections, thus denying new connections.

When a victim has no connections, the command in the following picture will return 0, or the number of current connections if any exist.



```
root@a3e17a01f659:/# netstat -tna | grep SYN_RECV | wc -l
0
```

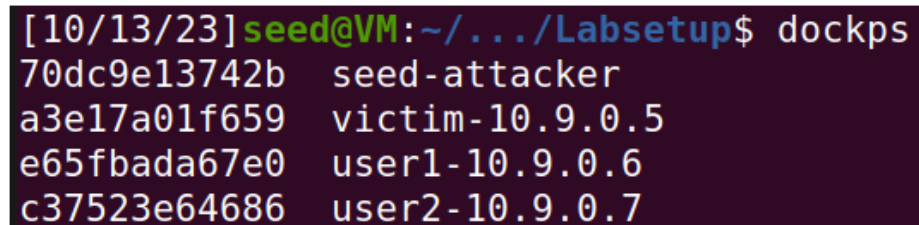
Figure 1: Example of host with no connections

As explained in the TCP\_Attacks.pdf from seedlabs, Ubuntu has protection against this kind of attack enabled by default, so to test this kind of attack, our docker setup has this turned off.

### 1.1: Launching the attack using python

We have been given a python file to execute our attack with, but it leaves out the port and ip which we are to supply.

After using **dcup** to open the hosts we can then use dockps in another container to see our available hosts.



```
[10/13/23]seed@VM:~/.../Labsetup$ dockps
70dc9e13742b  seed-attacker
a3e17a01f659  victim-10.9.0.5
e65fbada67e0  user1-10.9.0.6
c37523e64686  user2-10.9.0.7
```

Figure 2: Caption

We can then add our victims ip-address and standard port 23 to the synflood.py file.

As we saw from earlier, we had 0 connections before the attack. Running the SYN flood attack on the victim we then get a larger amount of connection slots filled.

```

root@903eae8dd480:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@903eae8dd480:/# █

```

Figure 5: Telnet connection timing out after the SYN attack

```

root@a3e17a01f659:/# ss -n state syn-recv sport = :23 | wc -l
98

```

Figure 3: Connections after SYN flood attack

We were told that we should expect our first attempt with the python implementation to fail, since the default number of available connections should be higher than what the python implementation would be able to maintain. In our testing we disproved this information, as our first attempt with the python implementation succeeded. In the following pictures we try to connect to the victim using a different host using telnet.

```

[10/10/23]seed@VM:~/.../Labsetup$ docksh 90
root@903eae8dd480:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS

```

Figure 4: Telnet connection success before the SYN attack

We thus see that the SYN attack blocks connections.

The assignment text goes into details for several measures we were able to take if the python implementation did not work, such as setting a cap on connections for the victim, but since we did not encounter these problems we will not go into details in regards to this. But it works as a segue into the next subsection.

## 0.1 1.2: Launching the attack using C

If the python implementation was unable to fill up the cap for connections on the host, we can instead use a given C implementation. This implementation should be several times faster as it's a more low-level language that is optimised

for performance. We are therefore able to send more SYN packets to fill up the connection limit. What this would mean in practice is that you can target networks more efficiently, since you can create more connections with the same amount of hardware.

What we see in practice is that because of these optimisations the C code was easily able to fill up the connection capacity for the host and effectively shut down the service. To test this we reset our connections:

```
root@a3e17a01f659:/# netstat -tna | grep SYN_RECV | wc -l
0
```

Figure 6: No connections on the host

Before we can launch our attack, we need to compile the program outside of a container, so the attacker host 10.9.0.1 can use it. This is done using the following command.

```
gcc -o synflood synflood.c
```

With the compiled program, the attacker can now run the program with the target IP and port, thus starting the attack:

```
synflood 10.9.0.5 23
```

If we now try to Telnet into our host 10.9.0.5 from another host, then we get the following:

```
seed@VM:~/.../volumes$ docksh 9a
root@9a1c72143c97:/# telnet 10.9.0.5 23
Trying 10.9.0.5...
^C
root@9a1c72143c97:/#
```

Figure 7: Unable to connect to host

What this means is that we have successfully overloaded the host and if we look at the connection in host 10.9.0.5 then we can see why:

```
net.ipv4.tcp_max_syn_backlog = 10000
root@fa34a5235e61:/# netstat -tna | grep SYN_
RECV | wc -l
128
```

Figure 8: Number of connections after C attack

### 1.3 Enable the SYN Cookie Countermeasure

A way to counter this kind of attack is to enable SYN cookies:

```
root@fa34a5235e61:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@fa34a5235e61:/# netstat -tna | grep SYN_
RECV | wc -l
0
root@fa34a5235e61:/# netstat -tna | grep SYN_
RECV | wc -l
128
```

