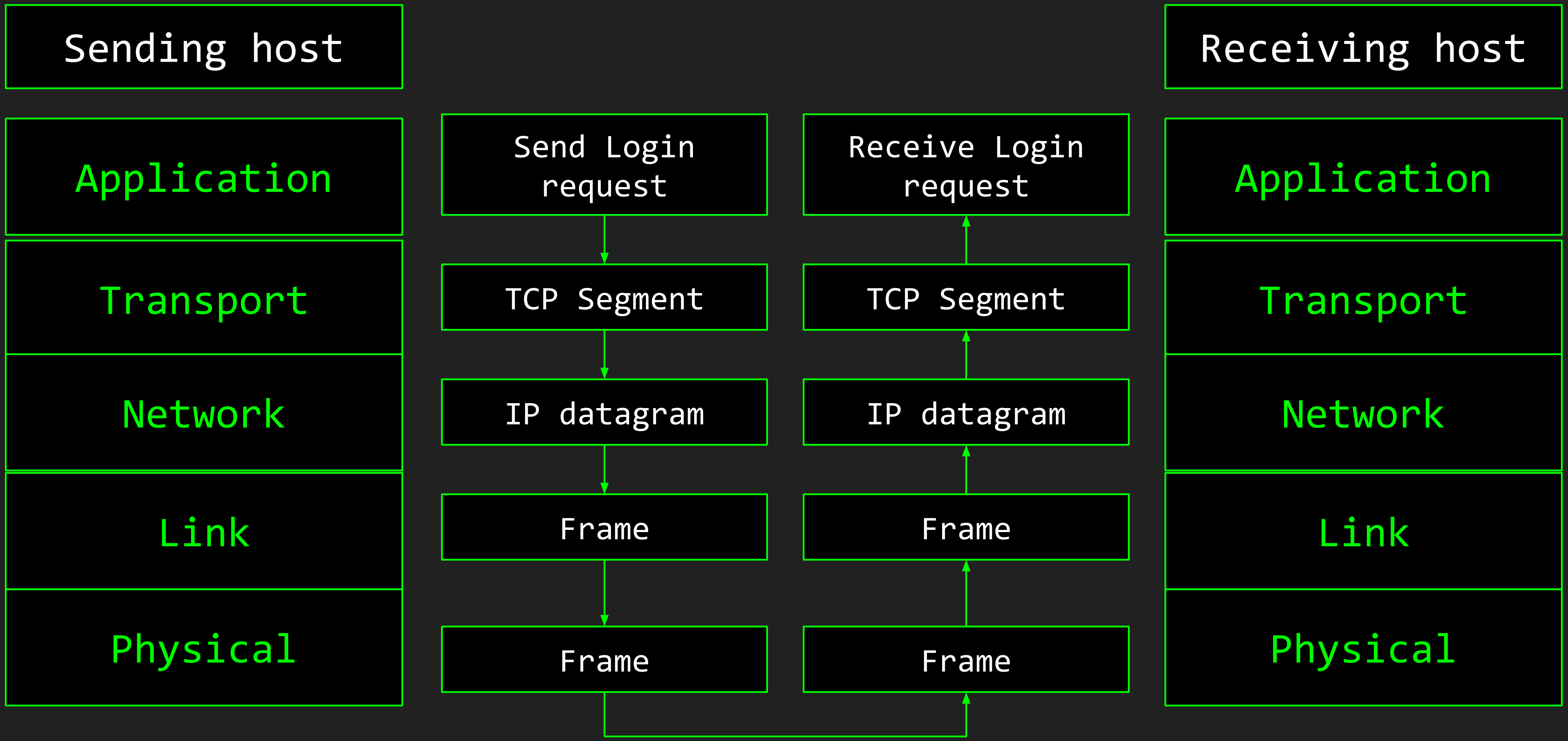
Network Analysis and Monitoring

Network Security

> Internet Stack recap

ISO/OSI Model		TCP/IP Model
Application Presentation Session	HTTP, FTP, DHCP, DNS, HTTPS, SSH, ...	Application
Transport	TCP, UDP	Transport
Network	IPv4, IPv6	Internet
Link	MAC Address	Network Access
Physical	Ethernet, Wi-Fi, ...	Network Access

> Internet Stack recap



> Focus: Transport Layer

Transmission Control Protocol (TCP):

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

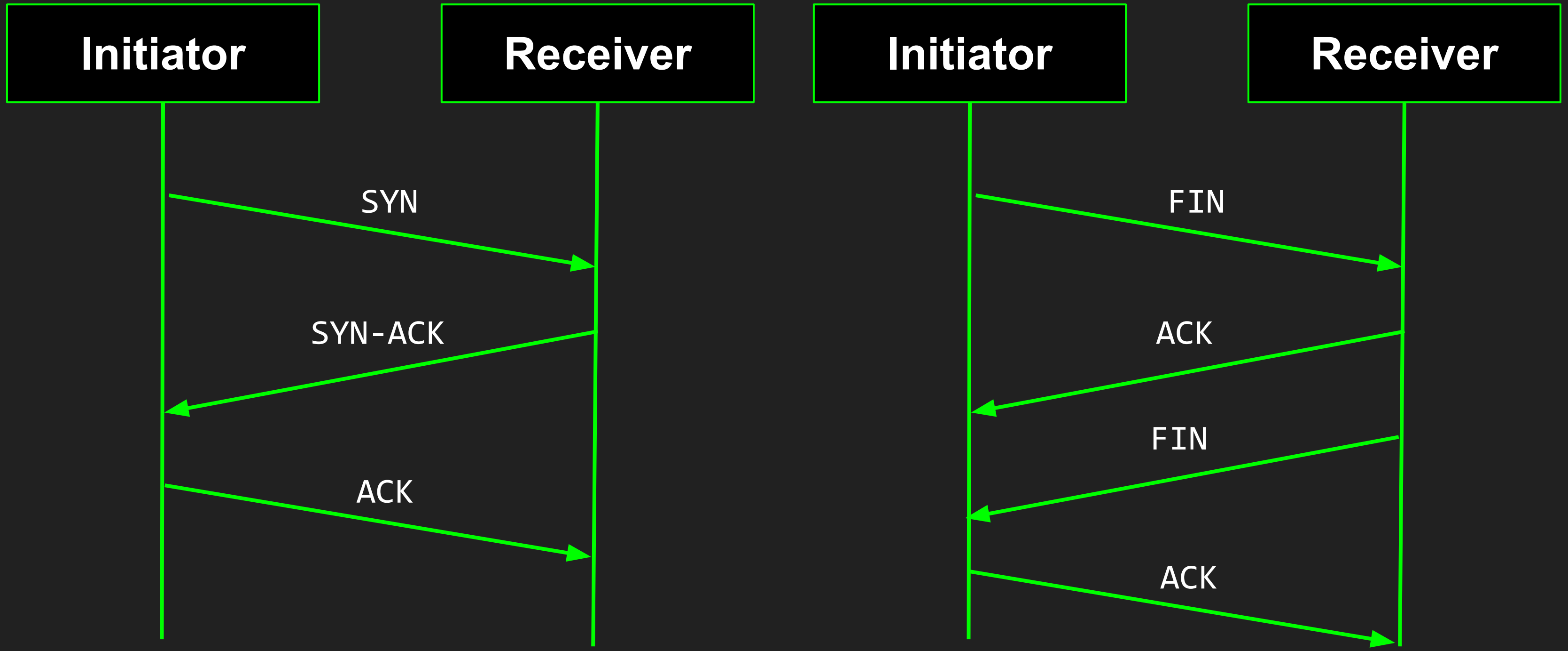
User Datagram Protocol (UDP):

