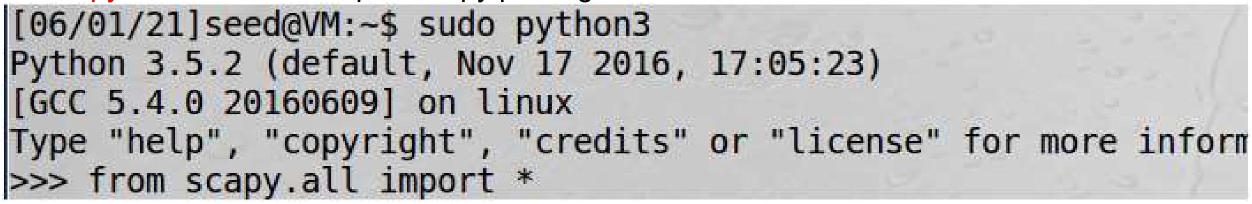
# Lab 4

(Due your class day June 9/14)

## 1. Packet Construction with Scapy

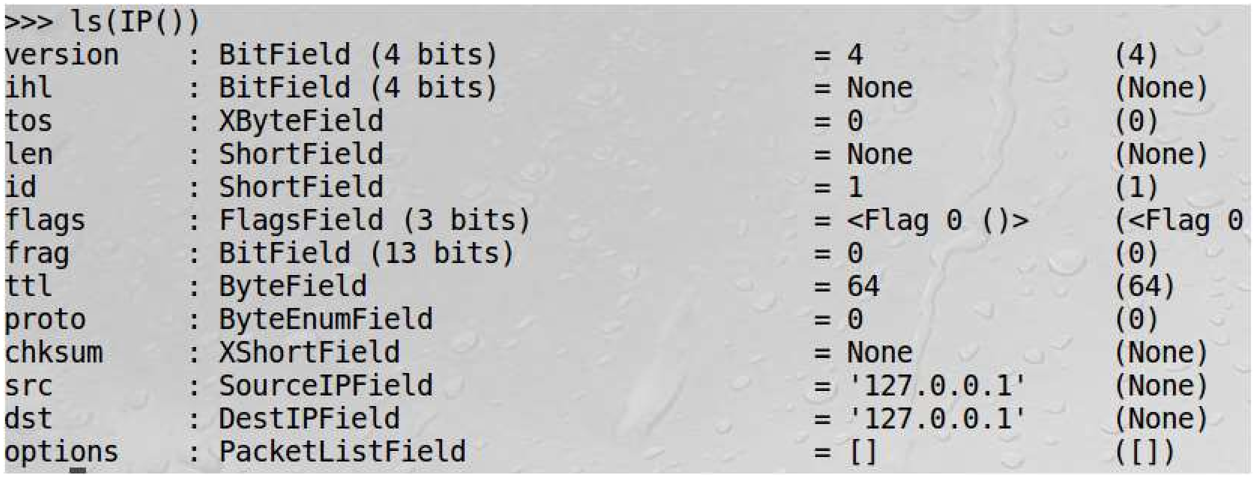
In this exercise, you will practice to construct several packets using scapy. To start, run $sudo python3 and then import scapy package.



```
[06/01/21]seed@VM:~$ sudo python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more inform
>>> from scapy.all import *
```

Then, you are ready to practice constructing various packets. In each question, show your screen shots as the evidence of your work.

1) IP() is the function to construct a default IP header. You can use ls(IP()) the content. The first column is the field of IP header and the third column is the example format for the value of each field.



```
>>> ls(IP())
version    : BitField (4 bits)                  = 4              (4)
ihl        : BitField (4 bits)                  = None           (None)
tos        : XByteField                         = 0              (0)
len        : ShortField                         = None           (None)
id         : ShortField                         = 1              (1)
flags      : FlagsField (3 bits)                = <Flag 0 ()>    (<Flag 0
frag       : BitField (13 bits)                 = 0              (0)
ttl        : ByteField                          = 64             (64)
proto      : ByteEnumField                      = 0              (0)
chksum     : XShortField                        = None           (None)
src        : SourceIPField                      = '127.0.0.1'    (None)
dst        : DestIPField                        = '127.0.0.1'    (None)
options    : PacketListField                    = []             ([])
```

You can assign the value to create the IP header you want. For example, to create a header with destination 8.8.8.8, use $ IP(dst=”8.8.8.8”). Please construct an ip header **iph** with source 10.0.2.4 and destination www.mit.edu. Use ls to show packet header.

