James Oswald
ICSI 424 Computer Security
Lab 07

Lab Task Set 1: Using Tools to Sniff and Spoof Packets:

I begin by installing scapy for sniping and spoofing packets with python.

```
[11/09/20 JO481765]seed@VM:~$ sudo pip3 install scapy
The directory '/home/seed/.cache/pip/http' or its paren
```

...

```
    99% |                                | 1.0MB 1.0MB/
    100% |                               | 1.0MB 1.0MB
/s
Installing collected packages: scapy
  Running setup.py install for scapy ... done
Successfully installed scapy-2.4.4
You are using pip version 18.1, however version 20.3b1
is available.
You should consider upgrading via the 'pip install --up
grade pip' command.
[11/09/20 JO481765]seed@VM:~$
```

I then test it out on command line just to make sure it's working properly:

```
[11/09/20 JO481765]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 mo
re information.
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
```

Task 1.1: Sniffing Packets:

I begin by writing the sniffer program into python. The filter is set to ICMP so we will only capture ICMP packets.

sniffer.py

```
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

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

Task 1.1A.

I begin by making the sniffer executable then running it as root.

```
[11/09/20 JO481765]seed@VM:~/lab08$ chmod a+x sniffer.p
y
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:bc:55:08
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
```

Inorder to actually demonstrate it can capture packets, i need to generate ICMP packets for it to capture, which I do by opening a new terminal and using the ping utility which operates on ICMP

```
[11/09/20 JO481765]seed@VM:~$ ping google.com
PING google.com (172.217.12.142) 56(84) bytes of data.
64 bytes from lga34s19-in-f14.1e100.net (172.217.12.142
): icmp_seq=1 ttl=112 time=6.31 ms
64 bytes from lga34s19-in-f14.1e100.net (172.217.12.142
): icmp_seq=2 ttl=112 time=6.21 ms
64 bytes from lga34s19-in-f14.1e100.net (172.217.12.142
): icmp_seq=3 ttl=112 time=6.02 ms
```

Which as we can see, clearly demonstrates that we can capture these packets in the sniffer program from the output:

```
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xf18c
        id        = 0xffb
        seq       = 0x4a
###[ Raw ]###
           load      = '\'\x9c\xa9_;/\x07\x00\x08\t\n\x
0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x1
9\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Next I observe what happens when I try to run the program not as root but as a normal user. I observe that it immediately fails with a permission error. This is expected since being able to sniff all packets coming into the system as a normal user would be a massive security vulnerability.

```
[11/09/20 JO481765]seed@VM:~/lab08$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 8, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/se
ndrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.5/dist-packages/scapy/se
ndrecv.py", line 907, in _run
    *arg, **karg)] = iface
  File "/usr/local/lib/python3.5/dist-packages/scapy/ar
ch/linux.py", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.S
OCK_RAW, socket.htons(type))  # noqa: E501
```

```
  File "/usr/lib/python3.5/socket.py", line 134, in __i
nit__
    _socket.socket.__init__(self, family, type, proto,
fileno)
PermissionError: [Errno 1] Operation not permitted
[11/09/20 JO481765]seed@VM:~/lab08$
```

Task 1.1B.

For the first filter the program is unchanged, it was already configured and demonstrated to capture exclusively ICMP packets in the last task.

For the second filter task, I change the filter line to:

```
pkt = sniff(filter='tcp and dst port 23 and src host 10.0.2.15', prn=p
```

Which is designed to read outgoing tcp packets from my local IP address on port 23 which is for Telnet traffic.