- Best-Effort delivery
- Connection-less

> TCP Segment

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

> 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

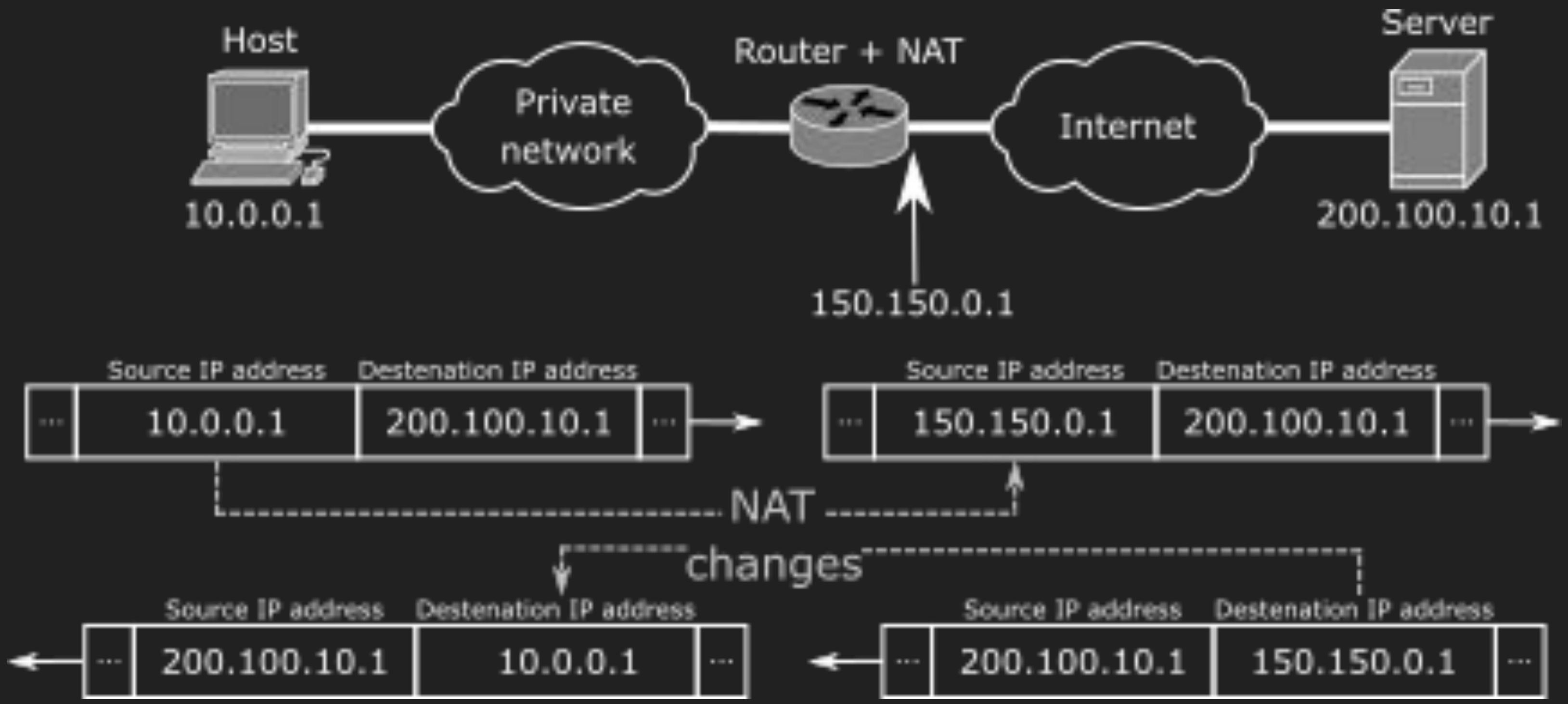
Main approaches:

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

Why?

