

LAB 2

Working with Scapy will allow the system to sniff and manipulate packets on the fly using the python environment.

IP() = will generate a ip packet with default values.

```
from scapy.all import *  
  
a = IP()  
a.show()
```

```
####[ IP ]####  
  version    = 4  
  ihl        = None  
  tos        = 0x0  
  len        = None  
  id         = 1  
  flags      =  
  frag       = 0  
  ttl        = 64  
  proto      = hopopt  
  chksum     = None  
  src        = 127.0.0.1  
  dst        = 127.0.0.1  
  \options   \
```

Target 1: 192.168.85.131

Target 2: 192.168.85.130

Attacker: 192.168.85.132

Task 1.1

Using the script provided in the lab, it sniffed a packet within my network. It showed a packet was sent to 192.168.85.131 from 192.168.85.130.

```
get_pip.py
[02/15/20]seed@VM:~$ sudo python task1_1.py
###[ Ethernet ]###
  dst      = 00:0c:29:d5:5c:8c
  src      = 00:0c:29:3e:90:cb
  type     = 0x800
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 14543
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xd583
  src      = 192.168.85.130
  dst      = 192.168.85.131
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x6897
  id       = 0x1472
  seq      = 0x1
###[ Raw ]###
  load     = '\x16\xc2G^\xd2\x06\x00\x08\t\n\x0b\x0c\r\x
x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1
e\x1f !"#$%&'()*+,-./01234567'
```

When I tried to use it without Super User Do Once (Sudo) It output an error. It could be that the fact it was adding BPF within the kernel which only higher privilege accounts can manipulate data within the kernel.

```

    pkt.show()

pkt = sniff(filter='icmp' , prn=print_pkt)
[02/06/20]seed@VM:~$ sudo python task1_1.py
^C[02/06/20]seed@VM:~$
GNU bash, version 4.3.46(1)-release (i686-pc-linux-gnu)
[02/06/20]seed@VM:~$ nano task1_1.py
[02/06/20]seed@VM:~$ nano task1_1.py
[02/06/20]seed@VM:~$ sudo python task1_1.py
^C[02/06/20]seed@VM:~$ python task1_1.py
Traceback (most recent call last):
  File "task1_1.py", line 8, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/sendrecv.py", line 731, in sniff
    *arg, **karg)] = iface
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/arch/linux.py", line 567, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
  File "/usr/lib/python2.7/socket.py", line 191, in __init__
    _sock = _realsocket(family, type, proto)
socket.error: [Errno 1] Operation not permitted
[02/06/20]seed@VM:~$

```

Task 1.B

If I wanted to filter ICMP with Scapy I can do that using the sniff function and use the BPF syntax, filter="icmp" which will sniff packets that contain ICMP in the ens33 network.

```

GNU nano 2.5.3      File: listenicmp.py

from scapy.all import *

sniff(filter='icmp', count=5, iface='ens33')

```

Which will output this.

```
[02/06/20]seed@VM:~$ sudo python listenicmp.py
Searching ICMP Packets.....
('Source IP:', '192.168.85.131')
('Destination IP:', '192.168.85.131')
('Protocols:', 1)

('Source IP:', '192.168.85.132')
('Destination IP:', '192.168.85.132')
('Protocols:', 1)

('Source IP:', '192.168.85.131')
('Destination IP:', '192.168.85.131')
('Protocols:', 1)

('Source IP:', '192.168.85.132')
('Destination IP:', '192.168.85.132')
('Protocols:', 1)

('Source IP:', '192.168.85.131')
('Destination IP:', '192.168.85.131')
('Protocols:', 1)
```

Same Goes with TCP Packets coming from Port 23.

```
[02/15/20]seed@VM:~$ sudo python listen23tcp.py
Searching TCP Packets.....
('Source IP:', '192.168.85.130')
('Destination IP:', '192.168.85.131')
('Protocols:', 6)

('Source IP:', '192.168.85.131')
('Destination IP:', '192.168.85.130')
('Protocols:', 6)

('Source IP:', '192.168.85.130')
('Destination IP:', '192.168.85.131')
('Protocols:', 6)

('Source IP:', '192.168.85.130')
('Destination IP:', '192.168.85.131')
('Protocols:', 6)

('Source IP:', '192.168.85.131')
('Destination IP:', '192.168.85.130')
('Protocols:', 6)

[02/15/20]seed@VM:~$ █
```

And a subnet from a different network that my router not evening my on my NAT, which is why its waiting for a packet from a different network.

```
[02/06/20]seed@VM:~$ sudo python listensub128.230.py
Searching Subnet
```

Task1.2

With scappy I can allow how many hops a packet can take before it talks back to the source address with the ttl variable.

