

FCS Assignment 3

-Dewangee Agrawal (2016034)

Part I :

- 1) I used the GnuPG library for creating the 4096 bit public-private key pair.

gpg --full-gen-key

```
dewangee ~$ gpg --full-gen-key
gpg (GnuPG) 2.2.11; Copyright (C) 2018 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/Users/dewangee/.gnupg' created
gpg: keybox '/Users/dewangee/.gnupg/pubring.kbx' created
Please select what kind of key you want:
  (1) RSA and RSA (default)
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
  0 = key does not expire
  <nb> = key expires in n days
  <nw> = key expires in n weeks
  <nm> = key expires in n months
  <ny> = key expires in n years
Key is valid for? (0) 1y
Key expires at Fri Nov 15 00:16:39 2019 IST
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Dewangee Agrawal
Email address: dewangee16034@iiitd.ac.in
Comment: FCS assignment 3
You selected this USER-ID:
  "Dewangee Agrawal (FCS assignment 3) <dewangee16034@iiitd.ac.in>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /Users/dewangee/.gnupg/trustdb.gpg: trustdb created
gpg: key 40594381C6DC8E08 marked as ultimately trusted
gpg: directory '/Users/dewangee/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/Users/dewangee/.gnupg/openpgp-revocs.d/5E61DB8CF30038D27D1EA27140594381C6DC8E08.rev'
public and secret key created and signed.

pub   rsa4096 2018-11-14 [SC] [expires: 2019-11-14]
      5E61DB8CF30038D27D1EA27140594381C6DC8E08
uid           Dewangee Agrawal (FCS assignment 3) <dewangee16034@iiitd.ac.in>
sub   rsa4096 2018-11-14 [E] [expires: 2019-11-14]

Dewangee ~$
```

- 2) *gpg --encrypt --sign --armor -r dewangee16034@iiitd.ac.in file.txt*

The "file.txt" was signed by my private key and encrypted using the public key. The encrypted file ("file.txt.asc") has been uploaded along with this PDF.

3) ***gpg -d file.txt.asc***

I was able to decrypt the file since I have the private key associated with the public key that was used to encrypt the file. The decrypted file ("file_decrypted") has been uploaded along with this PDF. But, if someone else has the file, they cannot decrypt it since they do not have my private key.

Part II :

1) The commands are -

- a) SHA1 - ***shasum -a 1 file.txt***
- b) SHA3 - ***sha3sum -a file.txt***
- c) SHA 224 - ***shasum -a 224 file.txt***
- d) SHA 256 - ***shasum -a 256 file.txt***
- e) SHA 384 - ***shasum -a 384 file.txt***
- f) SHA 512 - ***shasum -a 512 file.txt***
- g) MD5 - ***md5 file.txt***

In case of large files, the speed of the algorithms is -

MD5 > SHA-1 > SHA-224 > SHA-256 > SHA-384 > SHA-512 > SHA-3

```
Desktop — -bash — 142x41
Last login: Thu Nov 15 05:37:05 on ttys003
Dewangee:~ dewangee$ cd Desktop/
Dewangee:Desktop dewangee$ time shasum -a 1 file.txt
3148a2bd9a781afa2264f751226179a96a7036e2 file.txt

real    0m0.128s
user    0m0.104s
sys     0m0.020s
Dewangee:Desktop dewangee$ time shasum -a 224 file.txt
3bd17115f34bda352db8ddf2d4685946a6ffabaff0db1a4466ed3f1e file.txt

real    0m0.235s
user    0m0.212s
sys     0m0.021s
Dewangee:Desktop dewangee$ time shasum -a 256 file.txt
080e26f5b88f2f030a26a2b55654218bf4a960bb643f14772f5139b23620159c file.txt

real    0m0.238s
user    0m0.213s
sys     0m0.021s
Dewangee:Desktop dewangee$ time shasum -a 512 file.txt
68b3f57471339766e83890252a7e5880706ad9f8ba317cbb4946936249b3e74036634ff9bf167528373927f363d9e023ec9a183a0beebf56f5aeafb398d5d403 file.txt

real    0m0.184s
user    0m0.160s
sys     0m0.020s
Dewangee:Desktop dewangee$ time shasum -a 384 file.txt
a50f7071b71a3b395517552558486ef9e8e081e16a059bb6c90d54f6308126accdd6a4521be5828f8aa78b24d16904 file.txt

real    0m0.227s
user    0m0.196s
sys     0m0.026s
Dewangee:Desktop dewangee$ time md5 file.txt
MD5 (file.txt) = 4a2a9d7d13218651cf013769a7c660c4

real    0m0.100s
user    0m0.083s
sys     0m0.022s
Dewangee:Desktop dewangee$
```