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

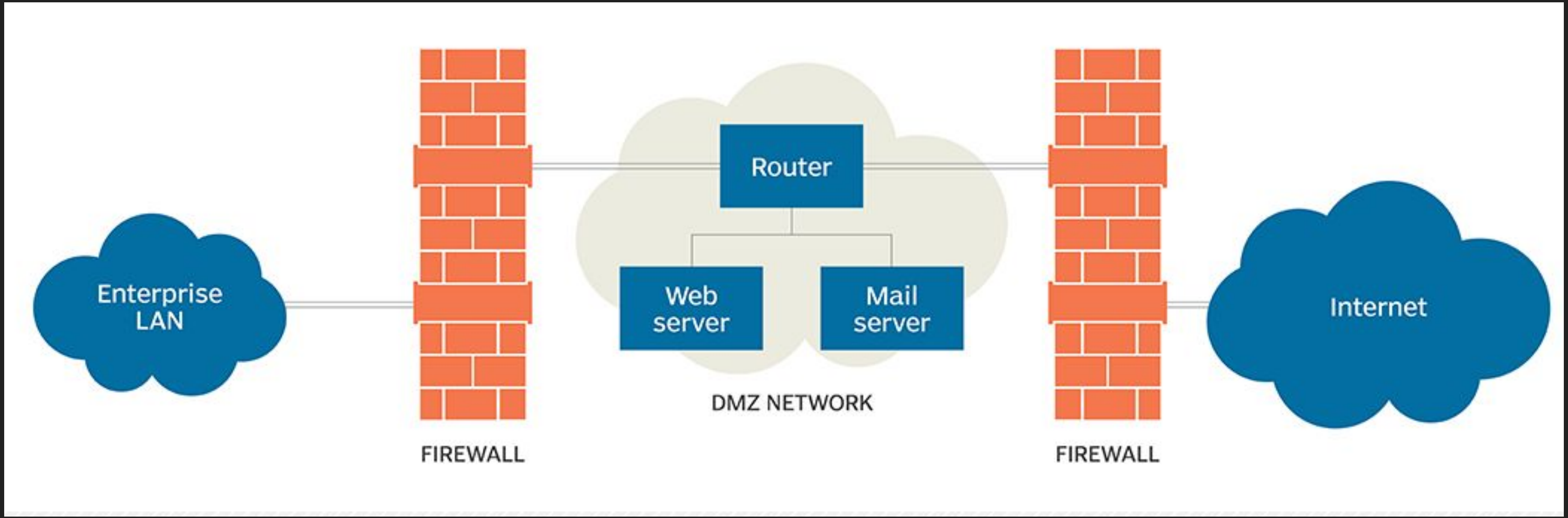
> NAT concept



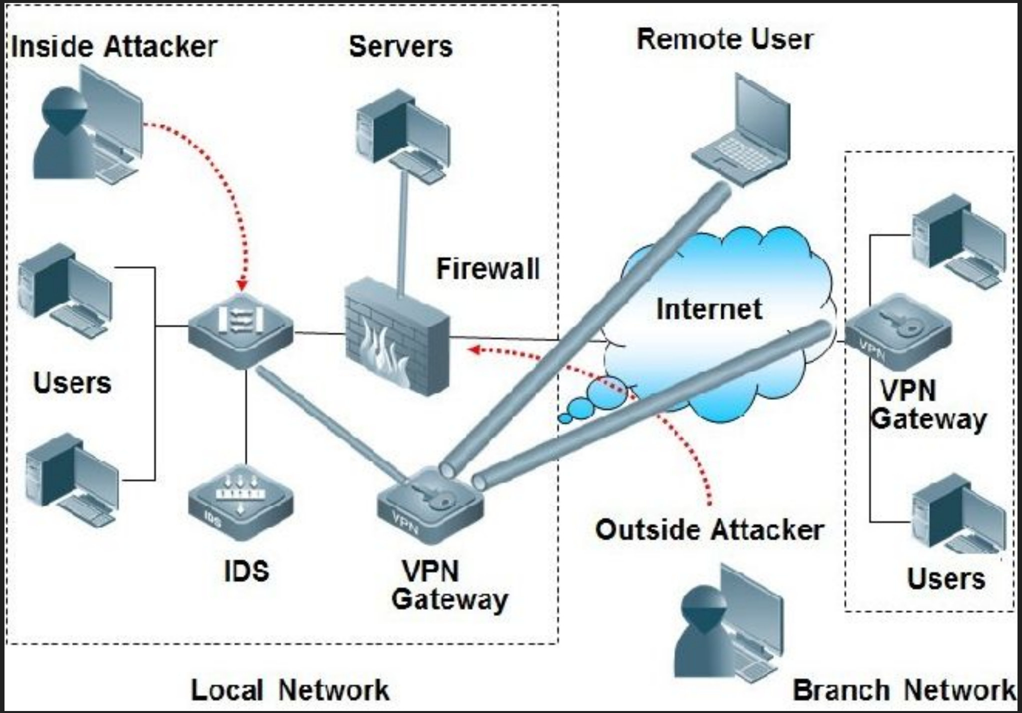
Host
10.0.0.1
Private network
Router + NAT
150.150.0.1
Internet
Server
200.100.10.1
Source IP address
Destenation IP address
10.0.0.1
200.100.10.1
150.150.0.1
200.100.10.1
NAT
changes
200.100.10.1
10.0.0.1
200.100.10.1
150.150.0.1

> 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

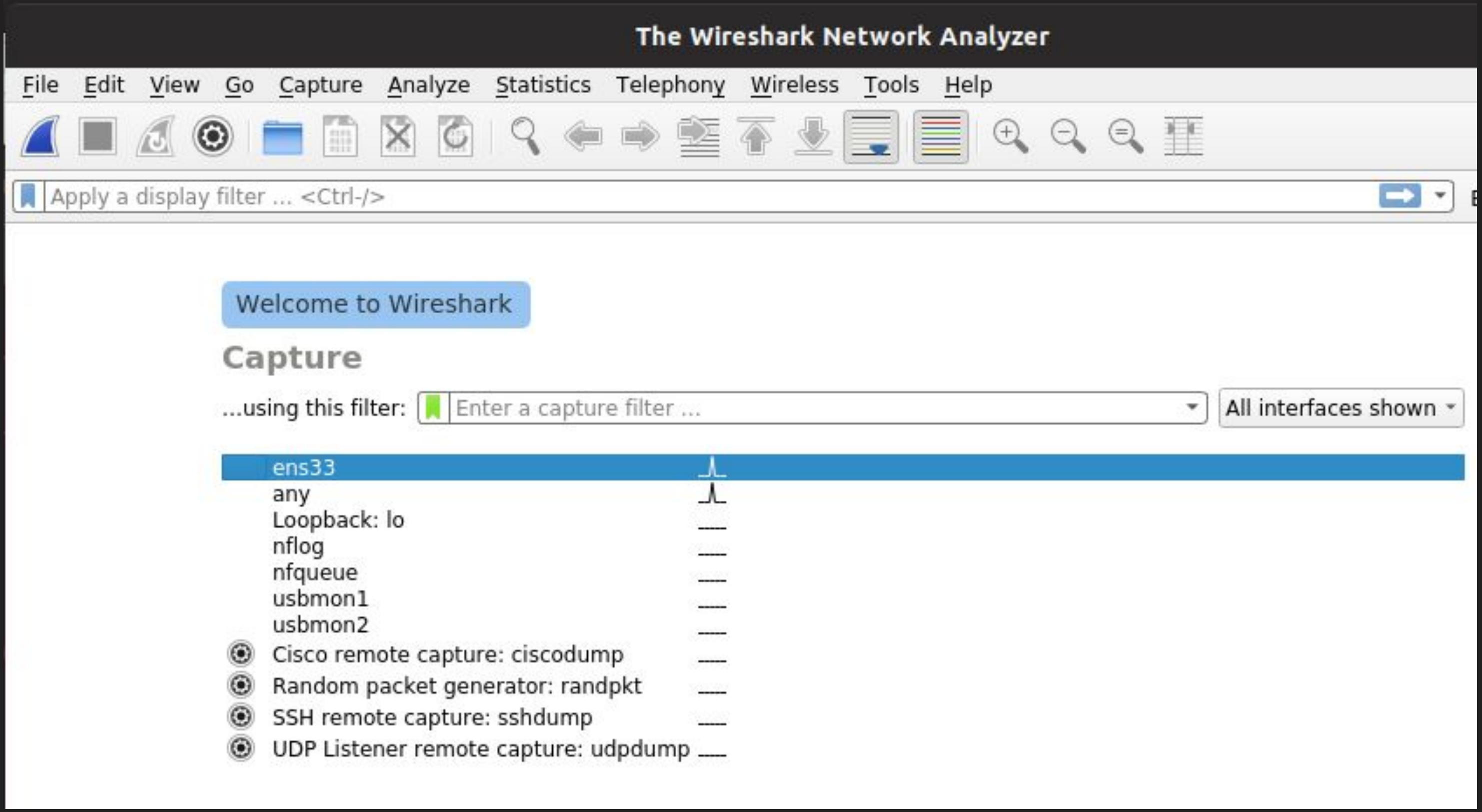
Installation:

```
> sudo apt update
> sudo apt install wireshark
```

