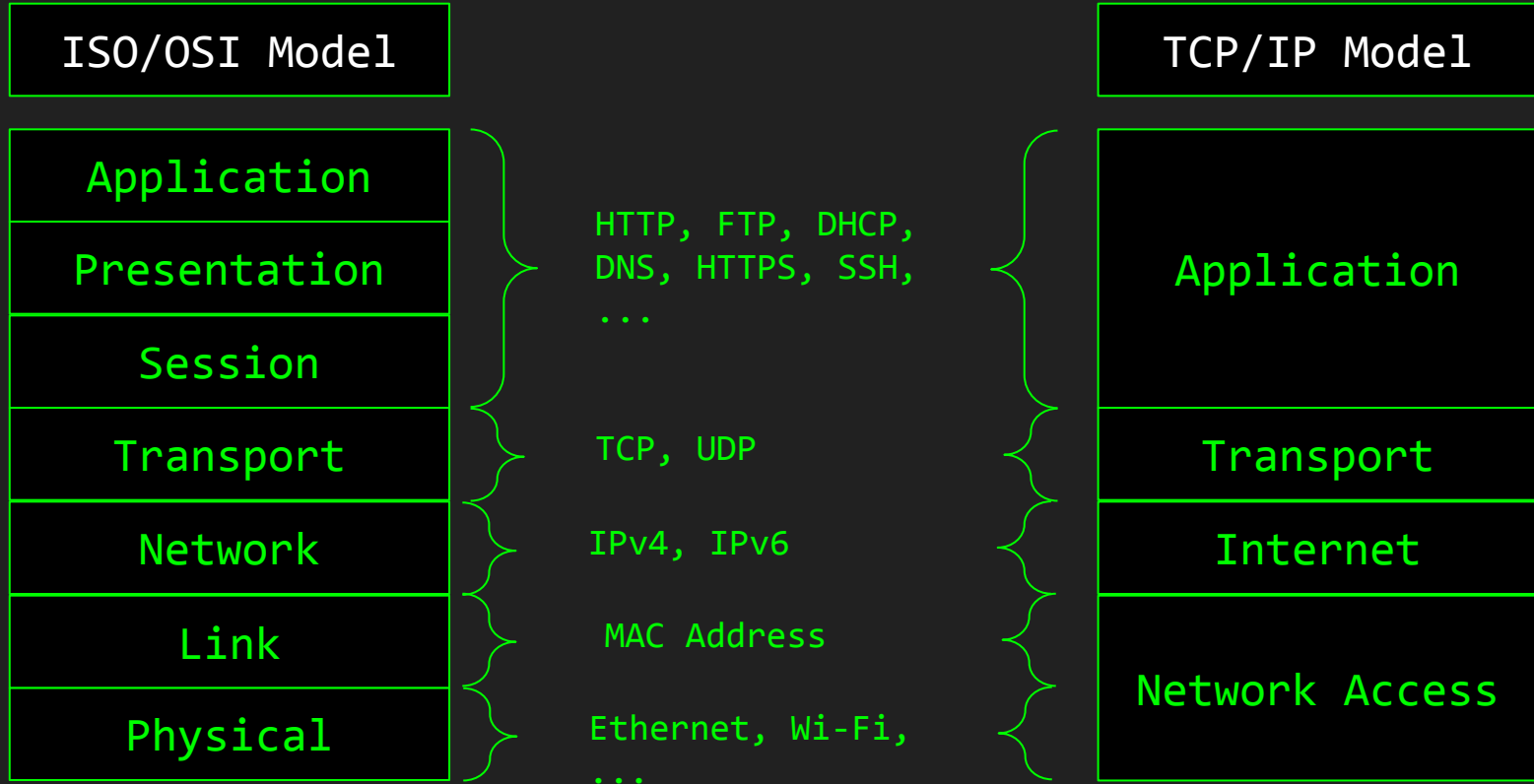


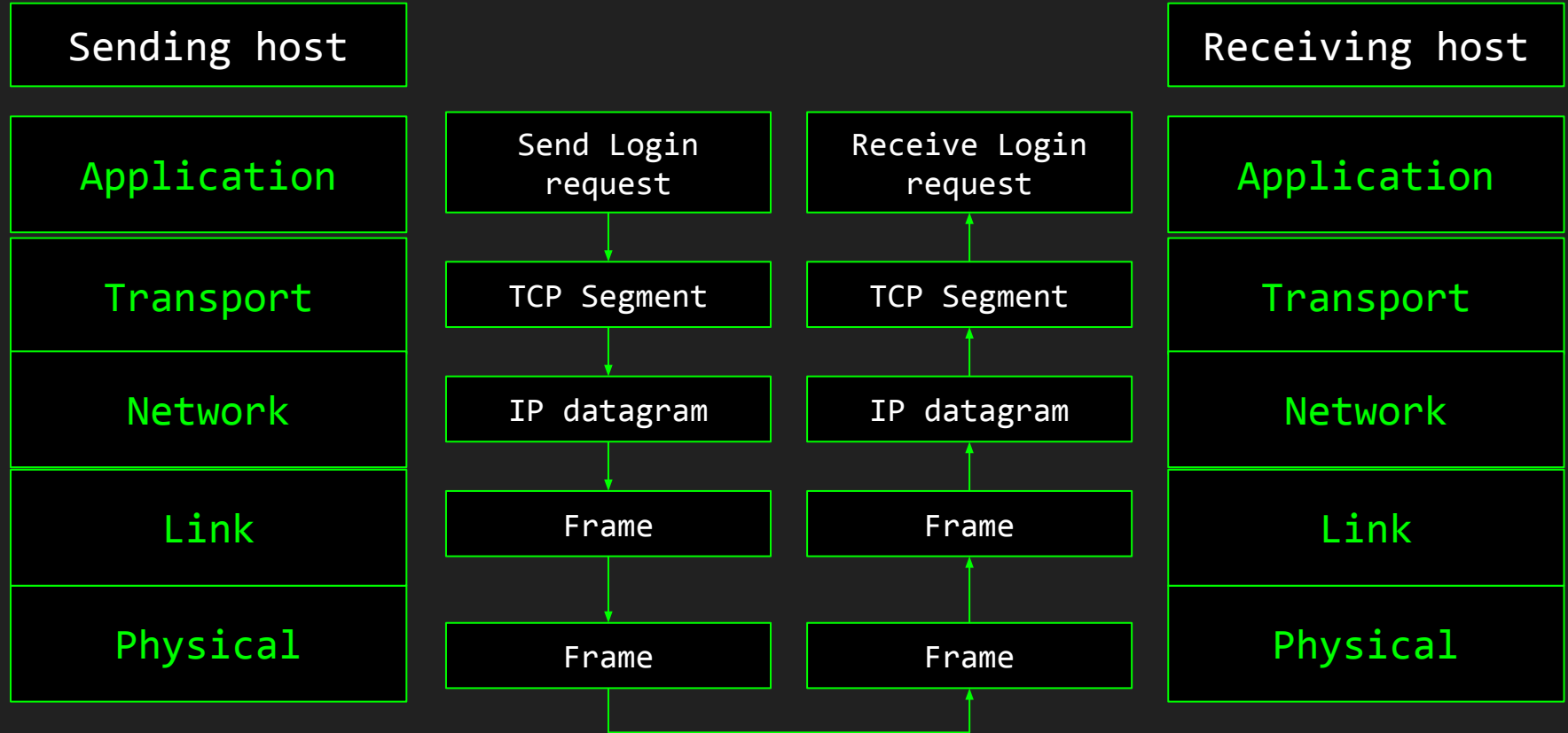
Network Analysis and Monitoring

Network Security

> Internet Stack recap



> Internet Stack recap



> Focus: Transport Layer

Transmission Control Protocol (TCP):

- Reliable, ordered and error-checked delivery
- Connection oriented

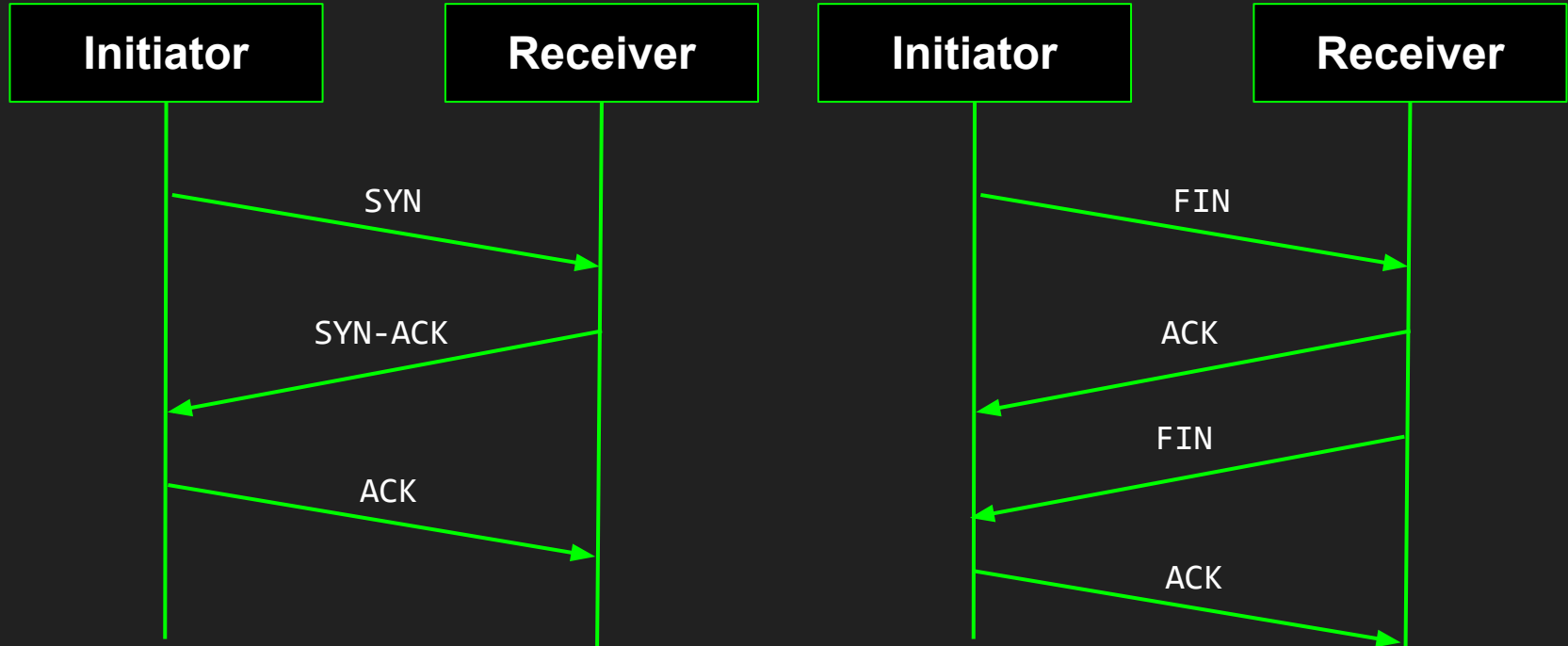
User Datagram Protocol (UDP):