I then run the program with the new filter:

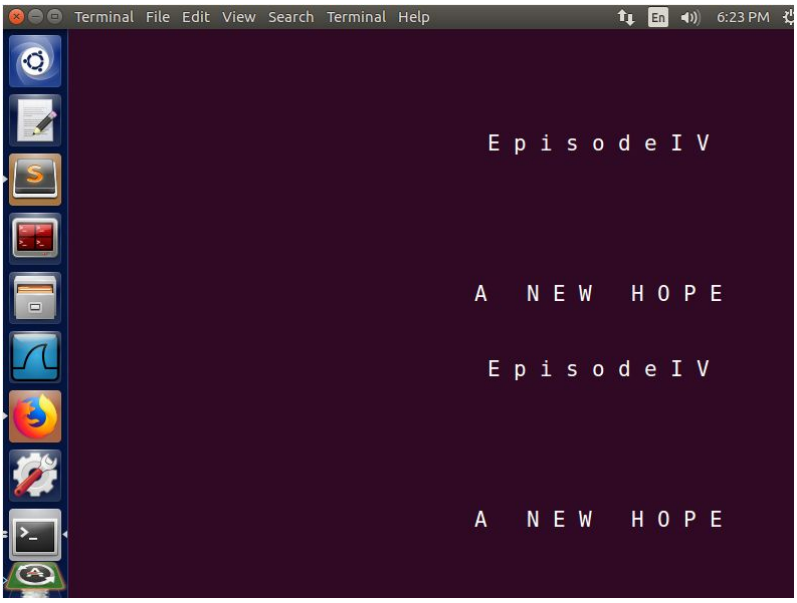
```
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./sniffer.py
```

I test it out using a famous telnet address in another terminal window that plays the entire first star wars movie in the terminal

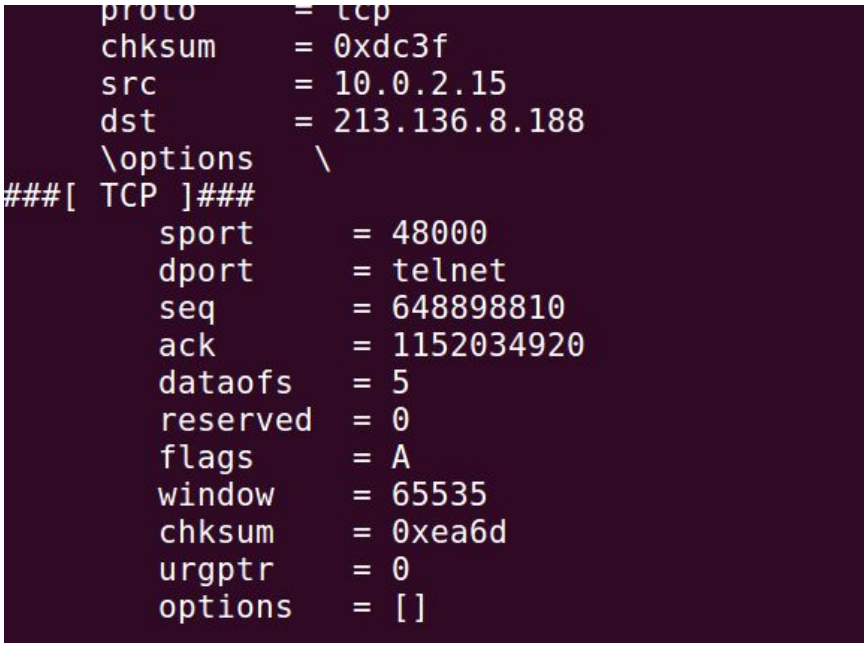
`telnet towel.blinkenlights.nl`

Unfortunately, the VM's weird display size messes it up, but nevertheless we see that we got the desired output in the terminal running the sniffing program.

Telnet output:



Section of one of the TCP packets in the sniffer output:



Finally we change our filter to capture packets to or from a subnet

```
pkt = sniff(filter='dst net 10.0.0.0/16', prn=print_pkt)
```

```
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./sniffer.py
###[ Ethernet ]###
```

We can see that traffic destination is all on the given subnet:

```
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 89
     id        = 28088
     flags     =
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = 0x54b8
     src       = 172.20.0.1
     dst       = 10.0.2.15
     \options   \
###[ UDP ]###
        sport     = domain
        dport     = 15607
        len       = 69
        chksum    = 0x756e
###[ DNS ]###
```