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

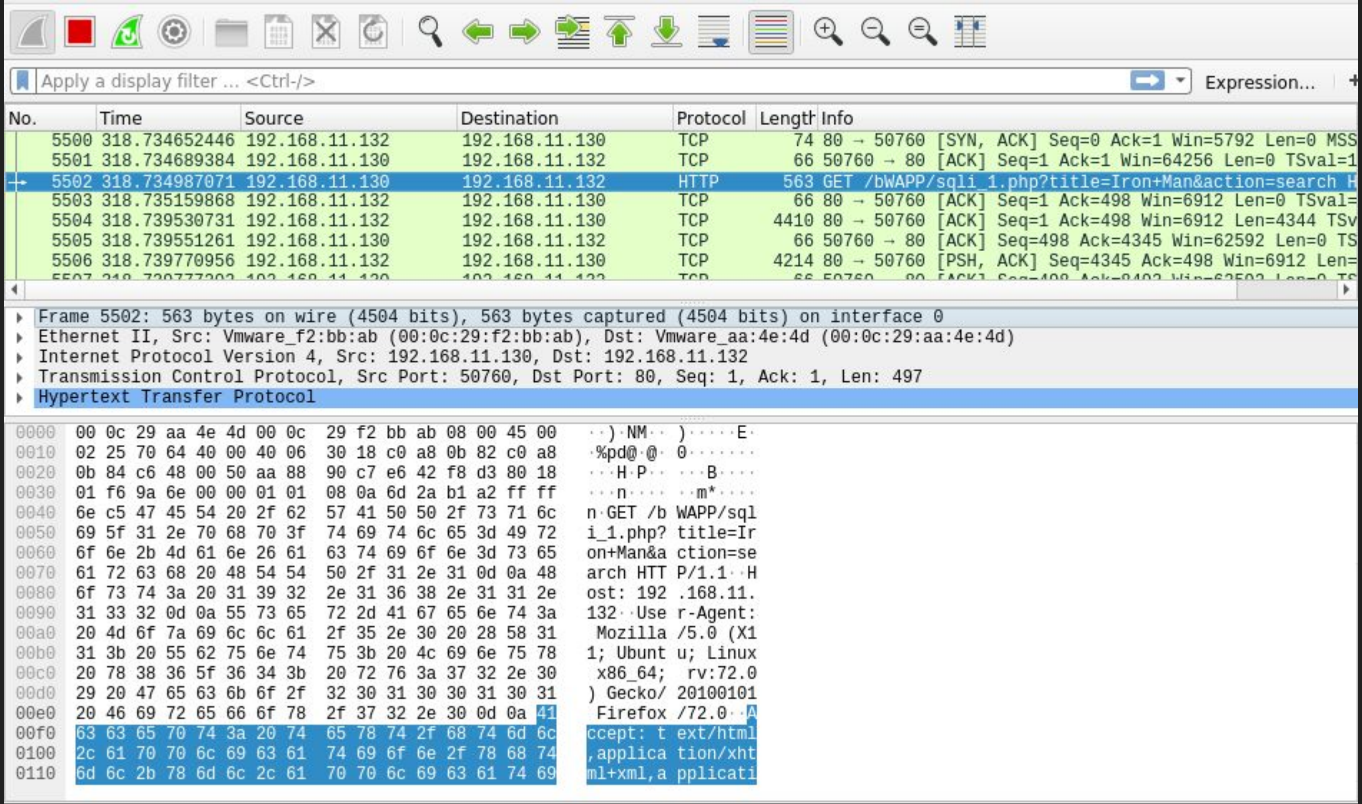
Remember: run it as **SuperUser**!

> Let’s sniff some traffic!

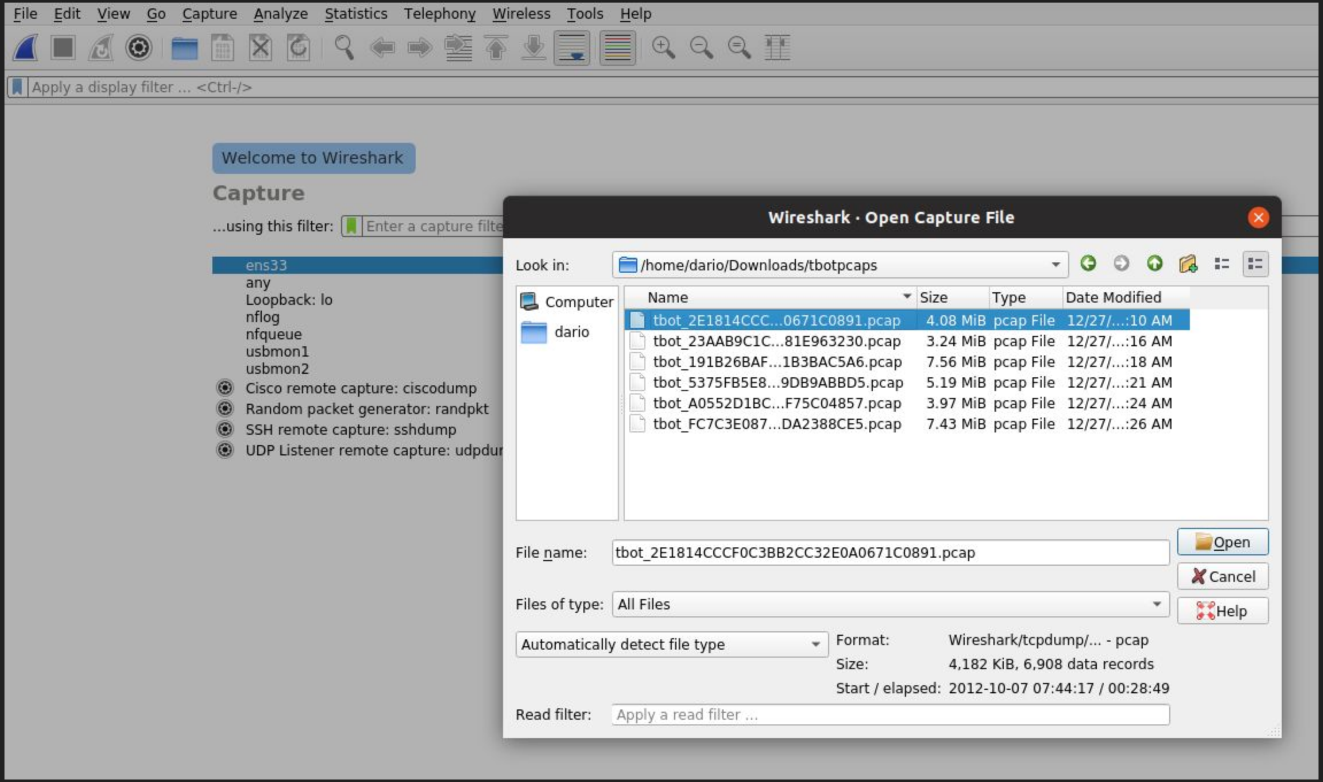


> Let’s sniff some traffic!