- Best-Effort delivery
- Connection-less

> TCP Segment

Offset	Ottetto	0								1								2								3							
Ottetto	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (se ACK è impostato)																															
12	96	Data offset	Reserved 0 0 0 0				C	E	U	A	P	R	S	F	Window Size																		
						W	C	R	C	S	S	Y	I																				
						R	E	G	K	H	T	N	N																				
16	128	Checksum																Urgent pointer (se URG è impostato)															
20	160	Options (facoltativo)																															
20/24	160/192	Data																															
...	...																																

> Open and close a TCP connection



> Network Address Translation (NAT)

A method for remapping one IP into another:

131.114.10.12 → 192.168.1.10

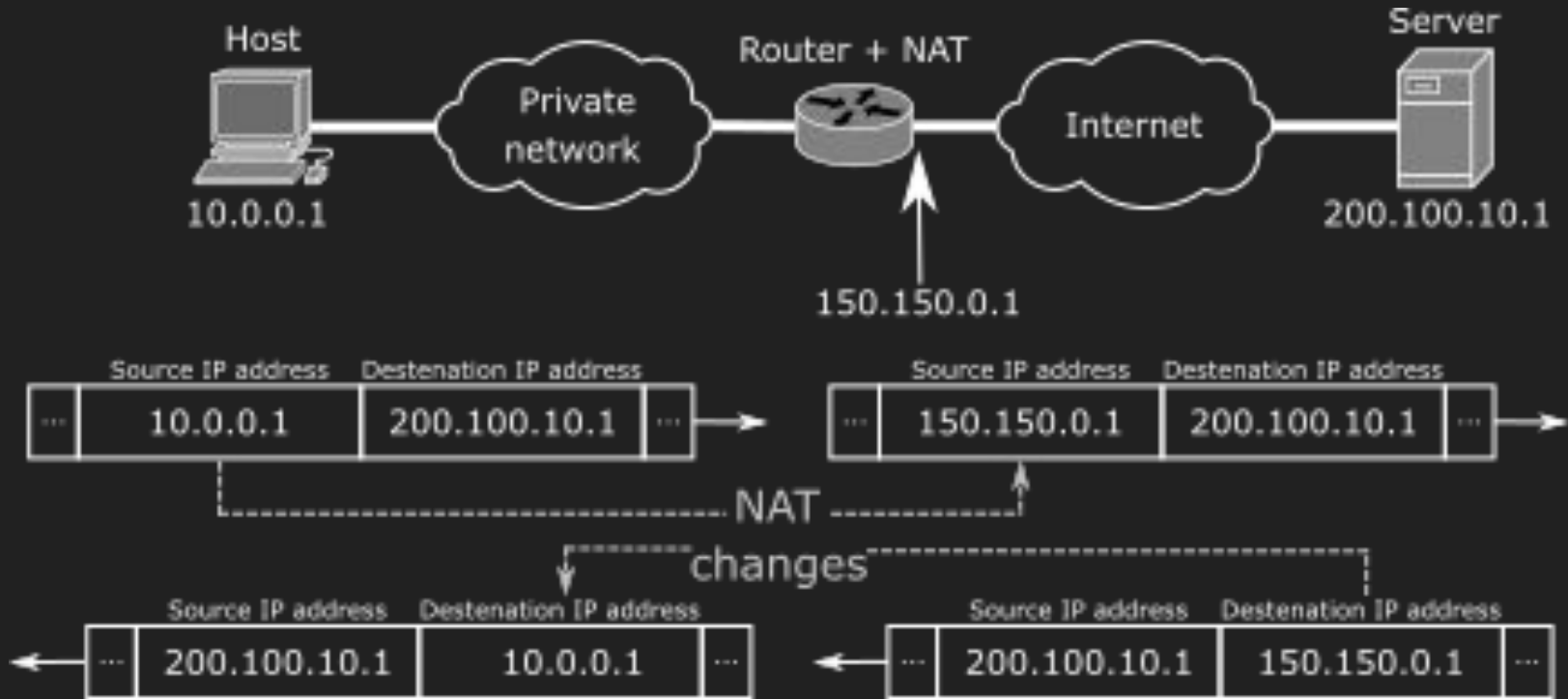
Main approaches:

- one-to-one
- One-to-many

Why?

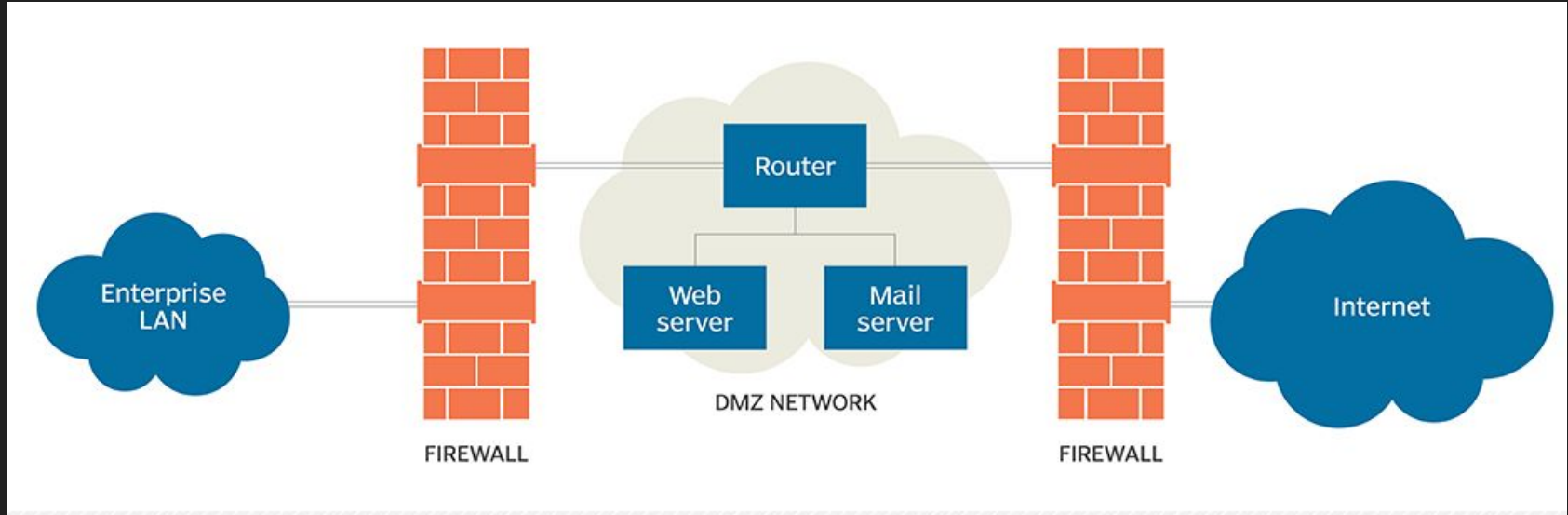
- IPv4 address availability: $256^4 \approx 4.3$ Billion
- (IPv6 availability: 655 zillion x Earth square meter)

> NAT concept

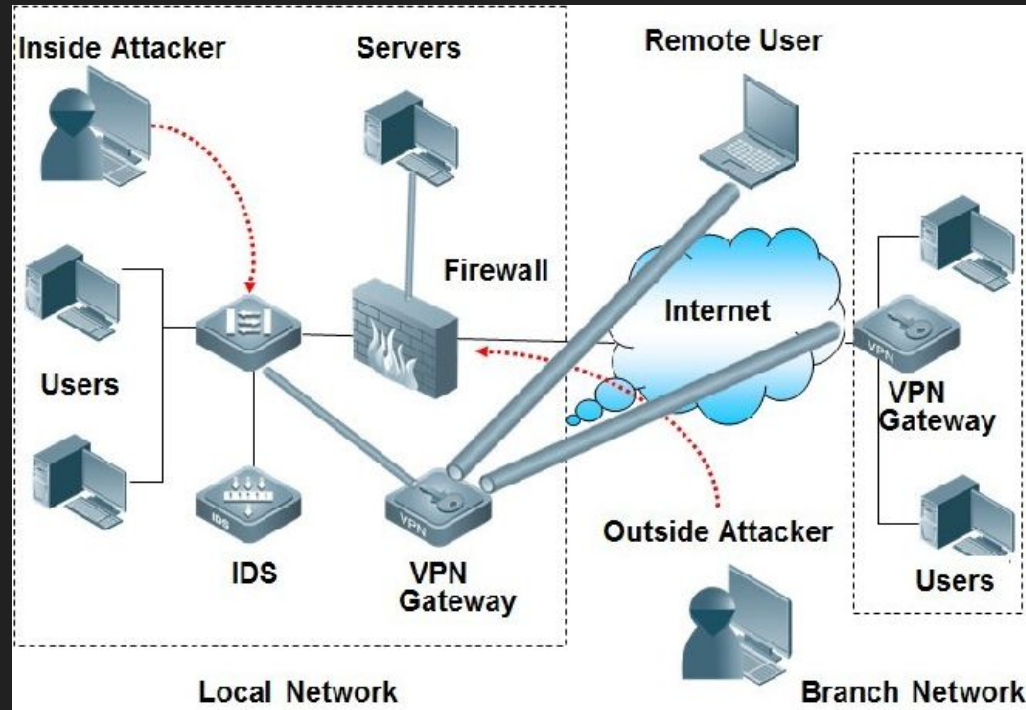


> Network Security Architecture: DMZ

DeMilitarized Zone: physical or logical subnet exposing services to an untrusted network