2) a) **Two files** were changed - **100west.txt** and **16.lws**

The methodology used is - The **checksum** of the original file (downloaded from textstories) and the suspected tampered file (downloaded from drive) was calculated. If the checksum remained same for both the files, this implies that the file wasn't modified. But, if the checksum was different, this implies the file has been modified.

Checksum is used because only identical files have the same hash.

```
Original — -bash — 103x34
Dewangee:Desktop dewangee$ cd Original/
Dewangee:Original dewangee$ shasum -a 256 13chil.txt
ecd6b203438161418c2177f5495a24759370cb5b6486117bc969973f322327a0 13chil.txt
Dewangee:Original dewangee$ shasum -a 256 13chil_tampered.txt
ecd6b203438161418c2177f5495a24759370cb5b6486117bc969973f322327a0 13chil_tampered.txt
Dewangee:Original dewangee$ shasum -a 256 14.lws.txt
d2408682443c6cf550607b1a24428285a2cf3409486560f2572d99f1d6877df2 14.lws.txt
Dewangee:Original dewangee$ shasum -a 256 14_tampered.lws
d2408682443c6cf550607b1a24428285a2cf3409486560f2572d99f1d6877df2 14_tampered.lws
Dewangee:Original dewangee$ shasum -a 256 16.lws.txt
41423605ad62cc369e0cac21cb440adeafab4fc78522a792e781e274e34dc9ac 16.lws.txt
Dewangee:Original dewangee$ shasum -a 256 16_tampered.lws
db1216ad35a25122ac6bba45d58a6b5d5769a9daae559df300ac9cb66fcfe4b 16_tampered.lws
Dewangee:Original dewangee$ shasum -a 256 17.lws.txt
894af471f39950fa6f4e0416c86f43bf7316ab75f7c0d209c0a99b0be2987f82 17.lws.txt
Dewangee:Original dewangee$ shasum -a 256 17_tampered.lws
894af471f39950fa6f4e0416c86f43bf7316ab75f7c0d209c0a99b0be2987f82 17_tampered.lws
Dewangee:Original dewangee$ shasum -a 256 100west.txt
d4f747f19fabccf391b4ee60507cb67eaf7a6fba664ef1b27ae4757ad2a551c9 100west.txt
Dewangee:Original dewangee$ shasum -a 256 100west_tampered.txt
dd68f7a74bcacd86f632d91e85fb211d941095141670042d3e5801b8128b3577 100west_tampered.txt
Dewangee:Original dewangee$
```

b) The modified file cannot be detected because checksum is not a very reliable process. The detection technique might also fail in case of MD5 collisions whereby the file is modified to create another one with the same checksum. This can destroy the integrity of the file.

c) Cryptographic hash functions are used instead of hash functions to solve 3 fundamental problems associated with hash functions -

- Pre-image resistance - If we are given a hash h , then it is difficult to find a pre-image m , such that $h = \text{hash}(m)$.
- Second Pre-image resistance - If we are given a message m_1 , then it is difficult to find a message m_2 , such that $\text{hash}(m_1) = \text{hash}(m_2)$.
- Collision resistance - It should be hard to find messages m_1 and m_2 such that $\text{hash}(m_1) = \text{hash}(m_2)$.

The security property violated is **Collision resistance**. This is violated in the above mentioned case since MD5 collisions are possible and they take away the use case of cryptographic hash functions over hash functions.