Task 1.2: Spoofing ICMP Packets

I rewrite the code to allow me to pass in an arbitrary source IP from command line

sniffer.py | Task1.2.py

```
1  #!/usr/bin/python3
2
3  import sys
4  from scapy.all import *
5
6  a = IP()
7  a.dst = "8.8.8.8"
8  a.src = sys.argv[1] #get an arbitrary source from command line
9  b = ICMP()
10 p = a/b      #add b as the IP payload
11 p.show()     #Display packet info showing our arbitrary source
12 send(p)      #Send the packet
```

I configure the program to echo to 8.8.8.8, one of google's DNS servers so that the outgoing request will be recorded by wireshark as well once the packet has been sent.

```
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./Task1.2.py 0
.0.0.0
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 0.0.0.0
  dst       = 8.8.8.8
  \options   \
###[ ICMP ]###
     type      = echo-request
```

The result displays that we have successfully faked our src address and a look at wireshark also confirms this.

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-…	PcsCompu_bc:55:08	Broadcast	ARP	42	Who has 10.0.2.2? T…
2	2020-…	RealtekU_12:35:02	PcsCompu…	ARP	60	10.0.2.2 is at 52:5…
3	2020-…	0.0.0.0	8.8.8.8	ICMP	42	Echo (ping) request…
4	2020-…	RealtekU_12:35:02	Broadcast	ARP	60	Who has 0.0.0.0? Te…

Task 1.3: Traceroute

After an hour or so I developed this traceroute program by making user of the ICMP type field and scapy's sr1 (send receive first response) function.

sniffer.py | Task1.2.py | Task1.3.py

```
#!/usr/bin/python3

import sys
from scapy.all import *

print("Traceroute to " + sys.argv[1])
counter = 1
while True:
    a = IP()
    a.dst = sys.argv[1] #desination
    a.ttl = counter
    result = sr1(a/ICMP(), verbose=False)   #send packet
    if result.type != 11:       #11 is a time to live exceeded error code
        if result.type != 0:    #0 is the echo responce code
            raise "Unexpected ICMP responce code: " + str(result.type)
        print("Recived echo responce from " + result.src)
        print("Traceroute passed through " + str(counter - 1) + " routers"
        break
    print("Router #" + str(counter) + ": " + result.src)
    counter = counter + 1
```

Using this we can obtain the number of routers we pass through and the addresses of those routers.

Running the program to check how many routers I pass through on my way to google's 8.8.8.8 DNS, but this would work for any arbitrary IP to traceroute to.

```
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./Task1.3.py 8
.8.8.8
Traceroute to 8.8.8.8
Router #1: 10.0.2.2
Router #2: 172.20.0.1
Router #3: 38.74.72.1
Router #4: 10.9.0.50
Router #5: 38.104.52.185
Router #6: 154.54.40.62
Router #7: 154.54.47.218
Router #8: 154.54.12.18
Router #9: 66.110.96.5
Router #10: 72.14.195.232
Router #11: 108.170.248.97
Router #12: 172.253.72.129
Recived echo responce from 8.8.8.8
Traceroute passed through 12 routers
[11/09/20 JO481765]seed@VM:~/lab08$
```

Thus I have successfully developed a traceroute utility.