> Network Security Architecture



> Normal vs Promiscuous mode

- A NIC receives all the frame flowing on the physical mean;
- Each Frame has a destination MAC Address;

NORMAL MODE:

if the destination MAC address doesn't match → the frame is discarded

PROMISCUOUS MODE:

All the frame are accepted

> How to set promiscuous mode in Linux

Check network interfaces:

```
> sudo ifconfig
```

Set promiscuous mode:

```
> sudo ifconfig [interf] promisc
```

or

```
> sudo ip link set [interf] promisc on
```

Check that the interface is in promiscuous mode

```
> sudo ifconfig [interf]
```

or

```
> sudo ip show [interf] | grep -i promisc
```

> Wireshark

A tool for analyzing the internet traffic

Installation:

```
> sudo apt update
```

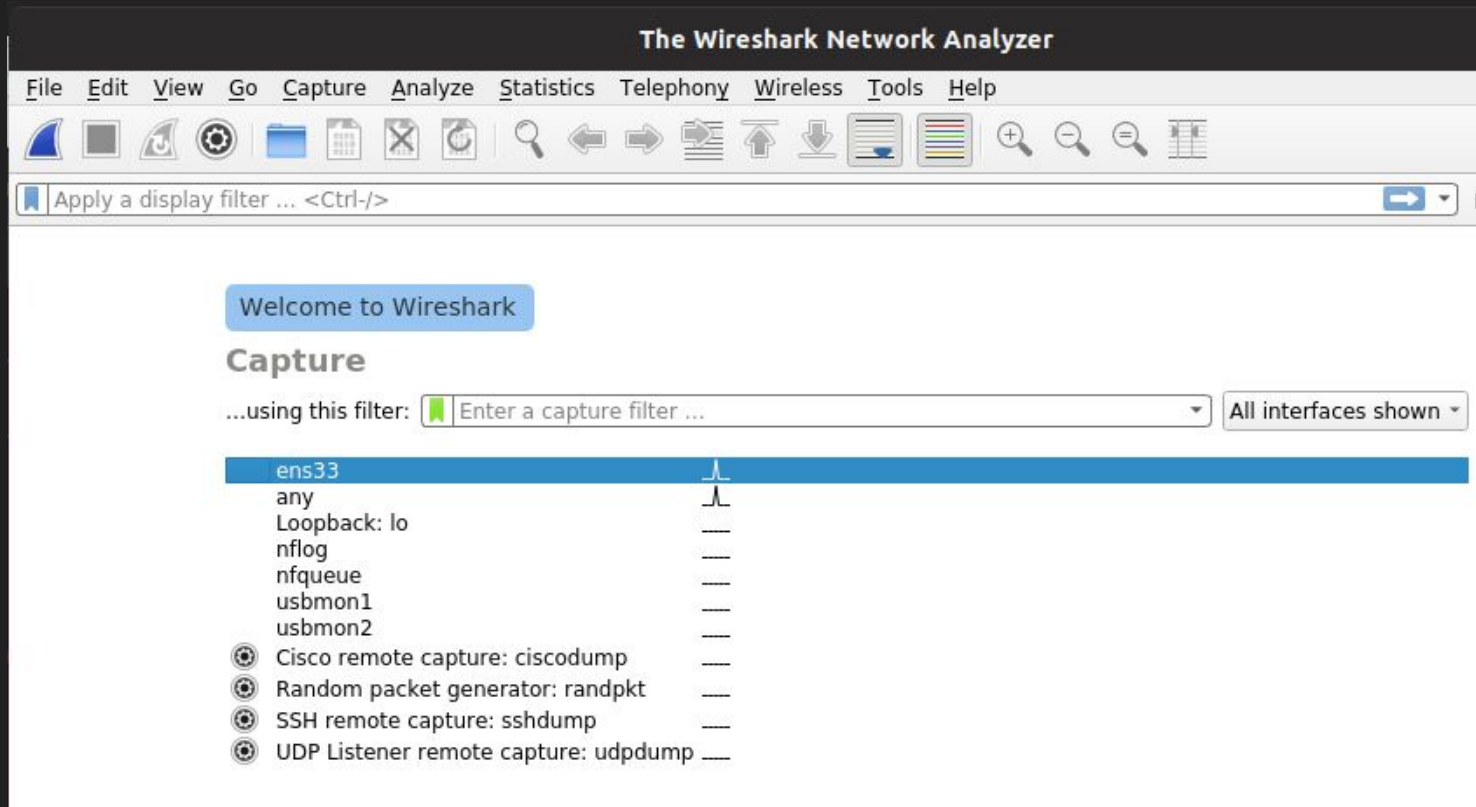
```
> sudo apt install wireshark
```

Or: <https://www.wireshark.org/#download>

Remember: run it as **SuperUser!**

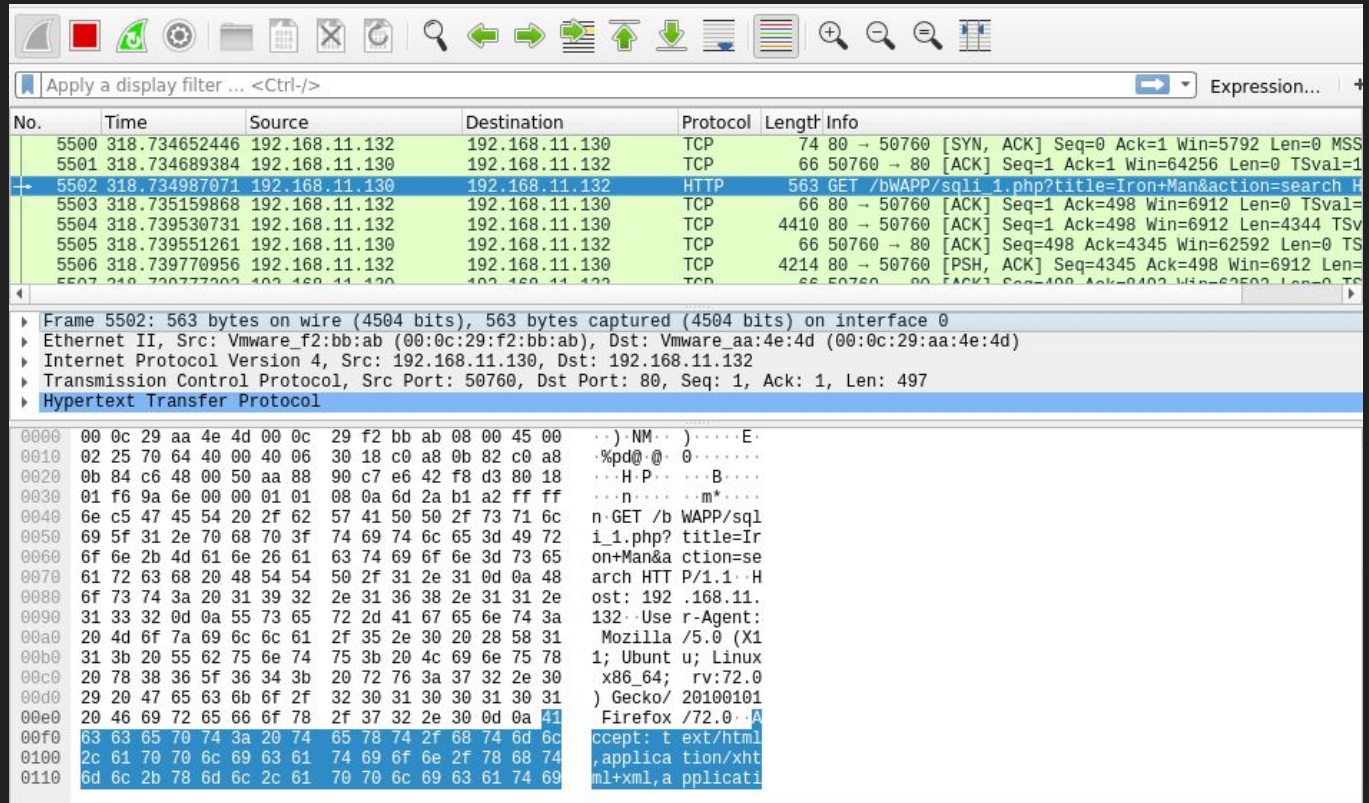


> Let's sniff some traffic!



> Let's sniff some traffic!

Are these
familiar
to you?