2) ICMP() will construct the default ICMP packet, which is a ping request packet. UDP() is the default UDP header. Create a UDP header with source port number 5000 and destination port number 53. Use ls to show your result and also the default ICMP().

**3)** You can create ping packet by stacking IP header over ICMP(). Create a ping packet to 8.8.8.8. Similarly, create an UDP packet with destination port 53 and source ip 10.0.2.4 and destination ip 8.8.8.8. Use show2() function to show the packet content.

4) For a packet pkt, pkt[IP] is IP datagram and pkt[UDP] is the UDP segment of pkt. Show the UDP segment and ICMP content for the packets in 3), using show2() function.

## 2. Sniffing Packets

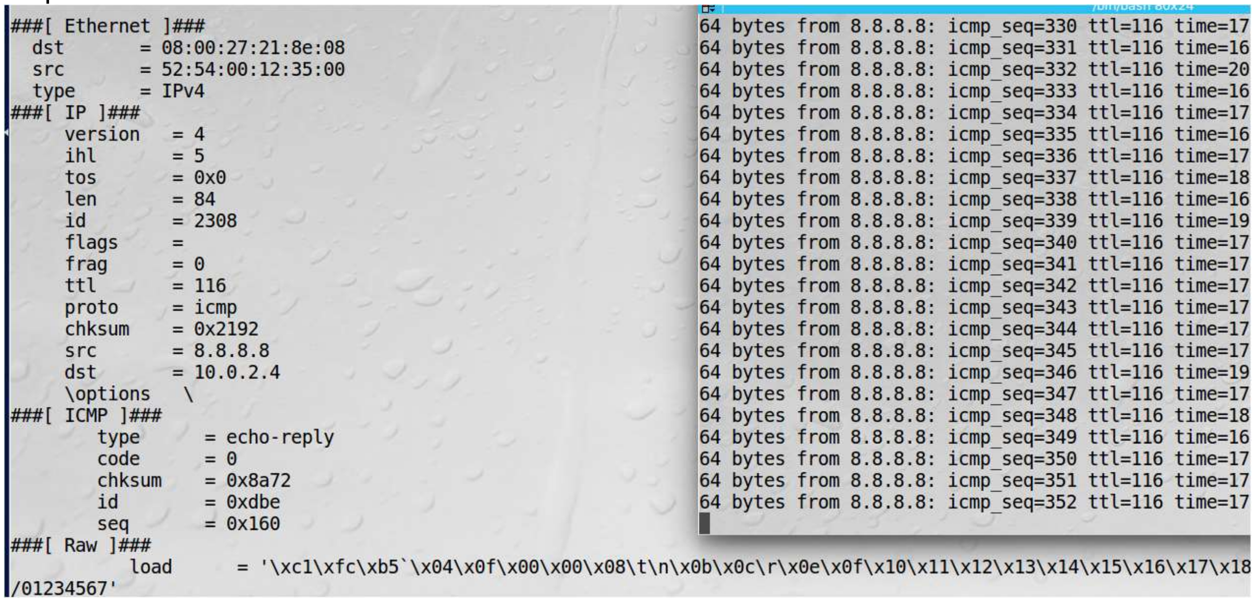
Wireshark is the most popular sniffing tool, and it is easy to use. We will use it throughout the entire lab. However, it is difficult to use Wireshark as a building block to construct other tools. We will use Scapy for that purpose. The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs. A sample code is provided in the following:

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

def  print_pkt(pkt):
      pkt.show2()

pkt = sniff(filter='icmp',prn=print_pkt)
----------------------------------------------------------
```

**Task A**. The above program sniffs packets. For each captured packet, the callback function print pkt() will be invoked; this function will print out some of the information about the packet. Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

```
// Run the program with the root privilege
$ sudo python sniffer.py

// Run the program without the root privilege
$ python sniffer.py
```

**Task B**. Usually, when we sniff packets, we are only interested certain types of packets. Please change the print_pkt function so that only the packet with source IP address 8.8.8.8 will be shown. The ip address of packet pkt can be accessed using pkt[IP].src. After you change your program, run it and then $ ping 8.8.8.8 from another terminal. The following is my sample output.