Here is an example that I set the Time To Live to 1 and make it talk to my router. It seem that the TTL exceeded the amount it take to hope.

ARP	60 Who has 192.168.85.2? Tell 192.168.85.132
ARP	60 192.168.85.2 is at 00:50:56:e6:07:fe
ICMP	60 Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response)
ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)

I was able to get a reply to my router once I increment the TTL to 2.

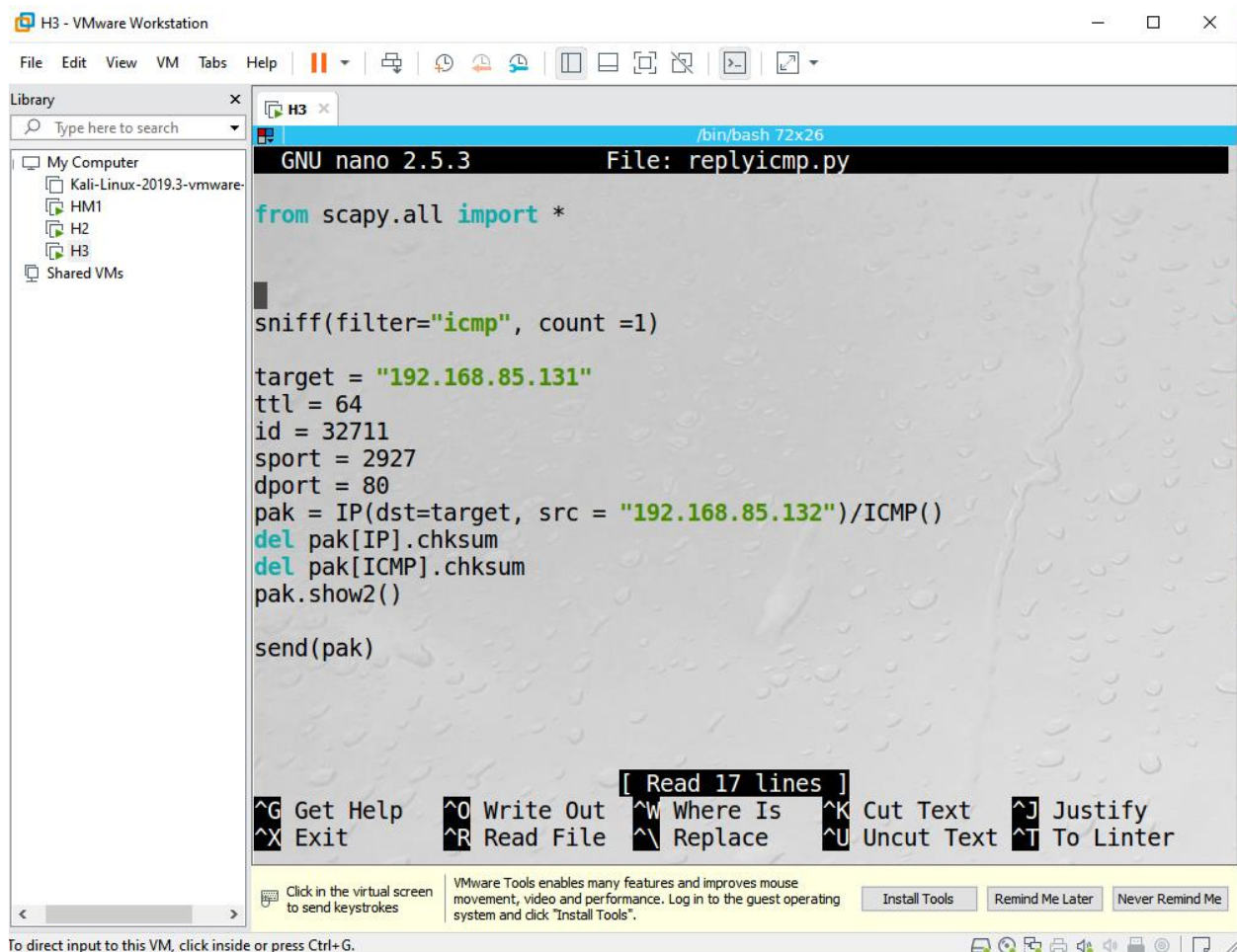
```
a = IP()

a.dst = '192.168.1.1'
a.ttl = 2
b= ICMP()
send(a/b)
```