The image shows a Wireshark packet capture window. The top toolbar contains various icons for file operations, network analysis, and search. Below the toolbar is a filter bar with the text "Apply a display filter ... <Ctrl-/>". The main packet list table has columns for No., Time, Source, Destination, Protocol, Length, and Info. Packet 5502 is selected, showing an HTTP GET request to /bWAPP/sqli_1.php?title=Iron+Man&action=search. The packet details pane on the right shows the structure of the packet: Ethernet II, Internet Protocol Version 4, and Hypertext Transfer Protocol. The packet bytes pane at the bottom shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
5500	318.734652446	192.168.11.132	192.168.11.130	TCP	74	80 → 50760 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=
5501	318.734689384	192.168.11.130	192.168.11.132	TCP	66	50760 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1
5502	318.734987071	192.168.11.130	192.168.11.132	HTTP	563	GET /bWAPP/sqli_1.php?title=Iron+Man&action=search H
5503	318.735159868	192.168.11.132	192.168.11.130	TCP	66	80 → 50760 [ACK] Seq=1 Ack=498 Win=6912 Len=0 TSval=
5504	318.739530731	192.168.11.132	192.168.11.130	TCP	4410	80 → 50760 [ACK] Seq=1 Ack=498 Win=6912 Len=4344 TSv
5505	318.739551261	192.168.11.130	192.168.11.132	TCP	66	50760 → 80 [ACK] Seq=498 Ack=4345 Win=62592 Len=0 TS
5506	318.739770956	192.168.11.132	192.168.11.130	TCP	4214	80 → 50760 [PSH, ACK] Seq=4345 Ack=498 Win=6912 Len=
5507	318.739773700	192.168.11.130	192.168.11.132	TCP	66	50760 → 80 [ACK] Seq=498 Ack=8403 Win=62592 Len=0 TS

Frame 5502: 563 bytes on wire (4504 bits), 563 bytes captured (4504 bits) on interface 0

Ethernet II, Src: Vmware_f2:bb:ab (00:0c:29:f2:bb:ab), Dst: Vmware_aa:4e:4d (00:0c:29:aa:4e:4d)

Internet Protocol Version 4, Src: 192.168.11.130, Dst: 192.168.11.132

Transmission Control Protocol, Src Port: 50760, Dst Port: 80, Seq: 1, Ack: 1, Len: 497

Hypertext Transfer Protocol

0000 00 0c 29 aa 4e 4d 00 0c 29 f2 bb ab 08 00 45 00 ..).NM..).....E..

0010 02 25 70 64 40 00 40 06 30 18 c0 a8 0b 82 c0 a8 %pd@.@. 0.....

0020 0b 84 c6 48 00 50 aa 88 90 c7 e6 42 f8 d3 80 18 ...H.P...B....

0030 01 f6 9a 6e 00 00 01 01 08 0a 6d 2a b1 a2 ff ff ...n....m*....

0040 6e c5 47 45 54 20 2f 62 57 41 50 50 2f 73 71 6c n-GET /b WAPP/sqli

0050 69 5f 31 2e 70 68 70 3f 74 69 74 6c 65 3d 49 72 i_1.php? title=Ir

0060 6f 6e 2b 4d 61 6e 26 61 63 74 69 6f 6e 3d 73 65 on+Man&a ction=se

0070 61 72 63 68 20 48 54 54 50 2f 31 2e 31 0d 0a 48 arch HTT P/1.1..H

0080 6f 73 74 3a 20 31 39 32 2e 31 36 38 2e 31 31 2e ost: 192.168.11.

0090 31 33 32 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 132..Use r-Agent:

00a0 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 58 31 Mozilla /5.0 (X1

00b0 31 3b 20 55 62 75 6e 74 75 3b 20 4c 69 6e 75 78 1; Ubuntu u; Linux

00c0 20 78 38 36 5f 36 34 3b 20 72 76 3a 37 32 2e 30 x86_64; rv:72.0

00d0 29 20 47 65 63 6b 6f 2f 32 30 31 30 30 31 30 31) Gecko/ 20100101

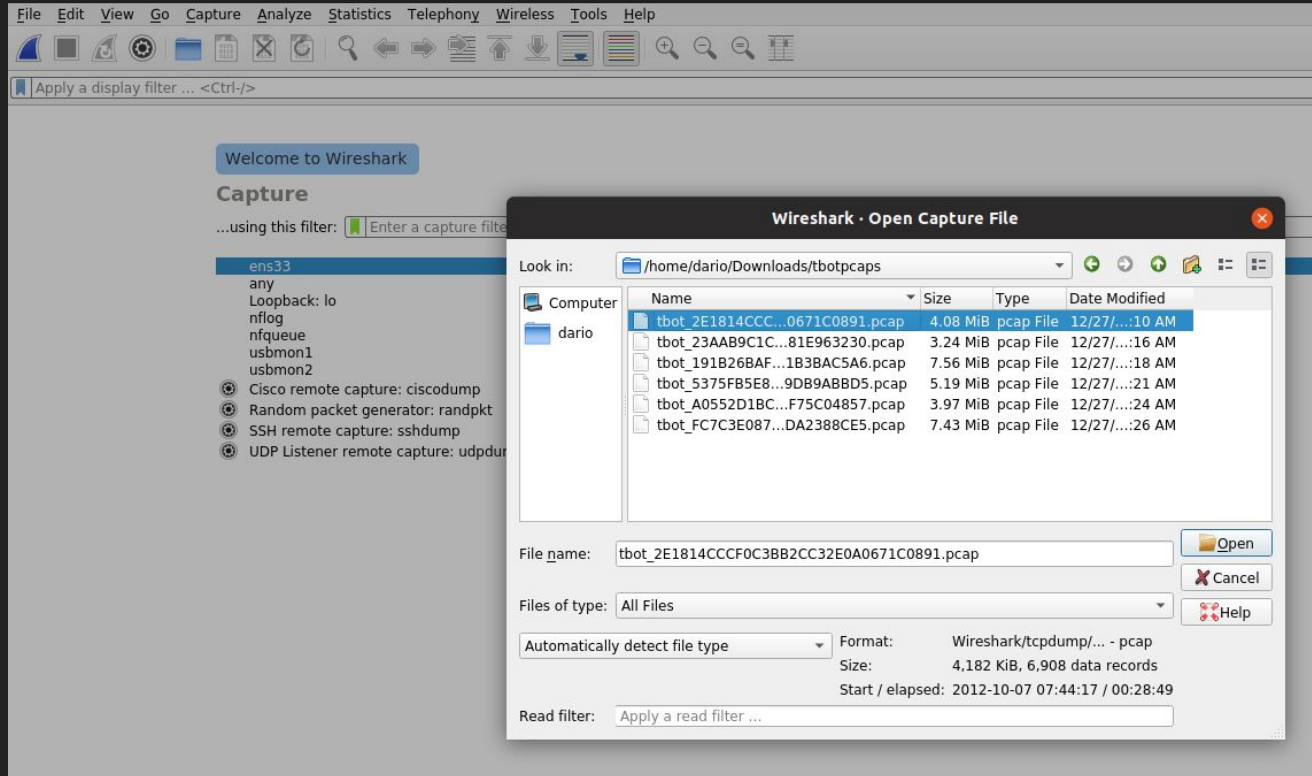
00e0 20 46 69 72 65 66 6f 78 2f 37 32 2e 30 0d 0a 41 Firefox /72.0..A

00f0 63 63 65 70 74 3a 20 74 65 78 74 2f 68 74 6d 6c ccept: t ext/html

0100 2c 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 68 74 ,applicati on/xht

0110 6d 6c 2b 78 6d 6c 2c 61 70 70 6c 69 63 61 74 69 ml+xml,a pplicati

> Wireshark: upload a PCAP file



> Wireshark: apply a filter

In the Filter tab, it is possible to filter results.

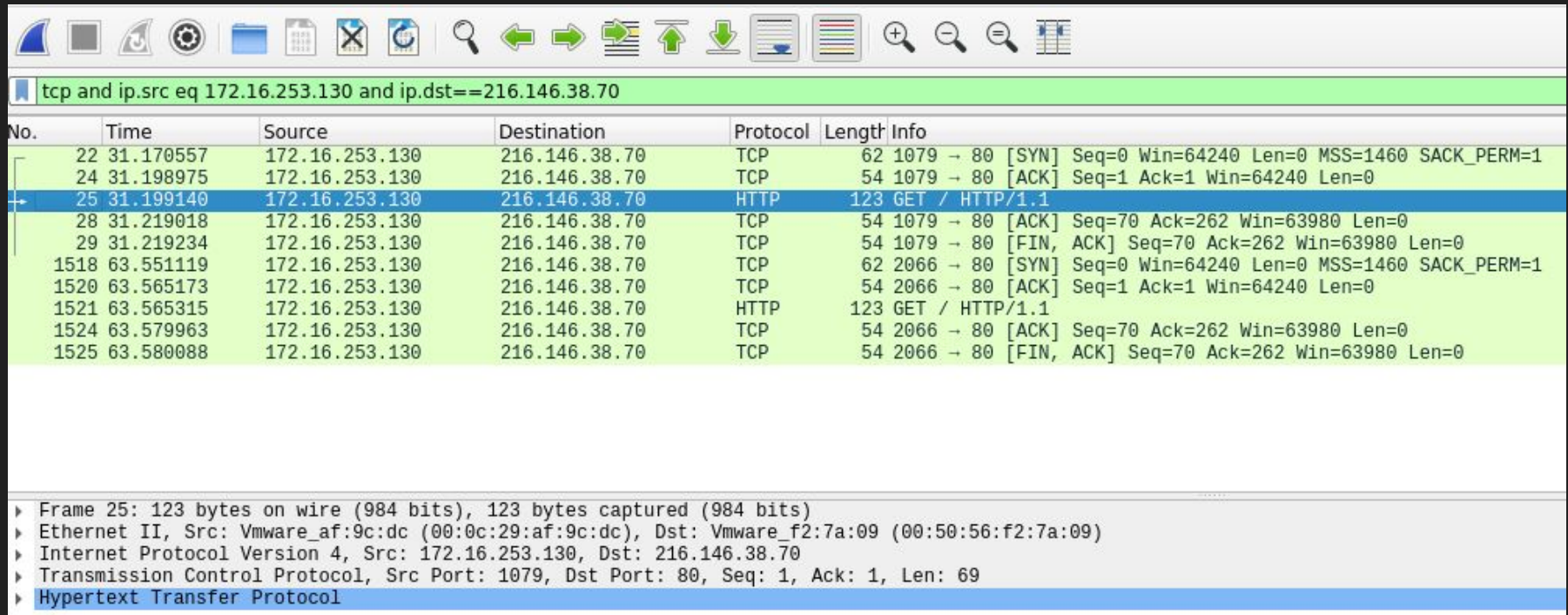
- Source or destination IP: `ip.src==1.1.1.1 and ip.dst==2.2.2.2`
- Port and kind of traffic: `tcp.port eq 25 or icmp`
- Packets containing bytes sequences: `udp[8:3]==81:60:03`
- Packets containing a sequence everywhere: `udp contains 81:60:03`
- Part of a MAC address: `eth.addr[0:3]==00:06:5B`

...More? <https://wiki.wireshark.org/DisplayFilters>



SCAN ME

> Wireshark: apply a filter in practice



The image shows the Wireshark network protocol analyzer interface. At the top, the packet capture filter is set to `tcp and ip.src eq 172.16.253.130 and ip.dst==216.146.38.70`. Below the filter, a list of captured packets is displayed. The selected packet is number 25, which is an HTTP GET request. The packet details pane at the bottom shows the structure of this packet: Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol.