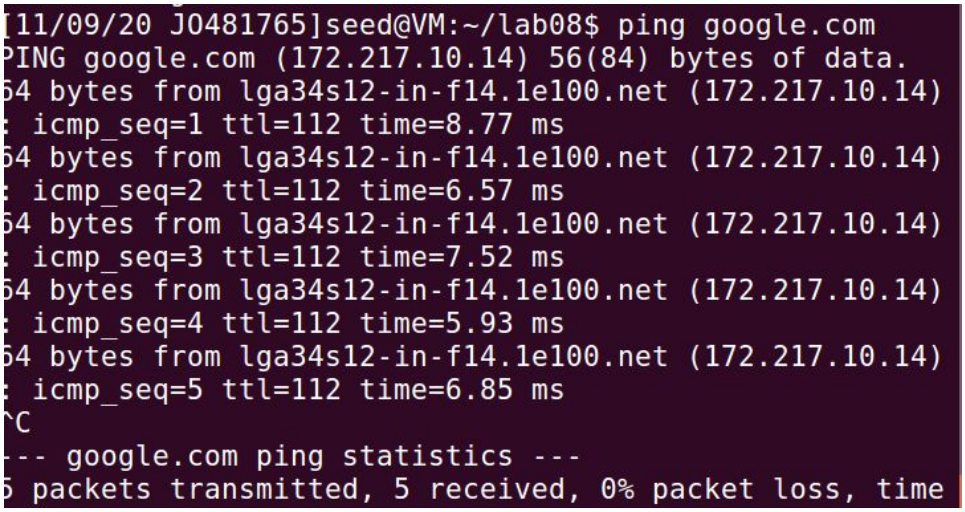
Task 1.4: Sniffing and-then Spoofing

On my first VM I write the sniffing and spoofing program:

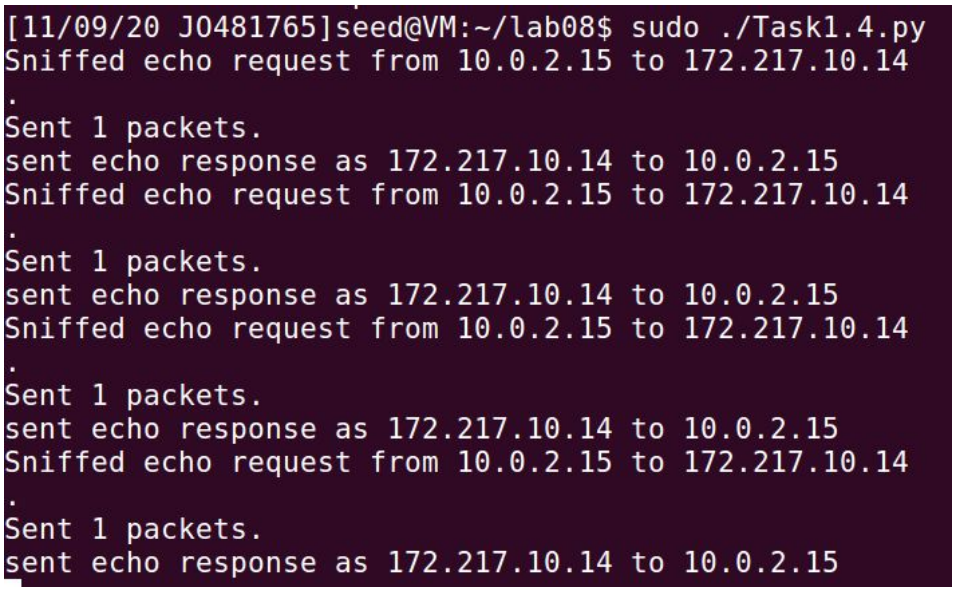
```
#!/usr/bin/python3

from scapy.all import *

def spoof(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Sniffed echo request from " + pkt[IP].src + " to " + pkt[I
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        newpkt = ip/icmp
        send(newpkt)
        print("sent echo response as " + newpkt[IP].src + " to " + newpkt

sniff(filter="icmp and src host 10.0.2.15", prn=spoof)
```

This program works by copying the src and pretending to be the intended destination, even if it doesn't exist.

On my second VM I ping google.com to create echo requests that can be received and spoofed by the above program



As we can see on the VM running the spoofer, these were actually all spoofed by me after sniffing them.



Thus I have successfully sniffed and spoofed packets using python.