```
###[ Ethernet ]###
  dst       = 08:00:27:21:8e:08
  src       = 52:54:00:12:35:00
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 2308
     flags     =
     frag      = 0
     ttl       = 116
     proto     = icmp
     chksum    = 0x2192
     src       = 8.8.8.8
     dst       = 10.0.2.4
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x8a72
        id        = 0xdbe
        seq       = 0x160
###[ Raw ]###
           load      = '\xc1\xfc\xb5`\x04\x0f\x00\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18
/01234567'
```

```
64 bytes from 8.8.8.8: icmp_seq=330 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=331 ttl=116 time=16
64 bytes from 8.8.8.8: icmp_seq=332 ttl=116 time=20
64 bytes from 8.8.8.8: icmp_seq=333 ttl=116 time=16
64 bytes from 8.8.8.8: icmp_seq=334 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=335 ttl=116 time=16
64 bytes from 8.8.8.8: icmp_seq=336 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=337 ttl=116 time=18
64 bytes from 8.8.8.8: icmp_seq=338 ttl=116 time=16
64 bytes from 8.8.8.8: icmp_seq=339 ttl=116 time=19
64 bytes from 8.8.8.8: icmp_seq=340 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=341 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=342 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=343 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=344 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=345 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=346 ttl=116 time=19
64 bytes from 8.8.8.8: icmp_seq=347 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=348 ttl=116 time=18
64 bytes from 8.8.8.8: icmp_seq=349 ttl=116 time=16
64 bytes from 8.8.8.8: icmp_seq=350 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=351 ttl=116 time=17
64 bytes from 8.8.8.8: icmp_seq=352 ttl=116 time=17
```

## 3. Sniff function with BPF Filters.

In this problem, you will practice sniff function with BPF filter. You will use BPF filter rules (see reference file BPF.pdf if necessary) to write filter expressions for each of the following restrictions.

a) Capture only the ICMP packet and destination IP 8.8.8.8. To verify whether your filter is correct, apply your filter to the sample code in problem 2 and ping 8.8.8.8.

b) (**optional**) Capture any TCP packet that comes from a particular IP and with a destination port number 23. To verify your filter, apply your filter to the sample code in problem 2 and telnet to this particular IP (it is ok if you telnet yourself).

c) (**optional**) Capture packets that come from source port 53 and a **particular** destination ip . Apply your filter to the sample code in problem 2 and run $dig @8.8.8.8 www.mit.edu and see what is captured.

## 4. Spoofing ICMP Packets

As a packet spoofing tool, Scapy allows us to set the fields of IP packets to arbitrary values. The objective of this task is to spoof IP packets with an arbitrary source IP address. Please construct an ICMP echo request packets, and send them to 8.8.8.8. Then, use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address. The following code shows an example of how to spoof an ICMP packets. Show the screen shot of one reply packet from 8.8.8.8.

```
-----------------------------------------------------------------
>>> from scapy.all import *
>>> a = IP(dst=”10.0.2.3”)
>>> b = ICMP()
>>> p = a/b
>>> send(p)
------------------------------------------------------------------
```

## 5. Sniffing-then-Spoofing

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-andthen-spoof program. You need two VMs on the same LAN (if you have only one VM, use two terminals). From VM A, you ping IP 8.8.8.8. This will generate an ICMP echo request packet. Do the following.

- Construct a sniff-and-then-spoof program and run on VM B to monitor the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique.
- Since 8.8.8.8 is alive, the victim should receive two responses for each request that have the same **id** and the same **seq**. Run Wireshark to observe two such responses and find out the forged response sent by your program. To do this, you can check the MAC in two packets. The forged packet should have the MAC of VM B.