No.	Time	Source	Destination	Protocol	Length	Info
22	31.170557	172.16.253.130	216.146.38.70	TCP	62	1079 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
24	31.198975	172.16.253.130	216.146.38.70	TCP	54	1079 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
25	31.199140	172.16.253.130	216.146.38.70	HTTP	123	GET / HTTP/1.1
28	31.219018	172.16.253.130	216.146.38.70	TCP	54	1079 → 80 [ACK] Seq=70 Ack=262 Win=63980 Len=0
29	31.219234	172.16.253.130	216.146.38.70	TCP	54	1079 → 80 [FIN, ACK] Seq=70 Ack=262 Win=63980 Len=0
1518	63.551119	172.16.253.130	216.146.38.70	TCP	62	2066 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
1520	63.565173	172.16.253.130	216.146.38.70	TCP	54	2066 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
1521	63.565315	172.16.253.130	216.146.38.70	HTTP	123	GET / HTTP/1.1
1524	63.579963	172.16.253.130	216.146.38.70	TCP	54	2066 → 80 [ACK] Seq=70 Ack=262 Win=63980 Len=0
1525	63.580088	172.16.253.130	216.146.38.70	TCP	54	2066 → 80 [FIN, ACK] Seq=70 Ack=262 Win=63980 Len=0

▶ Frame 25: 123 bytes on wire (984 bits), 123 bytes captured (984 bits)
▶ Ethernet II, Src: Vmware_af:9c:dc (00:0c:29:af:9c:dc), Dst: Vmware_f2:7a:09 (00:50:56:f2:7a:09)
▶ Internet Protocol Version 4, Src: 172.16.253.130, Dst: 216.146.38.70
▶ Transmission Control Protocol, Src Port: 1079, Dst Port: 80, Seq: 1, Ack: 1, Len: 69
▶ Hypertext Transfer Protocol

> PyShark

Python wrapper for tshark, allowing python packet parsing using wireshark dissectors

Installation:

```
> sudo apt install python-pip  
> sudo pip install pyshark
```

Or:

```
> git clone https://github.com/KimiNewt/pyshark.git  
> cd pyshark/src  
> python setup.py install
```

> PyShark: usage

```
>>> import pyshark
>>> cap = pyshark.FileCapture('/tmp/mycapture.cap')
>>> cap
<FileCapture /tmp/mycapture.cap (589 packets)>
>>> print cap[0]
Packet (Length: 698)
Layer ETH:
    Destination: BLANKED
    Source: BLANKED
    Type: IP (0x0800)
Layer IP:
    Version: 4
    Header Length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 684
    Identification: 0x254f (9551)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 1
    Protocol: UDP (17)
    ...
```

> PyShark: reading from a live interface

```
>>> capture = pyshark.LiveCapture(interface='eth0')
>>> capture.sniff(timeout=50)
>>> capture
<LiveCapture (5 packets)>
>>> capture[3]
<UDP/HTTP Packet>

for packet in capture.sniff_continuously(packet_count=5):
    print 'Just arrived:', packet
```

> PyShark: access packet data

```
>>> packet['ip'].dst
```

```
192.168.0.1
```

```
>>> packet.ip.src
```

```
192.168.0.100
```

```
>>> packet[2].src
```

```
192.168.0.100
```

Check if a packet has a layer:

```
>>> 'IP' in packet
```

```
True
```

> PyShark: Decrypting packet captures

It is possible to specify a password for a pcap file:

```
>>> cap1 = pyshark.FileCapture('/tmp/capture1.cap',  
decryption_key='password')
```

It is possible to specify a password and an encryption type for a live capture:

```
>>> cap2 = pyshark.LiveCapture(interface='wi0',  
decryption_key='password', encryption_type='wpa-psk')
```

> PyShark

...More?

<https://github.com/KimiNewt/pyshark>



SCAN ME

> Scapy

A Python program that enables the user to send, sniff and dissect and forge network packets.

Installation:

```
> sudo apt install python-pip
```

```
> sudo pip install scrapy
```

Or:

```
> git clone https://github.com/secdev/scapy.git
```

```
> cd scapy
```

```
> sudo python setup.py install
```

```

aSPY//YASa
  apyyyyCY/////////YCa
    sY////////YSpCs   scpCY//Pp
ayp ayyyyyySCP//Pp           syY//C
AYAsAYYYYYYYY//Ps           cY//S
  pCCCCY//p                   cSSps y//Y
    SPPPP///a                 pP///AC//Y
      A//A                     cyP////C
        p///Ac                 sC///a
          P///YCpc             A//A
    sccccp///pSP///p         p//Y
  sY/////////y   caa         S//P
  cayCyayP//Ya             pY/Ya
    sY/PsY////YCc           aC//Yp
      sc   sccaCY//PCypaapyCP//YSs
        spCPY////////YPSps
          ccaacs

```

> Scapy: dependencies and usage

Dependencies:

- Linux: tcpdump (scapy is native)
- MAC OS: libpcap (scapy is native)
- Windows: python, Npcap, Scapy (**Avoid if possible!**)

Usage:

- Terminal: `> sudo scapy`
- Script: `from scapy.all import *`

> Scapy: build a packet

```
>>> a=IP(ttl=10) # create an IP packet with ttl=10 (default 64)
>>> a # show the packet
< IP ttl=10 |>
>>> a.dst="192.168.1.1" # add a destination IP
>>> a # show the packet
< IP ttl=10 dst=192.168.1.1 |>
>>> del(a.ttl) # delete the ttl from the packet (back to default)
>>> a # show the packet
< IP dst=192.168.1.1 |>
```

> Scapy: stacking layers

```
>>> IP() # build an IP packet
```

```
<IP |>
```

```
>>> IP()/TCP() # stacking protocols ('/') with default values
```

```
<IP frag=0 proto=TCP |<TCP |>>
```

```
>>> Ether()/IP()/TCP() # stack Ether/IP/TCP
```

```
<Ether type=0x800 |<IP frag=0 proto=TCP |<TCP |>>>
```

```
>>> IP()/TCP()/"GET / HTTP/1.0\r\n\r\n" # stack IP/TCP and a payload
```

```
<IP frag=0 proto=TCP |<TCP |<Raw load='GET / HTTP/1.0\r\n\r\n' |>>>
```

> Scapy: dissect packets

```
>>> a=Ether()/IP(dst="www.slashdot.org")/TCP()/"GET /index.html HTTP/1.0 \n\n" # build Eth frame
>>> hexdump(a) # show packet in hexadecimal
00 02 15 37 A2 44 00 AE F3 52 AA D1 08 00 45 00  ...7.D...R....E.
00 43 00 01 00 00 40 06 78 3C C0 A8 05 15 42 23  .C....@.x<....B#
FA 97 00 14 00 50 00 00 00 00 00 00 00 50 02  ....P.....P.
20 00 BB 39 00 00 47 45 54 20 2F 69 6E 64 65 78  ..9..GET /index
2E 68 74 6D 6C 20 48 54 54 50 2F 31 2E 30 20 0A  .html HTTP/1.0 .
0A
.
>>> b=raw(a) # get raw bytes
>>> b # show raw packet
'\x00\x02\x15\x37\xa2\x44\x00\xae\xf3\x52\xaa\xd1\x08\x00\x45\x00\x00\x43\x00\x01\x00\x00\x40\x06\x78\x3c\xc0\xa8\x05\x15\x42\x23\xfa\x97\x00\x14\x00\x50\x00\x00\x00\x00\x50\x02\x20\x00\xbb\x39\x00\x00\x47\x45\x54\x20\x2f\x69\x6e\x64\x65\x78\x2e\x68\x74\x6d\x6c\x20\x48\x54\x54\x50\x2f\x31\x2e\x30\x20\x0a\x0a'
>>> c=Ether(b) # turn the raw packet back to an Ethernet frame
>>> c # show the Ethernet frame
<Ether dst=00:02:15:37:a2:44 src=00:ae:f3:52:aa:d1 type=0x800 |<IP version=4L
ihl=5L tos=0x0 len=67 id=1 flags= frag=0L ttl=64 proto=TCP chksum=0x783c
src=192.168.5.21 dst=66.35.250.151 options='' |<TCP sport=20 dport=80 seq=0L
ack=0L dataofs=5L reserved=0L flags=S window=8192 chksum=0xbb39 urgptr=0
options=[] |<Raw load='GET /index.html HTTP/1.0 \n\n' |>>>>
```

> Scapy: read PCAP files

```
>>> a=rdpcap("/spare/captures/isakmp.cap") # import a pcap
>>> a
<isakmp.cap: UDP:721 TCP:0 ICMP:0 Other:0> # recap of the file
```

To extract info from the pcap:

```
from scapy.all import *
scapy_cap = rdpcap('capture.pcap')
for packet in scapy_cap:
    print packet.src
```