Part III :

- 1) The code has been attached below. The code encrypts the passwords before storing them in the file. The passwords are hashed using a salt. This file can only be accessed by authorised users.

```

1  #include "passwd.h"
2  #include <crypt.h>
3  #include <unistd.h>
4  #include <errno.h>
5
6  static int is_registered(char *uname) {
7
8      FILE *db_file;
9
10     if ((db_file = fopen("user_db.txt", "r")) != NULL)
11     {
12         char user[2000], passwd[2000];
13         while (fscanf(db_file, "%s", user) == 1) {
14             fscanf(db_file, " ");
15             fscanf(db_file, "%s\n", passwd);
16             if (strcmp(uname, user) == 0) {
17                 return 1;
18             }
19         }
20         fclose(db_file);
21     }
22     else
23     {
24         printf("Error\n");
25         exit(1);
26     }
27
28
29
30
31     return 0;
32 }
33
34 int register_user(char *uname, char *passwd) {
35
36     if (access("user_db.txt", R_OK) != 0) {
37         if (errno == ENOENT) {
38             printf("File does not exist\n");
39             return -1;
40         }
41
42         if (errno == EACCES) {
43             printf("User does not have permissions to access database\n");
44             return -1;
45         }
46

```



```

        fprintf(stderr, "Error Occured\n");
        return -1;
    }

    if (is_registered(uname)) {
        fprintf(stderr, "Choose another username\n");
        return 0;
    }

    FILE *db_file = fopen("user_db.txt", "a");

    if (db_file != NULL)
    {
        fprintf(db_file, "%s", uname);
        fprintf(db_file, " ");
        fprintf(db_file, "%s\n", passwd);

        fclose(db_file);
    }
    else
    {
        exit(1);
    }

    return 1;
}

int auth_user(char *uname, char *passwd) {

    if (!is_registered(uname)) {
        fprintf(stderr, "User not registered\n");
        return 0;
    }

    FILE *db_file;

```

```

if ((db_file = fopen("user_db.txt", "r")) == NULL)
{
    char user[2000], password[2000];
    while (fscanf(db_file, "%s", user) == 1)
    {
        fscanf(db_file, " ");
        fscanf(db_file, "%s\n", password);

        if (strcmp(uname, user) != 0)
        {
        }
        else
        {
            if (strcmp(strdup(passwd), strdup(password)) == 0) {
                return 1;
            }
        }
    }

    fclose(db_file);
}

else
{
    exit(1);
}

return 0;
}

```



```

117 }
118
119 int main(int argc, char *argv[])
120 {
121     int register_flag = 0;
122     if (strcmp(argv[1], "-r") == 0) {
123         register_flag = 1;
124     }
125
126     else if (strcmp(argv[1], "-a") == 0)
127     {
128         register_flag = 0;
129     }
130     char uname[2000];
131
132     printf("Enter Username: ");
133     scanf("%s", uname);
134     unsigned long seed[2];
135     char salt[] = "$1$.....";
136     char temp[] = "abcdef";
137     const char *const seedchars =
138         "./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
139         "UVWXYZabcdefghijklmnopqrstuvwxyz";
140     seed[0] = 0;
141     seed[1] = 0;
142     char *password;
143     int i;
144
145     for (i = 0; i < 8; i++)
146         salt[3 + i] = seedchars[(seed[i / 5] >> (i % 5) * 6) & 0x3f];
147
148     password = crypt(getpass("Password:"), salt);
149     if (register_flag == 1) {
150         int a = register_user(uname, password);
151     }
152     else {
153         int a = auth_user(uname, password);
154
155         if (a == 1) {
156             printf("Authorised\n");
157         }
158         else {
159             printf("Incorrect\n");
160         }
161     }
162 }