Are these familiar to you?



> Wireshark: upload a PCAP file



> Wireshark: apply a filter

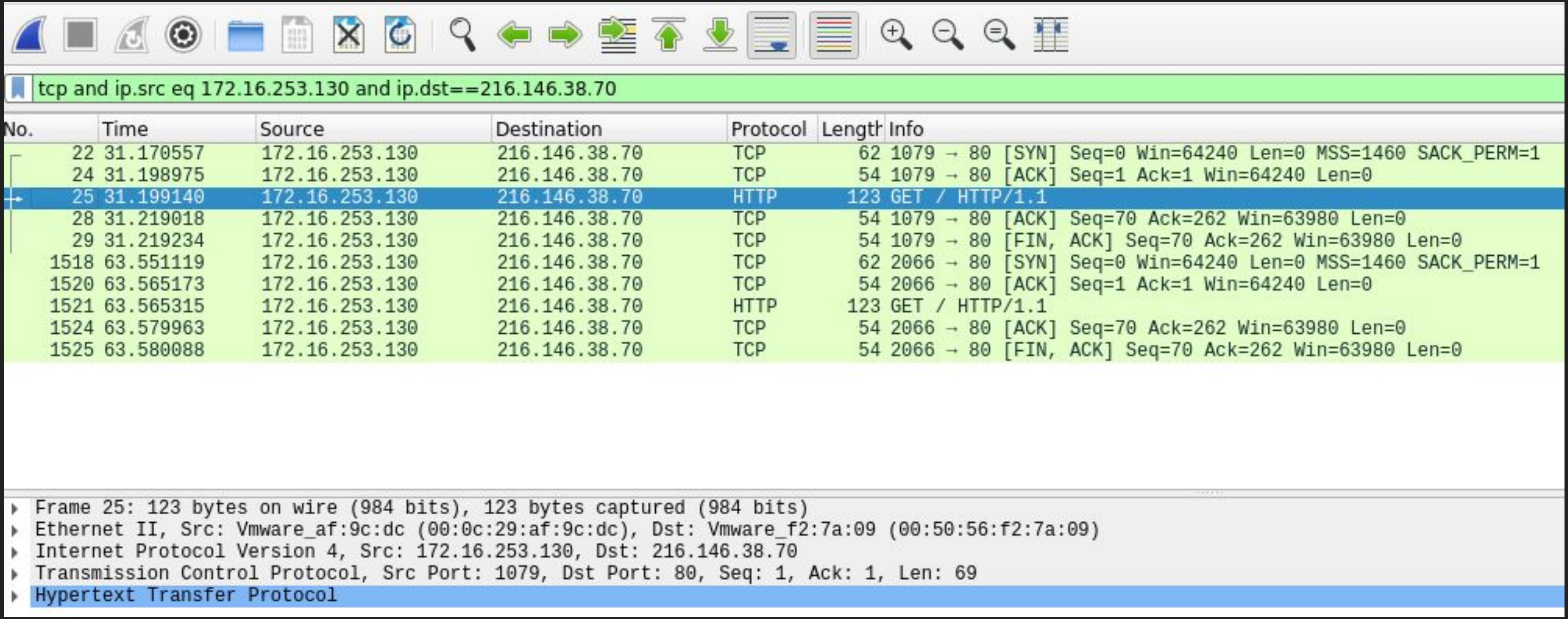
In the Filter tab, it is possible to filter results.

- 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



> PySharkPython 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
```

> PySharк: 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 scapy
```

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 ayyyyyyySCP//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
       scccccp///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 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\x157\xa2D\x00\xae\xf3R\xaa\xd1\x08\x00E\x00\x00C\x00\x01\x00\x00@\x06x<\xc0
 \xa8\x05\x15B#\xfa\x97\x00\x14\x00P\x00\x00\x00\x00\x00\x00\x00\x00P\x02 \x00
 \xbb9\x00\x00GET /index.html HTTP/1.0 \n\n'
>>> 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

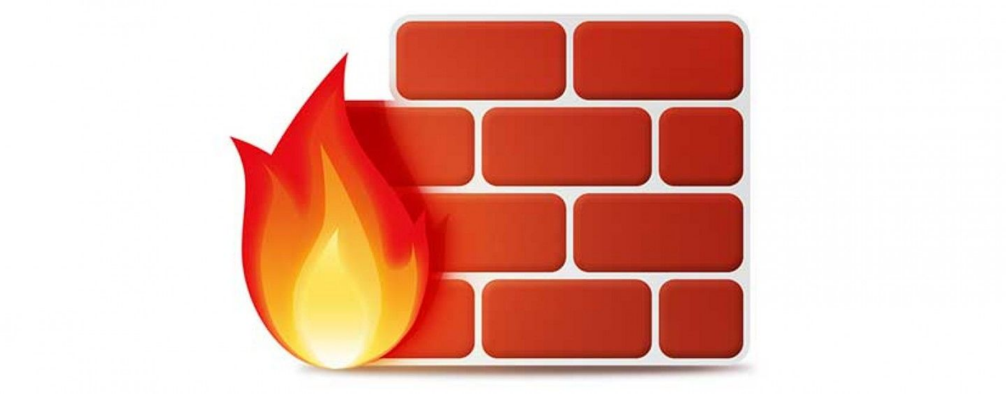
More info about Scapy:

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

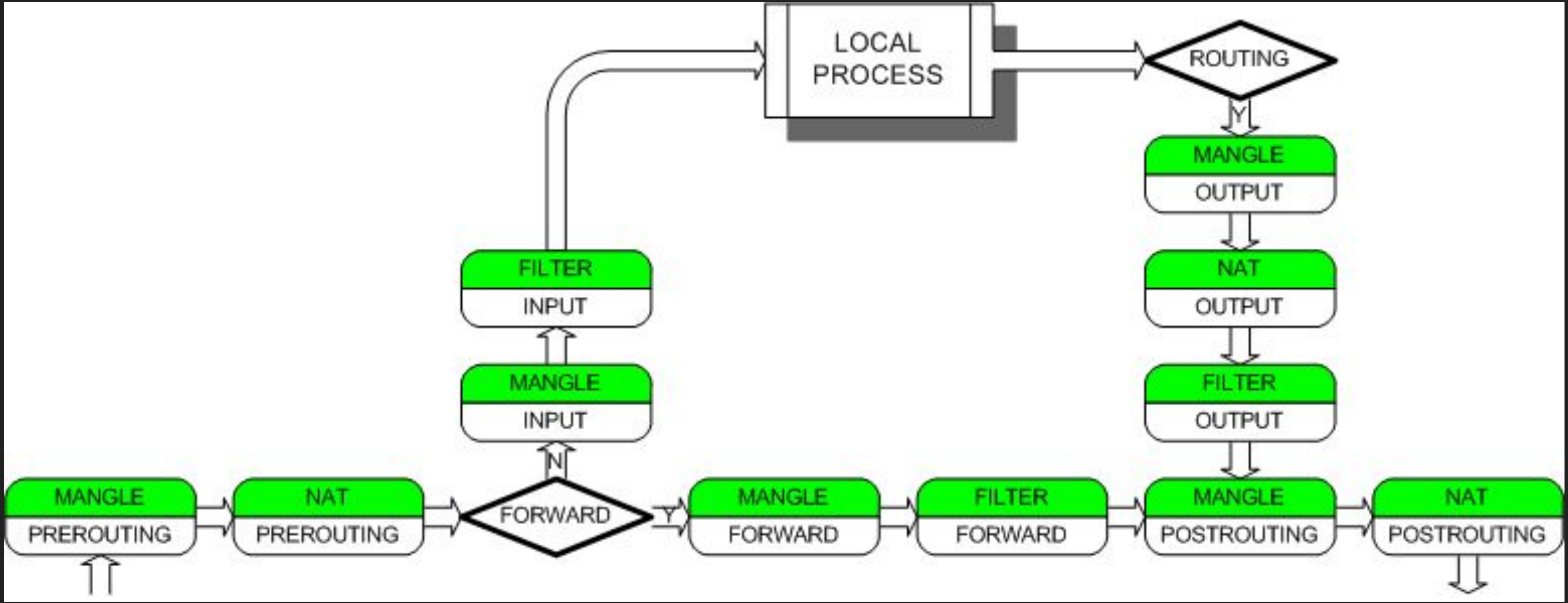
In most cases, you will use the following tables:

- 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



NETWORK

> 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/

SCAN ME

> Nmap

An open source tool for network exploration and security auditing.