ARP	60 192.168.85.2 is at 00:50:56:e6:07:fe
ICMP	60 Echo (ping) request id=0x0000, seq=0/0, ttl=2 (reply in 5)
ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=128 (request in ...)

In this scenario, 192.168.85.130 is pinging request to 192.168.85.131 but the attack 192.168.85.132 will intercept a packet and echo back to x.x.x.x.131.

In the code I used the sniff packet to capture 1 packet and manipulate the source address to my address to know it was me and had to delete the chksum so the processor does t not have to validate the checksum. Then after manipulating the packet I would use the send function to send the packet.



The screenshot shows a VMware Workstation window titled "H3 - VMware Workstation". The main window displays a terminal window running the GNU nano 2.5.3 editor, editing a file named "replyicmp.py". The code in the editor is as follows:

```
from scapy.all import *

sniff(filter="icmp", count =1)

target = "192.168.85.131"
ttl = 64
id = 32711
sport = 2927
dport = 80
pak = IP(dst=target, src = "192.168.85.132")/ICMP()
del pak[IP].chksum
del pak[ICMP].chksum
pak.show2()

send(pak)
```

Below the code, there is a status bar with various keyboard shortcuts: **^G** Get Help, **^X** Exit, **^O** Write Out, **^R** Read File, **^W** Where Is, **^N** Replace, **^K** Cut Text, **^U** Uncut Text, **^J** Justify, **^T** To Linter. At the bottom of the window, there is a message: "Click in the virtual screen to send keystrokes" and "VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click 'Install Tools'." There are also buttons for "Install Tools", "Remind Me Later", and "Never Remind Me".

Looking through Wireshark it looks like the attack was successfully because you can see that attacker address send a correct reply packet to the source address.

Capturing from ens33

6:17 PM

icmp

No.	Time	Source	Destination	Protocol
3	2020-02-11 18:17:09.8580416...	192.168.85.130	192.168.85.131	ICMP
4	2020-02-11 18:17:09.8580657...	192.168.85.131	192.168.85.130	ICMP
7	2020-02-11 18:17:09.8651055...	192.168.85.132	192.168.85.131	ICMP
8	2020-02-11 18:17:09.8651211...	192.168.85.131	192.168.85.132	ICMP

Frame 7: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface ens33

Ethernet II, Src: Vmware_9c:1f:16 (00:0c:29:9c:1f:16), Dst: Vmware_d5:5c:8c (00:0c:29:d5:5c:8c)

Internet Protocol Version 4, Src: 192.168.85.132, Dst: 192.168.85.131

Internet Control Message Protocol

Type: 8 (Echo (ping) request)

Code: 0

Checksum: 0xf7ff [correct]

[Checksum Status: Good]

Identifier (BE): 0 (0x0000)

System Settings

0010 d5 5c 8c 00 0c 29 9c 1f 16 08 00 45 00 ..).\\...)....E.

0020 00 1c 00 01 00 00 40 01 4e 88 c0 a8 55 84 c0 a8@. N...U...

0030 55 83 08 00 f7 ff 00 00 00 00 00 00 00 00 00 U... ..

Checksum (icmp.checksum). 2 bytes

Packets: 30 - Displayed: 4 (13.3%) - Profile: Default

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest

Install Tools | Download More Tools | Never Show Again