```

2) For Brute Force, the following code can be run on the system. The commands are

-

```

gcc -o ./brute brute_force.c
./brute <username>

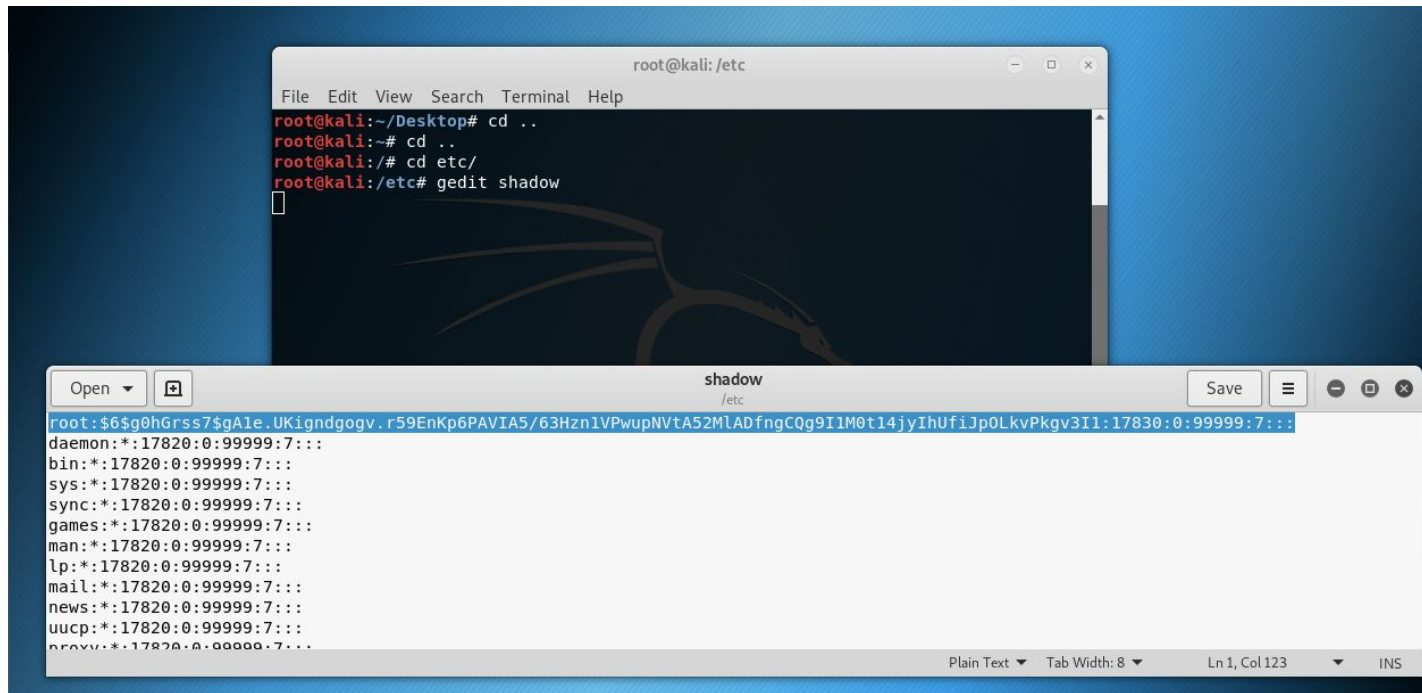
```

```
brute_force.c — Edited
brute_force.c > main
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main(int argc, char *argv[])
6 {
7     char username[200];
8
9     if (argc!=2)
10    {
11        exit(1);
12    }
13    else
14    {
15        strcpy(username, argv[1]);
16
17        FILE *fptr1;
18
19        fptr1 = fopen("./passwd.txt", "r");
20
21        if (fptr1 != NULL)
22        {
23            char password[1000];
24
25            while (fgets(password, 1000, fptr1) != NULL)
26            {
27                FILE *fp;
28
29                fp = popen("./main", "w");
30                fprintf(fp, "%s\n", username);
31                fprintf(fp, "%s\n", password);
32
33                pclose(fp);
34            }
35        }
36        else
37        {
38            exit(1);
39        }
40    }
41
42    return 0;
43 }
44 }
```

This code executes the previous code multiple times based on a password list saved in a file and launches brute force attack.

3) The passwd file in the etc folder stores the information about user-ids, account permissions and other information.

The shadow file stores sensitive information such as the account passwords in the form of a hash. So, when a user enters the password, it is hashed by the same crypt() function and this is compared to the value stored in this file.