Lab Task Set 2: Writing Programs to Sniff and Spoof Packets
Task 2.1: Writing Packet Sniffing Program
Task 2.1A: Understanding How a Sniffer Works

I begin by writing the C program and filling in the correct parameters for my machine

sniff.c

```c
#include <pcap.h>
#include <stdio.h>

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_cl
   printf("Got a packet\n");
}

int main(){
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "ip proto icmp";
  bpf_u_int32 net;
  handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
  pcap_compile(handle, &fp, filter_exp, 0, net);
  pcap_setfilter(handle, &fp);
  pcap_loop(handle, -1, got_packet, NULL);
  pcap_close(handle);   //Close the handle
  return 0;
}
```

I then compile and run it, and use ping to ensure it works as intended:
Compiling and running:

```
[11/09/20 JO481765]seed@VM:~/lab08$ gcc -o sniff sniff.
c -lpcap
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./sniff
```

Pinging to generate packets to sniff in another terminal:

```
[11/09/20 JO481765]seed@VM:~/lab08$ ping google.com
PING google.com (172.217.7.14) 56(84) bytes of data.
64 bytes from lga25s56-in-f14.1e100.net (172.217.7.14):
 icmp_seq=1 ttl=113 time=6.65 ms
64 bytes from lga25s56-in-f14.1e100.net (172.217.7.14):
 icmp_seq=2 ttl=113 time=6.68 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time
 1001ms
rtt min/avg/max/mdev = 6.653/6.668/6.683/0.015 ms
[11/09/20 JO481765]seed@VM:~/lab08$
```

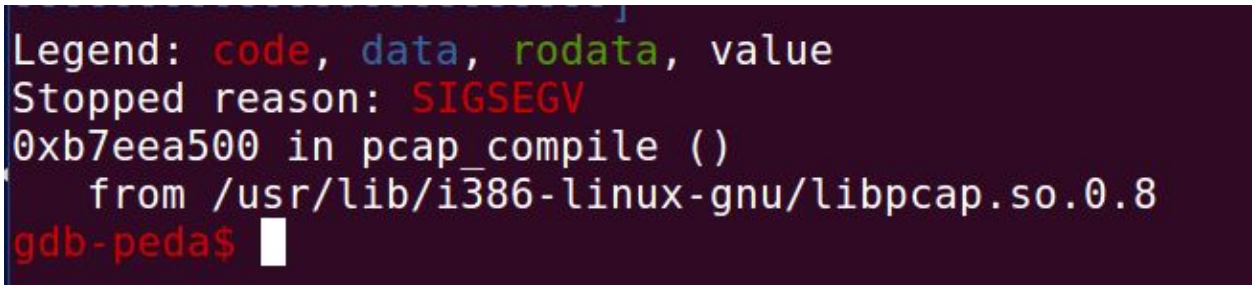
Receiving notification of successfully sniffing the packets in the main terminal:

Question 1

The pcap library uses a variety of library calls to function. The first library call, pcap_open_live opens up a packet capture handle for a network based on the network name specified. Next pcap_compile and pcap_setfilter are used to compile and set a filter to use. Finally pcap_loop is used to hand control over to pcap, a callback is passed to hand us back control to process whatever packet is received. Finally pcap_close is used to close the handle, but this only occurs if the loop stops.

Question 2

You need root privilege to run a sniffer because being able to sniff packets from users on the same computer is dangerous, the code fails on pcap_compile, I assume this is due to the fact that if we aware only sniffing our own traffic, this would be safe, but pcap sees we want to sniff more than our own traffic and stopps us.



Question 3

In order to demonstrate the effects of promiscuous mode, the VM network needs to be in bridged mode, actually enabling it is done via the third param to pcap_open_live. Once its on I would need to rewrite the program to display what packets were from where but it would show that it receives packets from everything on the network, rather than just packets intended for the current computer.