The previous script will extract all MAC files

> Scapy

More info about Scapy:

<https://scapy.readthedocs.io/en/latest/introduction.html>



SCAN ME

> Iptables

- Command line utility to configure Linux kernel Firewall
- It is used for IPv4 (*ip6tables* is used for IPv6)
- It can inspect, modify, forward, redirect and/or drop packets



> Iptables: basic concepts

- Organized in a collection of Tables (5)
- Each table is made of a set of Chains
- Each chain contains a list of Rules
- Rules are applied IN ORDER

FILTER TABLE	NAT TABLE	MANGLE TABLE	RAW TABLE
INPUT	OUTPUT	INPUT	OUTPUT
OUTPUT	PREROUTING	OUTPUT	PREROUTING
FORWARD	POSTROUTING	FORWARD	
		PREROUTING	
		POSTROUTING	

> Iptables: basic concepts

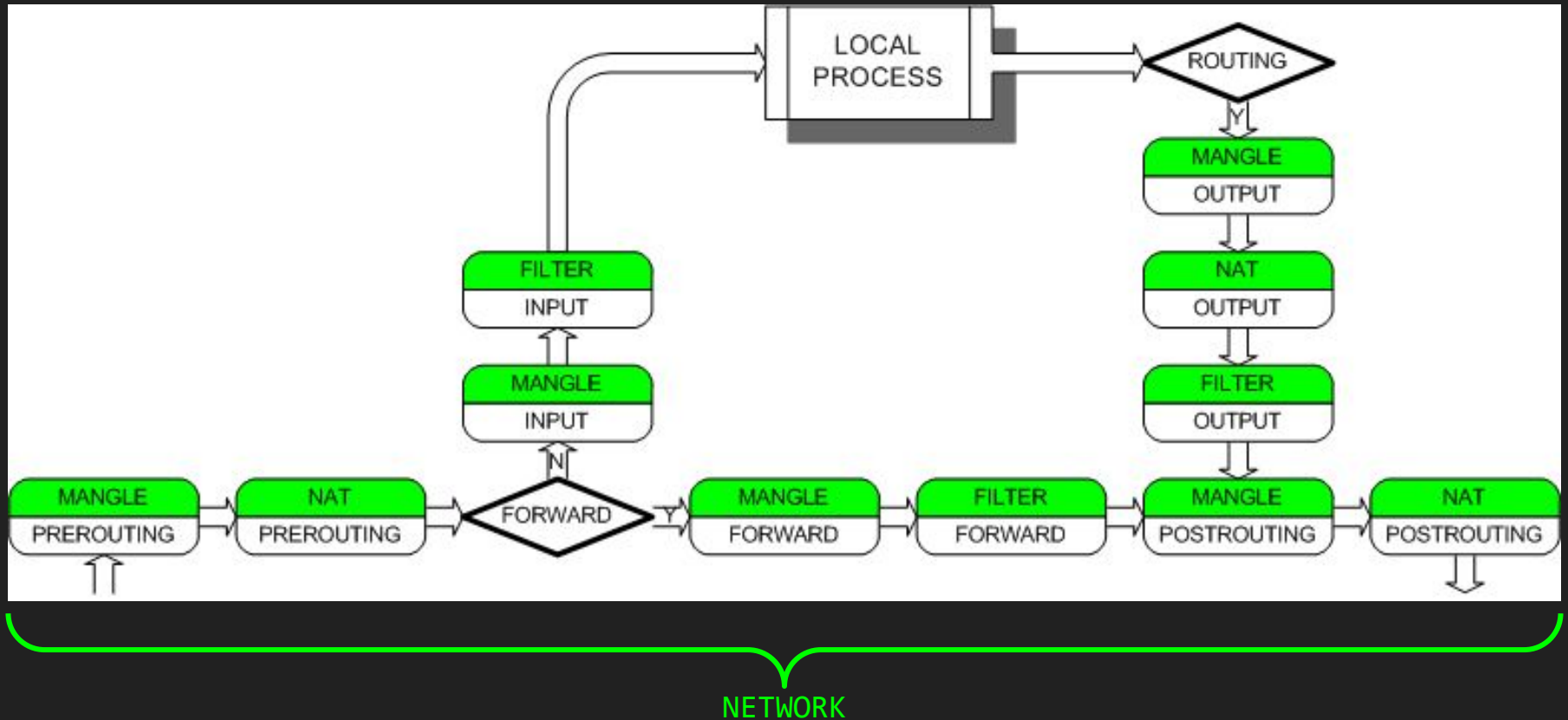
In most cases, you will use the following tables:

- FILTER: the default table, all the actions associated to the firewall will take place here
- NAT: used for network address translation (e.g. port forwarding)

The following tables will be used for complex configurations involving multiple routers:

- RAW: to configure packets to avoid connection tracking
- MANGLE: for specialized packet alterations

> Iptables: Table traverse



> Iptables: list current rules

No rules by default:

> `sudo iptables -L` (by default it shows FILTER)

```
dario@ubuntu:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                               destination
dario@ubuntu:~$
```

To list other Tables:

> `sudo iptables -t NAT -L`

> iptables: add rules

> sudo iptables -A INPUT -p tcp -s 192.168.1.2 -i eth0 -j DROP

- -A : select CHAIN
- -p: protocol (“all”, “udp”, “tcp”, ...)
- -s: source IP address
- (-d: destination IP address)
- -i: select input interface
- (-o: select output interface)
- -j: jump to Target (“ACCEPT”, “DROP”, ...)

More? <https://wiki.ubuntu-it.org/Sicurezza/Iptables>



SCAN ME

> iptables: delete rules

> sudo iptables -A INPUT -p tcp -s 192.168.1.2 -j ACCEPT

> sudo iptables -A INPUT -j DROP

```
dario@ubuntu:~$ sudo iptables -A INPUT -p tcp -s 192.168.1.2 -j ACCEPT
dario@ubuntu:~$ sudo iptables -A INPUT -j DROP
dario@ubuntu:~$ sudo iptables -L --line-numbers
Chain INPUT (policy ACCEPT)
num  target      prot opt source                destination
1    ACCEPT      tcp  --  192.168.1.2            anywhere
2    DROP        all  --  anywhere               anywhere

Chain FORWARD (policy ACCEPT)
num  target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
num  target      prot opt source                destination
dario@ubuntu:~$
```

> sudo iptables -D INPUT 1

- -D: delete rule number X

> NetfilterQueue

Provides access to packets matched by an iptables rule in Linux.

Installation:

```
> sudo apt install python-pip  
> sudo pip install NetfilterQueue
```

Or:

```
> git clone git@github.com:kti/python-netfilterqueue.git  
> cd python-netfilterqueue  
> python setup.py install
```

Dependencies:

```
> sudo apt install build-essential python-dev libnetfilter-queue-dev
```

> NetfilterQueue: usage

```
from netfilterqueue import NetfilterQueue

def print_and_accept(pkt):
    print(pkt) # print a description of the packet
    pkt.accept() # accept the incoming packet

nfqueue = NetfilterQueue()
nfqueue.bind(1, print_and_accept)
try:
    nfqueue.run()
except KeyboardInterrupt:
    print('')

nfqueue.unbind()
```


> NetfilterQueue

...More?

<https://pypi.org/project/NetfilterQueue/>



> Nmap

An open source tool for network exploration and security auditing.

Installation:

```
> sudo apt update
```

```
> sudo apt install nmap
```



NMAP

> Nmap usage

> nmap 192.168.1.1

Basic options:

- -p : port range [-p22; -p1-65535]
- -O : enable OS detection
- -v : verbose
- -A: OS detection + version detection + script scanning + traceroute

More?

<https://nmap.org/book/man-briefoptions.html>



> Nmap in action

```
root@kali:~# nmap -A 192.168.65.129
Starting Nmap 7.70 ( https://nmap.org ) at 2018-05-10 07:41 EDT
Nmap scan report for 192.168.65.129
Host is up (0.0011s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd 2.4.10 ((Debian))
|_http-server-header: Apache/2.4.10 (Debian)
|_http-title: Apache2 Debian Default Page: It works
111/tcp   open  rpcbind 2-4 (RPC #100000)
|_rpcinfo:
|   program version  port/proto  service
|   100000   2,3,4      111/tcp    rpcbind
|   100000   2,3,4      111/udp    rpcbind
|   100024   1          49702/tcp  status
|_  100024   1          59428/udp  status
MAC Address: 00:0C:29:84:93:75 (VMware)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop

TRACEROUTE
HOP RTT      ADDRESS
1   1.13 ms  192.168.65.129
```

> Nmap: port classification

- Open:

An application actively accept TCP connection and UDP datagram.
Finding those kind of ports is your objective!

- Closed:

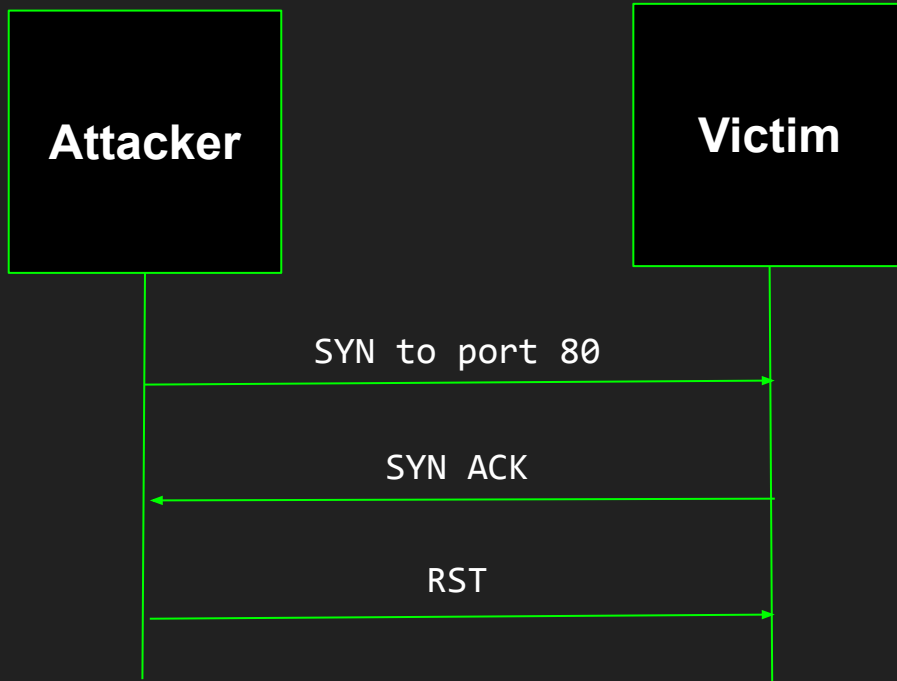
This port is accessible, but no applications are listening on it.
It could be interesting repeating the scan in a second time.