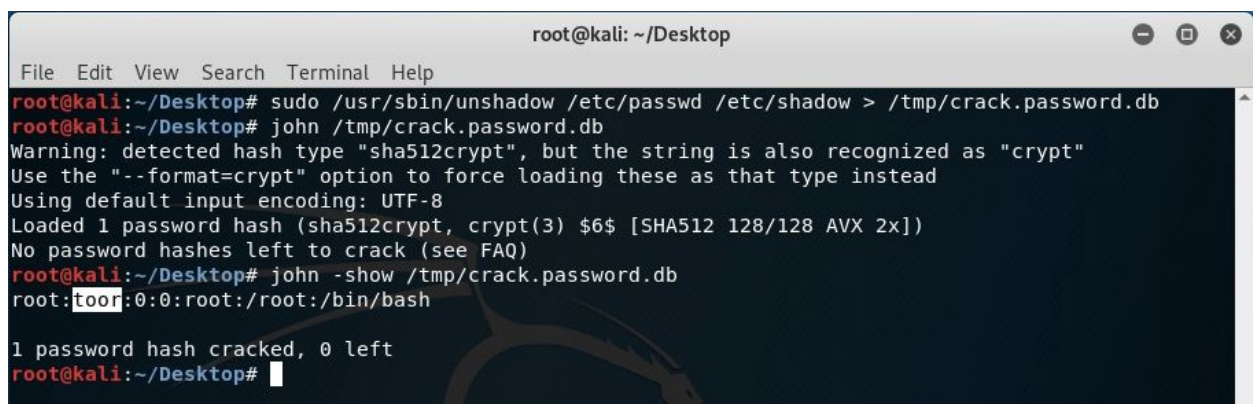
5) The commands used for the same include -

sudo /usr/sbin/unshadow /etc/passwd /etc/shadow > /tmp/crack.password.db

To crack the password : john /tmp/crack.password.db

To show the cracked File: john -show /tmp/crack.password.db

e



Part IV :

Note - The 4th question has been done on a Lab PC since iptables don't work on mac.

1) a) Block outside ping to my server -

```
iptables -A INPUT -p icmp --icmp-type echo-request -j DROP  
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j DROP
```

To check whether the ping to my server has been blocked, I ran the command-
ping 192.168.33.47. No response was observed.

To check whether traffic passes from my PC to other websites, I ran the command
ping google.com
Which is working.

```
iiitd@NameNode: ~  
iiitd@NameNode:~$ sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP  
iiitd@NameNode:~$ sudo iptables -A OUTPUT -p icmp --icmp-type echo-reply -j DROP  
iiitd@NameNode:~$ time ping 192.168.33.47  
PING 192.168.33.47 (192.168.33.47) 56(84) bytes of data.  
^C  
--- 192.168.33.47 ping statistics ---  
6 packets transmitted, 0 received, 100% packet loss, time 5039ms  
  
real    0m5.072s  
user    0m0.000s  
sys     0m0.000s  
iiitd@NameNode:~$ ping google.com  
PING google.com (172.217.166.238) 56(84) bytes of data.  
64 bytes from del03s14-in-f14.1e100.net (172.217.166.238): icmp_seq=1 ttl=55 time=2.88 ms  
64 bytes from del03s14-in-f14.1e100.net (172.217.166.238): icmp_seq=2 ttl=55 time=2.90 ms  
64 bytes from del03s14-in-f14.1e100.net (172.217.166.238): icmp_seq=3 ttl=55 time=3.25 ms  
64 bytes from del03s14-in-f14.1e100.net (172.217.166.238): icmp_seq=4 ttl=55 time=2.98 ms  
^C  
--- google.com ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3004ms  
rtt min/avg/max/mdev = 2.885/3.006/3.251/0.146 ms  
iiitd@NameNode:~$
```

On trying from another laptop, ping could not connect to my server.

```
Dewangee:Original dewangee$ ping 192.168.33.47  
PING 192.168.33.47 (192.168.33.47): 56 data bytes  
Request timeout for icmp_seq 0  
Request timeout for icmp_seq 1  
Request timeout for icmp_seq 2  
Request timeout for icmp_seq 3  
Request timeout for icmp_seq 4  
Request timeout for icmp_seq 5  
Request timeout for icmp_seq 6  
Request timeout for icmp_seq 7  
Request timeout for icmp_seq 8  
Request timeout for icmp_seq 9
```

b) I hosted a webpage, index.html, on the lab PC with the IP address 192.168.33.47. I used Apache2 for the same.