Figure 9: Number of connections from the C attack after cookies have been enabled

This looks the exact same as the attack without cookies, but if we try to Telnet into our host 10.9.0.5 then we see the following:

```
root@9a1c72143c97:/# telnet 10.9.0.5 23
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
fa34a5235e61 login: █
```

Figure 10: Telnet into SYN cookie network

The reason why we are able to get into the network now despite there being just

as many connections is that When a server with SYN cookies enabled receives a SYN packet from a client. Then it does not allocate resources or store state information for that connection until the three-way handshake is successfully completed. What this means is that we are still free to create a connection to the server, since the three-way handshake with the attacker has not been completed yet.

## Short questions

### What is OS fingerprinting?

OS fingerprinting is a way to determine the operating system that is running on a remote device. The reason why this information is valuable is that each operating system has value both for attacking and defending a system. If you know that you only expect iOS devices to use your service then you can shut down and prevent connections using Linux for instance. This can be used by Apple to keep their systems safe and protected from hackers trying to use third party software to access device information.

Alternatively for an attacker then this can be used to target specific exploits. For instance if a zero day exploit is found in MacOS, then the attacker can specifically target MacOS devices to try and target with this specific exploit in mind.

The way you do OS fingerprinting can be banner grabbing, which is taking the information from a HTTP header for instance:

```
GET /tutorials/other/top-20-mysql-best-practices/ HTTP/1.1
Host: code.tutsplus.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1;
          en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5
          (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,
       application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

Here we see under User-agent that the user is using Mozilla firefox 5.0 and Windows NT 6.1. This information can then be used to carry out a targeted attack, for instance if you see a user is using an outdated version of Windows with a specific exploit, then you can use that exploit against the user to begin the attack.

HTTP is not the only way though, there are many more such as FTP, SSH and Telnet. You can also use Time to live analysis since different operating systems use this differently, which allows you to figure out which operating system you are dealing with.

## Detection of SQL injection attacks

As we were introduced to in the very first lecture in this course, security grows with more investment. This makes sense as we can cover more areas or have more layers of security.

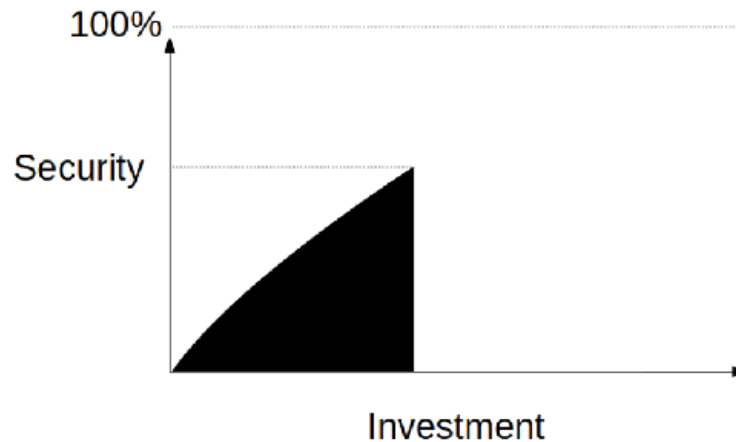


Figure 11: Picture from the ITS-2023-Intro PDF showing the growth of security with investment. With the context that 100% security is impossible.

Taking this into account, the correct choice given enough resources, would be to employ all three defensive measures, since a multilayered approach to security is better than a single layered one. The reason for choosing to use all three measures, is because they work well together.

A Network Intrusion Detection System (NIDS) analyses network traffic, and can therefore identify attacks before they even reach the web server. Since traffic at the network level is monitored, it can also detect attacks that may not be visible in the web server logs.

Given limited resources, a NIDS could be a good single layer solution, but this still only covers external intrusion and if an attack is taking place internally on the server, this would prove ineffective.

A Host Intrusion Detection System (HIDS) monitors the activities on the web server directly. This allows us to monitor file changes and other unusual behaviour which could be warning signs of an SQL injection attack.

Given the same example as with a NIDS, we could also use a HIDS as a single line of defense, but this leaves us wholly blind to the network layer.



If we were to only analyse the web server application logs, we would be entirely reactive instead of proactive in our defence. We would in a way be unlocking our doors and then only checking on things after they have happened. Which is a very bad way of going about security in our opinion.

So to reiterate our earlier conclusion, we believe that a combination of the three security measures would be more suited given the required amount of resources. This solution would allow us to monitor the network, web server and follow up and analyse the web server application logs afterwards to look for inconsistencies or indicators of wrong doings.