- Filtered:

Nmap can't determine the exact status of the port. This can be due to a firewall or a router.

> Scanning techniques (SYN SCAN)

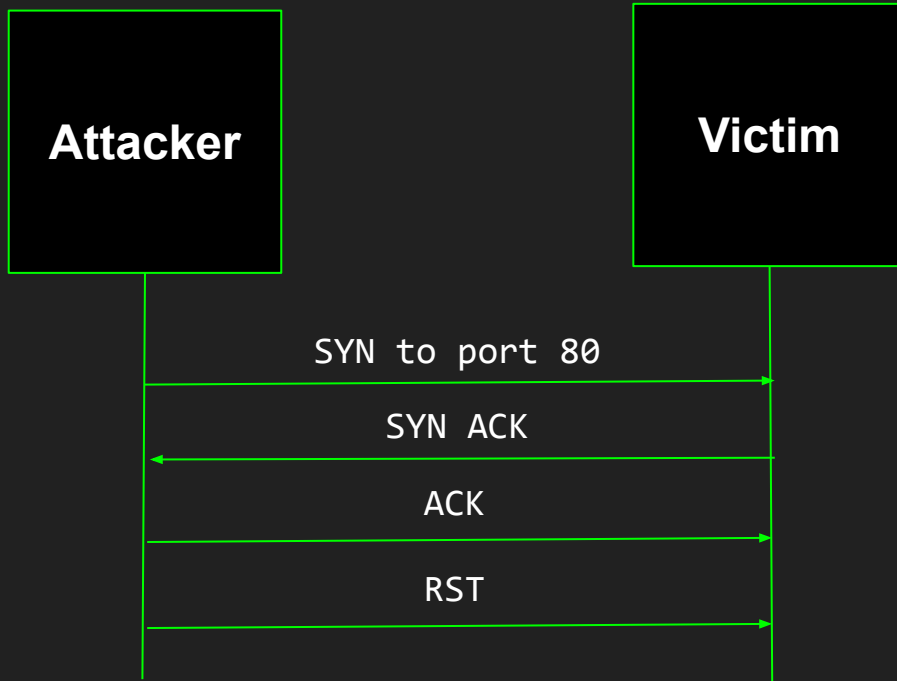
```
> nmap -sS 192.168.1.1
```



- Default option
- Quick (thousands of ports per seconds)
- Hard to discover (connections are never completed)
- Clear differentiation between open, close and filtered

> Scanning techniques (TCP connect SCAN)

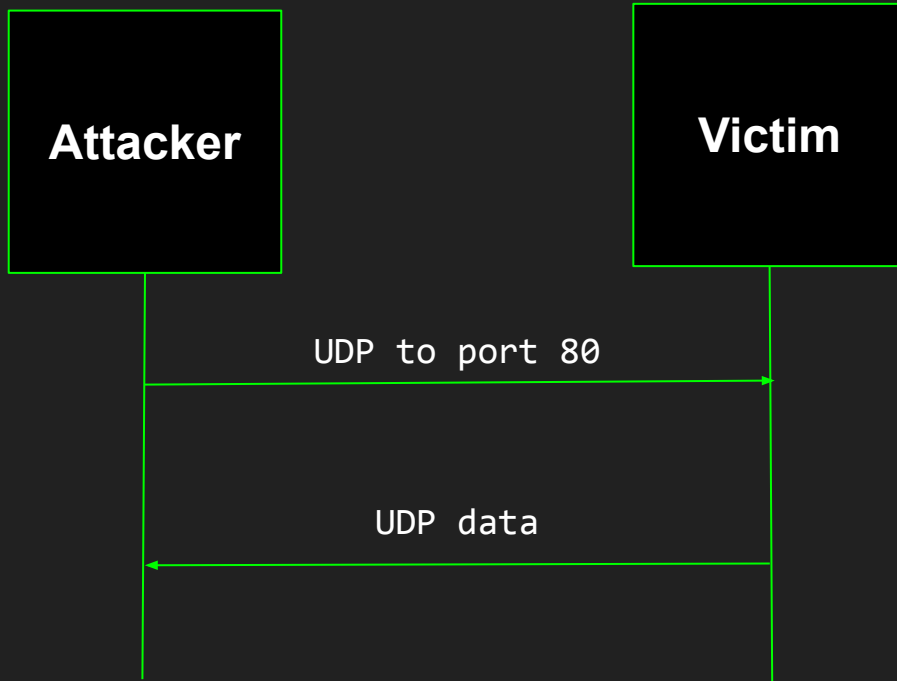
> `nmap -sT 192.168.1.1`



- Try to make a connection
- Slow
- Easy to discover (victims logs TCP connection)
- Use it if SYN SCAN is not possible

> Scanning techniques (UDP SCAN)

```
> nmap -sU 192.168.1.1
```



- Send UDP datagram to every port
- It is hard to get a UDP as reply, tons of filtered port
- Use together to SYN SCAN

> Scanning techniques (NULL|FIN|Xmas SCAN)

NULL scan: No bit sent

```
> nmap -sN 192.168.1.1
```

FIN scan: set FIN bit

```
> nmap -sF 192.168.1.1
```

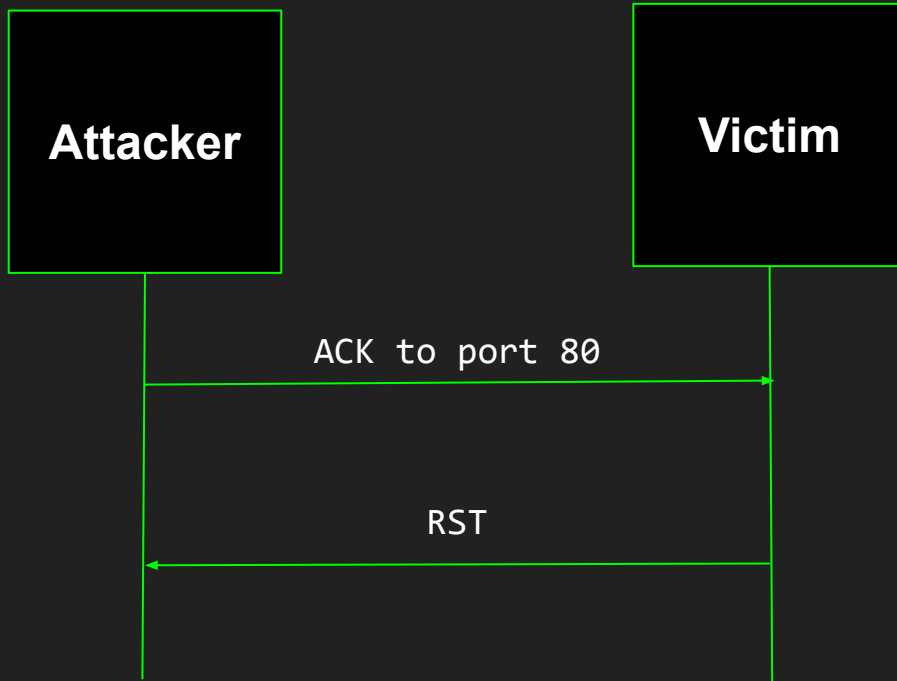
Xmas scan: set FIN PSH URG bits

```
> nmap -sX 192.168.1.1
```

- They behave similarly, but triggers destination ports to reply
- They can access some non-stateful firewalls and packet-filtering routers
- Very hard to discover

> Scanning techniques (TCP ACK SCAN)

> `nmap -sA 192.168.1.1`



- Not used for port status
- Try to map firewall rules
- If a firewall is stateful
- Which ports are filtered (categorized as *'unfiltered'*)

> Scanning techniques

...More?

<https://nmap.org/man/it/man-port-scanning-techniques.html>

SCAN ME



> Scanning techniques (idle SCAN)

Recruits a **Zombie** for scanning the network



> Scanning techniques (idle SCAN)

Basic facts:

- To determine if a port is open: send a SYN to the port. If SYN-ACK → open, if RST → closed
- A machine receiving an unsolicited:
 - SYN-ACK → replies RST
 - RST → ignores it
- Every IP packet has a fragment identification number (IP ID). Many OS simply increment it for each sent packet
- Probing the IP ID can tell how many packets have been sent since the last probe

> Scanning techniques (idle SCAN)

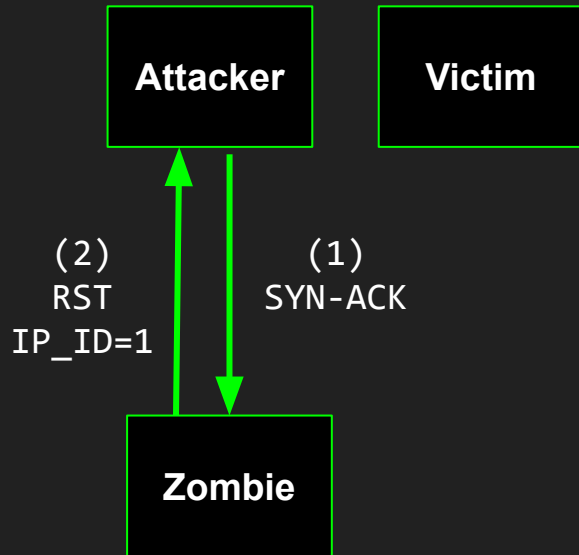
Step by Step:

1. Probe the zombie's IP ID and record it;
2. Forge a SYN from the zombie and send it to the victim;
3. Depending on the port state, the victim may or may not respond causing the zombie IP ID to increment;
4. Probe the zombie's IP ID again.
5. The victim port state can be determined!