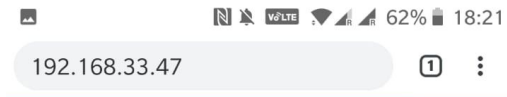
```
iiitd@NameNode: /var/www/html
iiitd@NameNode:/var/www/html$ gedit index.html
(gedit:6806): Gtk-WARNING **: Attempting to read the recently used resources file at '/home/iiitd/.local/share/recently-used.xbel', but the parser failed: Failed to open file '/home/iiitd/.local/share/recently-used.xbel': Permission denied.
iiitd@NameNode:/var/www/html$ sudo iptables -A INPUT -s 192.168.170.70 -j ACCEPT
iiitd@NameNode:/var/www/html$ sudo iptables -A OUTPUT -d 192.168.170.70 -j ACCEPT
iiitd@NameNode:/var/www/html$ sudo iptables -P INPUT DROP
iiitd@NameNode:/var/www/html$ sudo iptables -P OUTPUT DROP
iiitd@NameNode:/var/www/html$
```

Initially I blocked all IPs from accessing the file.

iptables -P INPUT DROP

iptables -P OUTPUT DROP

The error message observed was -



This site can't be reached

192.168.33.47 took too long to respond.

Try:

Checking the connection

ERR_CONNECTION_TIMED_OUT

RELOAD

DETAILS

The IP address of my phone is -

IP address

192.168.170.70
fe80::e396:cb8e:139:55ce

Wi-Fi MAC address

94:65:2d:b4:83:e1

Now, to allow only phone to access the webpage -

```
iptables -A INPUT -s 192.168.170.70 -j ACCEPT  
iptables -A OUTPUT -d 192.168.170.70 -j ACCEPT
```


2) a) Subnet mask for lab B519-
255.255.240.0/20

The PC I was using had the IP *192.168.33.47*.

Thus, the subnet ID - *192.168.33.47/20*.

Also, ssh is by default open on port 22.

Thus, the command used for the same -
nmap -sS -p 22 192.168.33.47/20 > file.txt

File.txt has been uploaded along with the document.

b) For OS fingerprinting, I used the command -

sudo nmap -O 192.168.43.110/20 > os_fingerprinting.txt

A terminal window titled "Desktop — -bash — 110x33" with a dark background. The terminal shows the following text:

```
Last login: Thu Nov 15 14:42:37 on ttys000
Dewangee:~ dewangee$ cd Desktop/
Dewangee:Desktop dewangee$ nmap -O 192.168.65.105/20 > os_fingerprint.txt
TCP/IP fingerprinting (for OS scan) requires root privileges.
QUITTING!
Dewangee:Desktop dewangee$ sudo !!
sudo nmap -O 192.168.65.105/20 > os_fingerprint.txt
Password:
Dewangee:Desktop dewangee$ sudo nmap -O 192.168.43.110/20 > os_fingerprinting.txt
Password:
Dewangee:Desktop dewangee$
```

The file has been uploaded along with the submission.

Out of 164 scanned IPs, the ones using windows turned out to be 8 and the ones using linux were 156.