Installation:

```
> sudo apt update
> sudo apt install 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

SCAN ME

> 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:

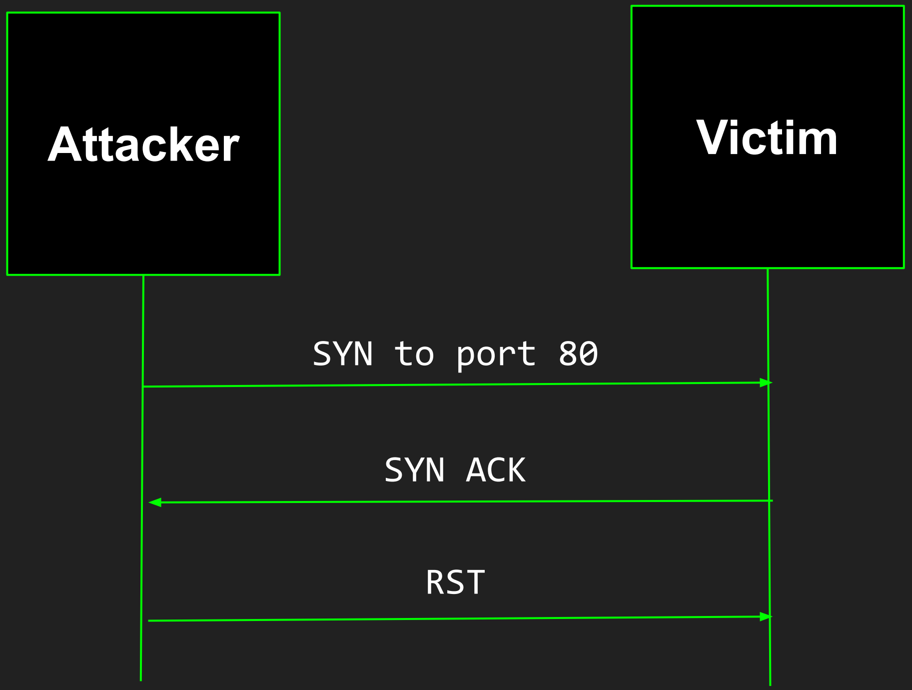
This port is accessible, but no applications are listening on it. It could be interesting repeating the scan in a second time.

- 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)

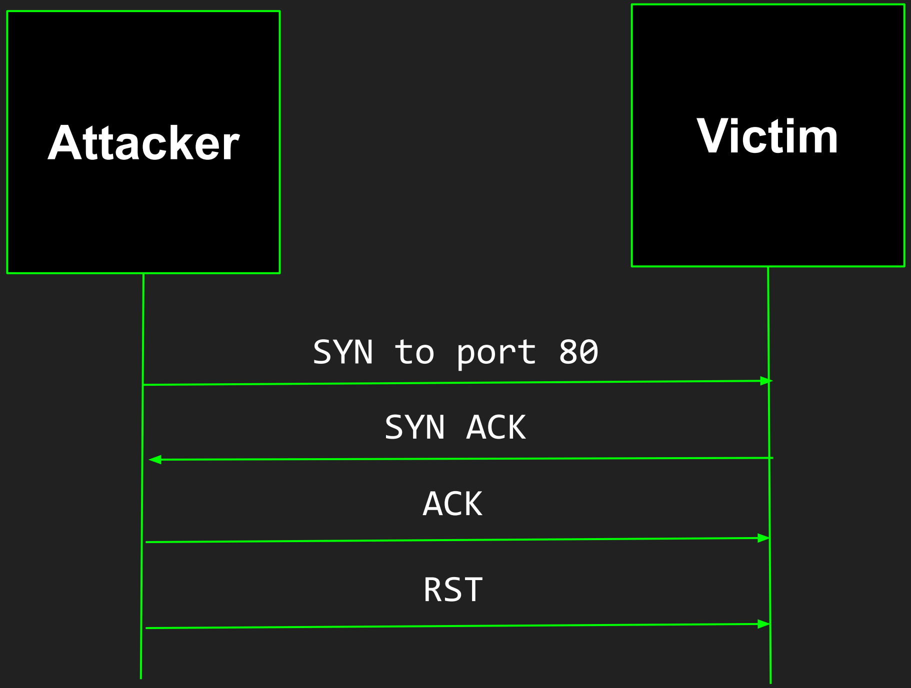
```
> nmap -sS 192.168.1.1
```



- 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)

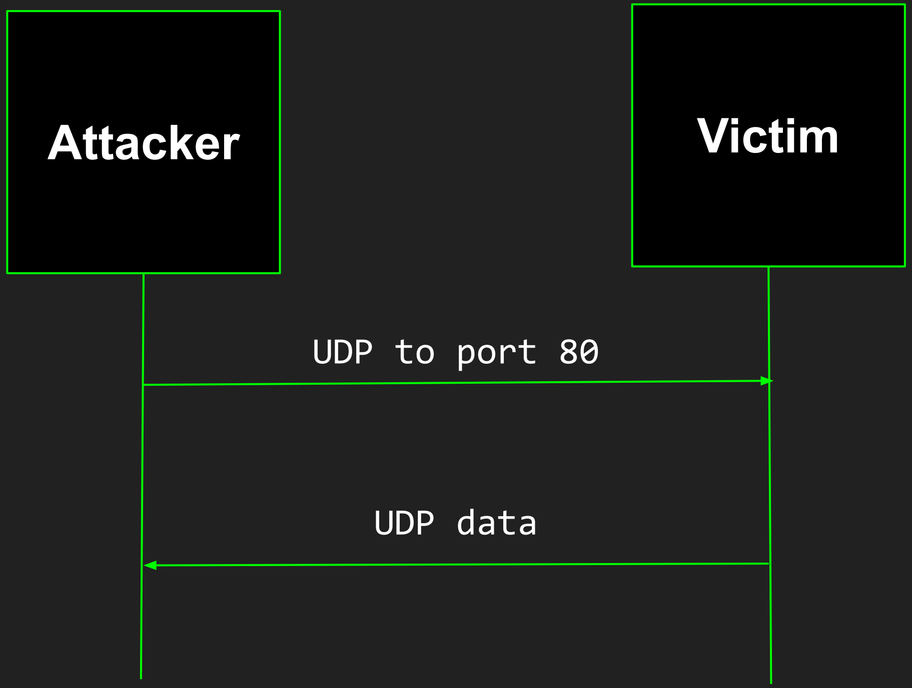
> nmap -sT 192.168.1.1



- 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

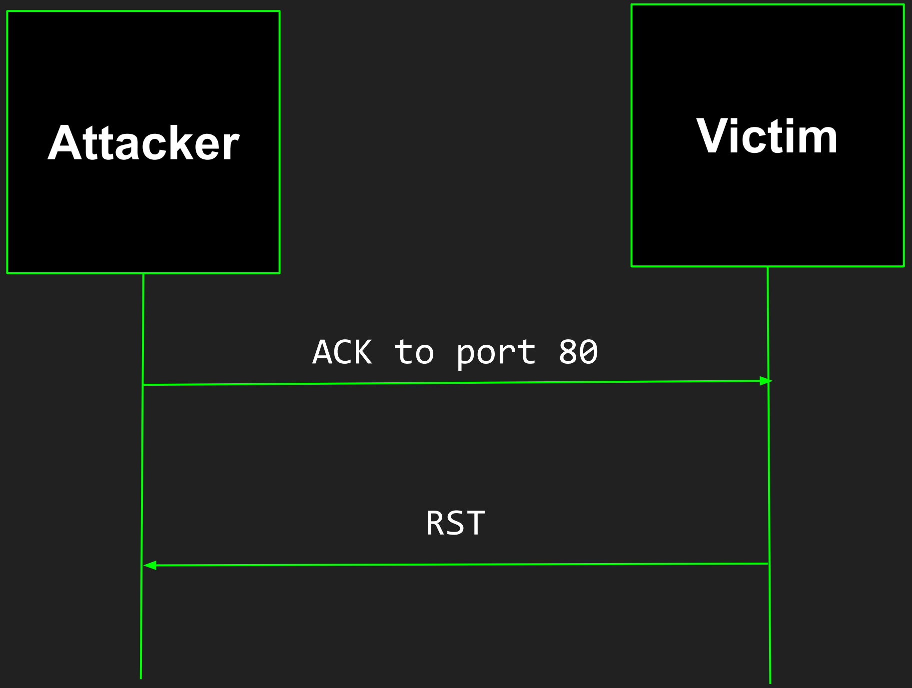