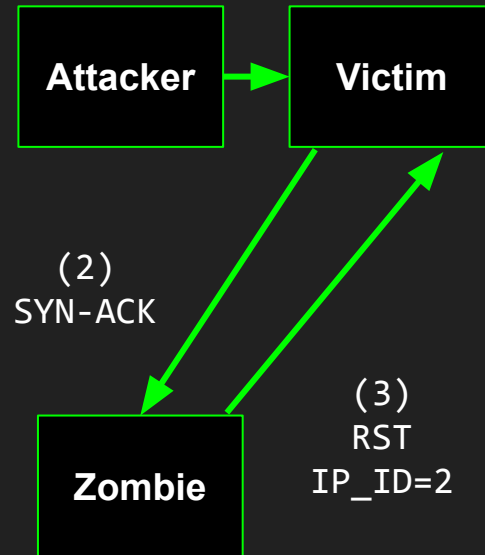
> Scanning techniques (idle SCAN)

Open port:

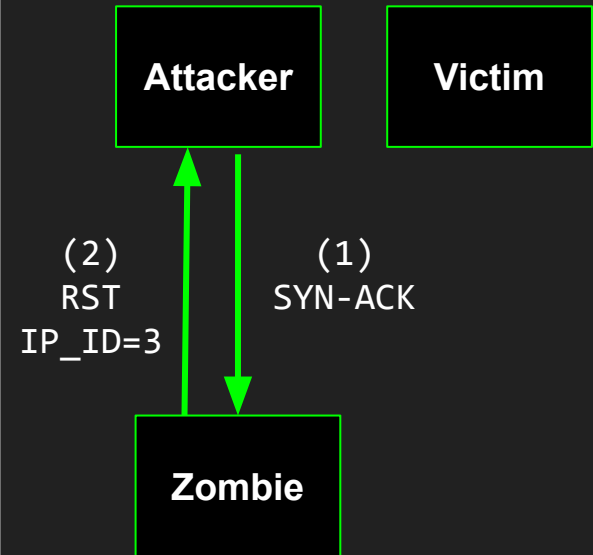
(Step 1)



(Step 2) (1)
SYN from Zombie



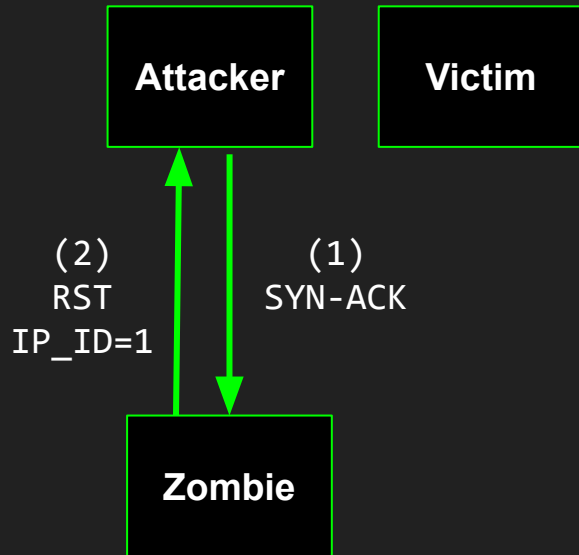
(Step 3)



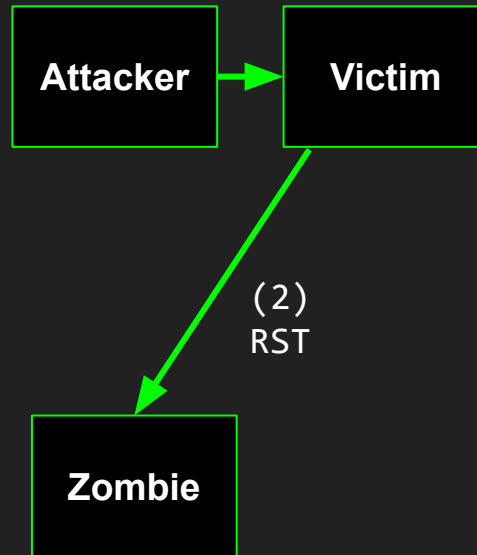
> Scanning techniques (idle SCAN)

Closed port:

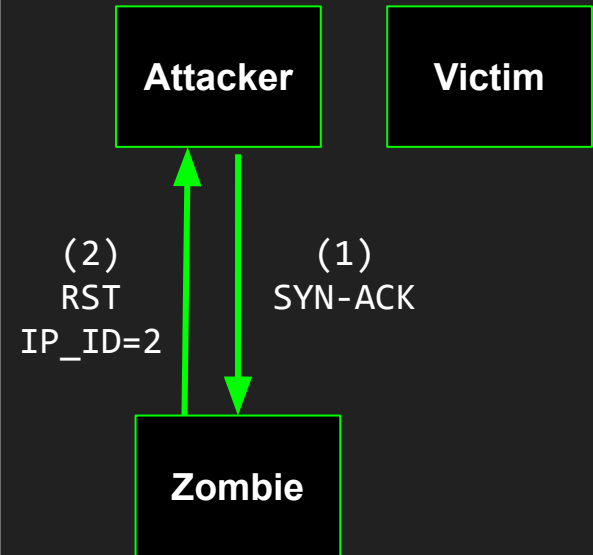
(Step 1)



(Step 2) (1) SYN from Zombie



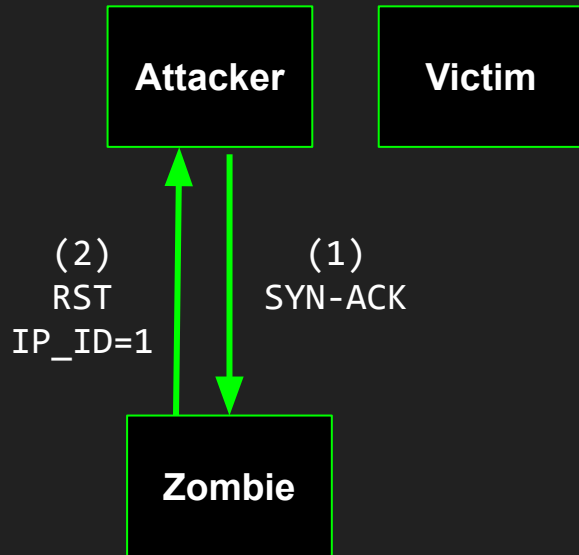
(Step 3)



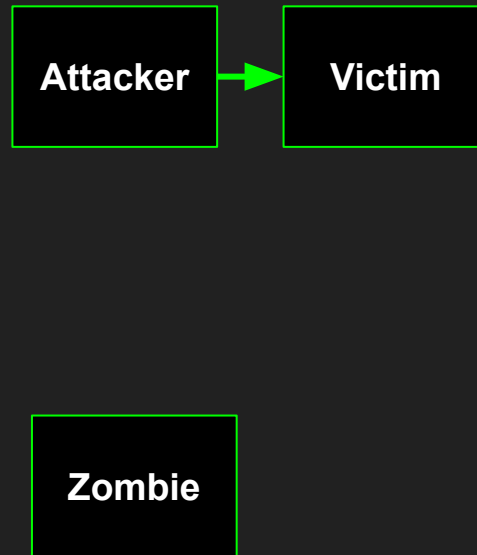
> Scanning techniques (idle SCAN)

Filtered port:

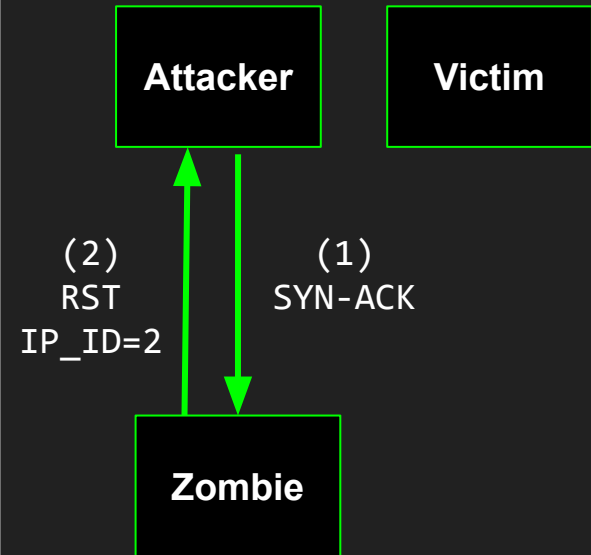
(Step 1)



(Step 2) (1)
SYN from Zombie



(Step 3)



> Scanning techniques (idle SCAN)

Look for a zombie:

```
> nmap -O -v 192.168.1.1
```

- -O : OS detection
- -v : verbose (helps checking if IP ID is incremental)

Result:

...

Network Distance: 1 hop

IP ID Sequence Generation: Incremental

...

> Scanning techniques (idle SCAN)

Launch the scan:

```
> nmap -sI 192.168.1.1 192.168.1.2
```



Zombie Victim

> Scanning techniques (idle SCAN)

...More?

<https://nmap.org/book/idlescan.html>



SCAN ME

> Time for a Quiz Game

Join at:

- www.kahoot.it
- Kahoot! App (Android or iOS)

...wait for the PIN