Task 2.1B: Writing Filters

I rewrite the sniffing program to exclusively capture the ICMP packets between me and linux.die.net at 172.67.69.187

```
struct bpf_program fp;
char filter_exp[] = "icmp and (host 172.67.69.187 and 10.0.2.15)";
bpf_u_int32 net;
```

To test this I run the program and then ping linux.die.net to generate ICMP packets between us

```
[11/09/20 JO481765]seed@VM:~/lab08$ ping linux.die.net
PING linux.die.net (172.67.69.187) 56(84) bytes of data
.
64 bytes from 172.67.69.187: icmp_seq=1 ttl=54 time=6.7
5 ms
64 bytes from 172.67.69.187: icmp_seq=2 ttl=54 time=6.5
3 ms
64 bytes from 172.67.69.187: icmp_seq=3 ttl=54 time=6.2
0 ms
^C64 bytes from 172.67.69.187: icmp_seq=4 ttl=54 time=6
.46 ms
```

Sure enough, after recompiling, the new sniff program works:

```
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./sniff
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
```

However, this doesn't display if i ping anyone else, like google for example:

```
[11/09/20 JO481765]seed@VM:~/lab08$ ping google.com
PING google.com (172.217.7.14) 56(84) bytes of data.
64 bytes from lga25s56-in-f14.1e100.net (172.217.7.14):
 icmp_seq=1 ttl=113 time=8.43 ms
64 bytes from lga25s56-in-f14.1e100.net (172.217.7.14):
 icmp_seq=2 ttl=113 time=14.2 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time
 1001ms
```

No packets because the filter only allows packets between me and linux.die.net

```
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./sniff
```

Next I test for filtering TCP with a portrange, to do this I edit my filter to:

```
//char filter_exp[] = "icmp and (host 172.67.69.187 a
char filter_exp[] = "tcp and dst portrange 10-100";
bpf_u_int32 net;
```

I recompile and run the program

```
[11/09/20 JO481765]seed@VM:~/lab08$ gcc -g -o sniff sni
ff.c -lpcap
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./sniff
Got a packet
Got a packet
Got a packet
Got a packet
```

I open firefox and immediately start receiving notifications. This is likely due to the fact that HTTP operates over port 80 which is in the 10-100 range.

Task 2.1C: Sniffing Passwords

I begin by sniffing for TCP on port 23 which is for telnet connections

```
//char filter_exp[] = "tcp and dst p
char filter_exp[] = "tcp port 23";
```

From here I modify the program to display the sniffed data. Unfortunately despite my modifications, I was never able to get a telnet password to display.

Task 2.2: Spoofing
Task 2.2A: Write a spoofing program

I begin by attempting an implementation of a basic spoofing program using raw sockets.

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <stdlib.h>

void main()
{
    int sd;
    struct sockaddr_in sin;
    char buffer[1024] = "Lab 08 yay";
    sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    if(sd < 0) {
        perror("socket() error");
        exit(-1);
    }

    memset((char*)&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = inet_addr("10.0.2.15");
    sin.sin_port = htons(9090);
    if(sendto(sd, buffer, sizeof(buffer), 0, (struct sockaddr *)&sin, sizeof(sin)) <
        perror("sendto() error"); exit(-1);
    }
    close(sd);
}
```

I compile and run the program many times, despite my best efforts, and hours of debugging, The program refuses to send the packets, they don't show up in netcat or wireshark

```
[11/09/20 JO481765]seed@VM:~/lab08$ gcc -o spoof spoof.
c -lpcap
[11/09/20 JO481765]seed@VM:~/lab08$ sudo ./spoof
[11/09/20 JO481765]seed@VM:~/lab08$
```

```
[11/09/20 JO481765]seed@VM:~/lab08$ nc -lvu 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

Task 2.2B: Spoof an ICMP Echo Request.
As it appears I can't spoof packets as shown in the last task, this is beyond by reach.

Question 4.
Yes, you can set the size to be an arbitrary value by setting the field, but you must also set the length in the sendto len parameter in order for it to work, otherwise sendto will cut it off.

Question 5
No this is automatically calculated and added.

Question 6
You need root privileges to run programs with raw sockets because raw sockets are dangerous since they can be used to spoof packets. If the program is run without root access, it would fail while trying to create a socket with a permission denied error.

Task 2.3: Sniff and then Spoof
As previously mentioned, I can't get spoofing to work properly, so i have no way of implementing this.