> Scanning techniques (NULL|FIN|Xmas SCAN)

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

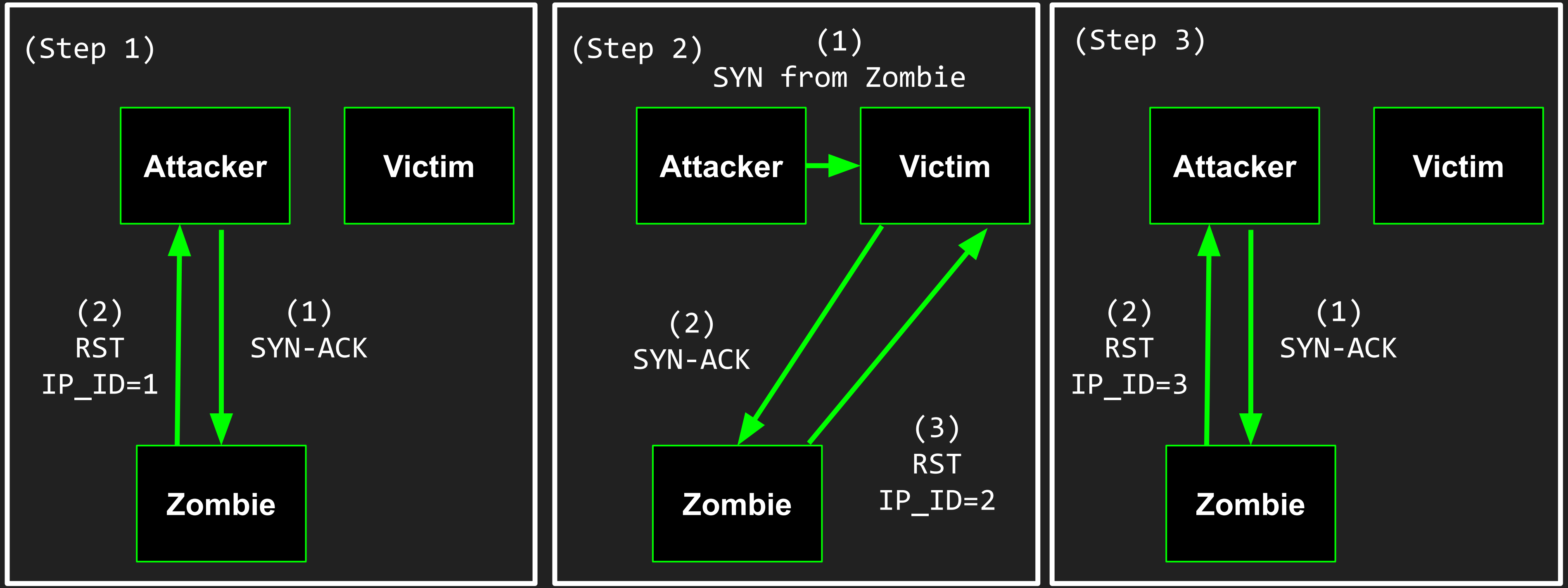
> Scanning techniques (idle SCAN)

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!

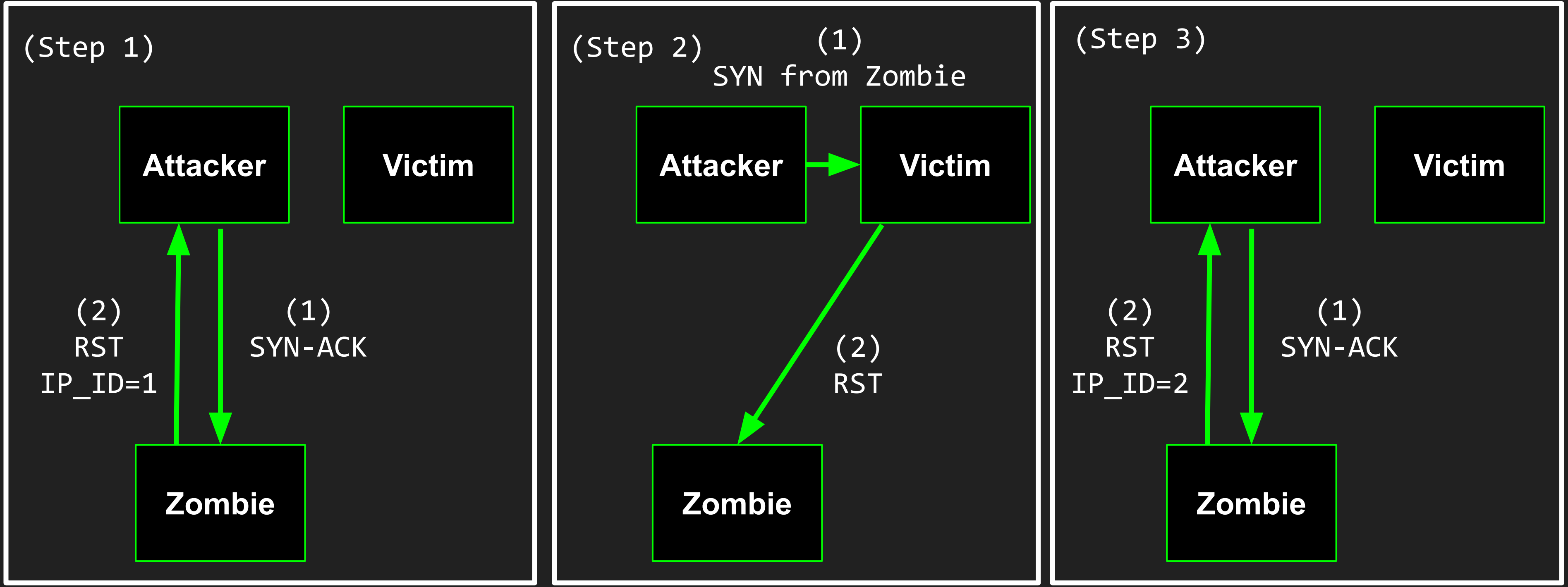
> Scanning techniques (idle SCAN)

Open port:



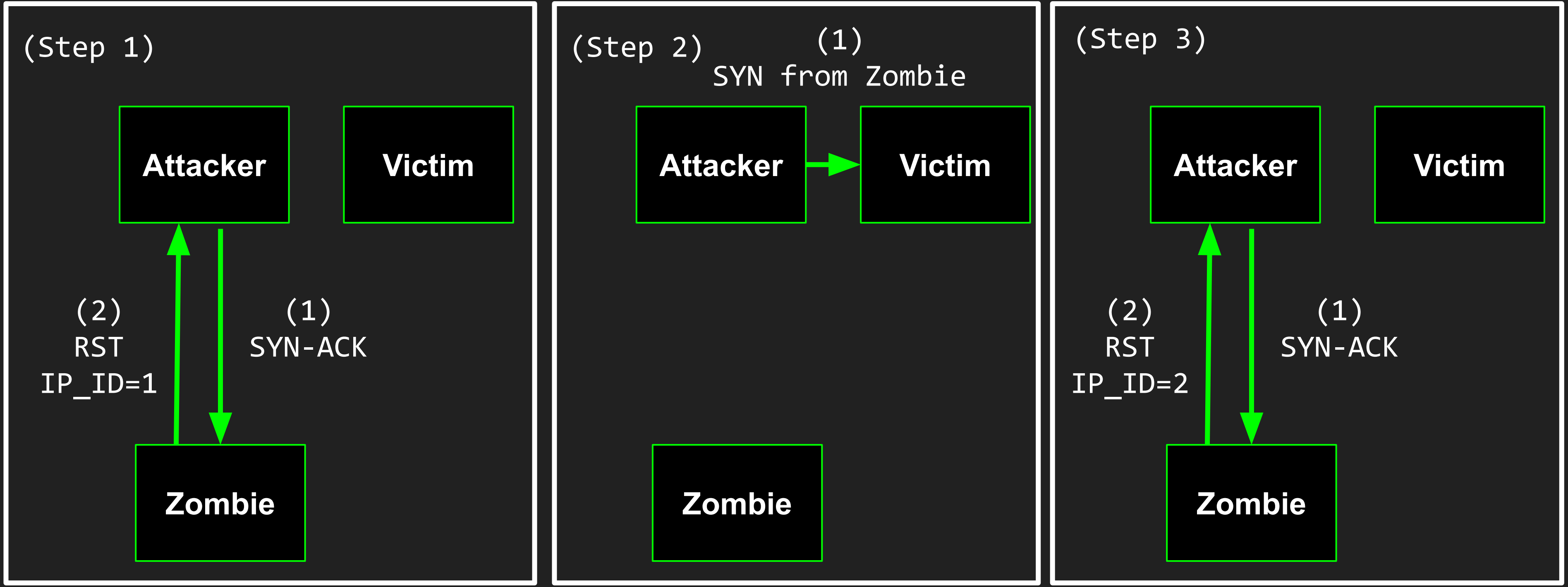
> Scanning techniques (idle SCAN)

Closed port:



> Scanning techniques (idle SCAN)

Filtered port:



> 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 $\underbrace{\texttt{192.168.1.1}}_{\text{Zombie}}$ $\underbrace{\texttt{192.168.1.2}}_{\text{Victim}}$

> 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