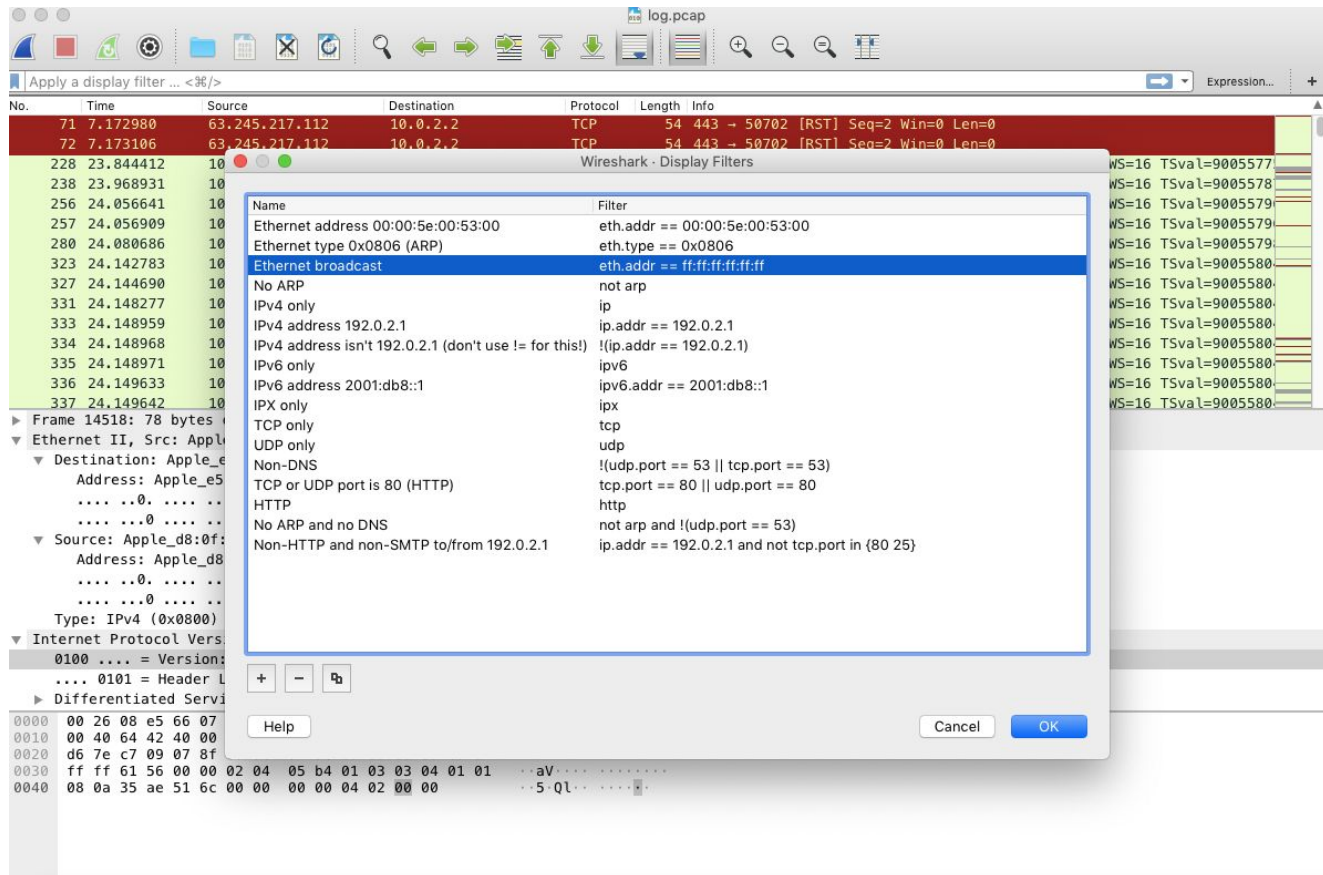
3) OpenVpn is installed by the commands o
wget https://git.io/vpn -O openvpn-install.sh && bash openvpn-install.sh

The keys were created for the user and the certificates using this process. After the new user is added the webpage is hosted by changing the server.conf file and

allowing to host the same webpage that was created earlier. I then added my phone IP address to the allowed hosts and accessed the webpage via the VPN.

Part V :

- 1) The ethernet broadcast of the local network is ff:ff:ff:ff:ff:ff.



The same is used as a display filter and the resulting MAC and IP Addresses are as follows. -

MAC Addresses -

log.pcap

eth.addr == ff:ff:ff:ff:ff:ff

No. Time Source Destination Protocol Length Info

Wireshark · Endpoints · log.pcap

Ethernet · 4 IPv4 · 6 IPv6 TCP UDP · 12

Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
00:26:08:e5:66:07	40	8859	40	8859	0	0
04:0c:ce:d8:0f:fa	14	3458	14	3458	0	0
8c:a9:82:50:f0:a6	28	3196	28	3196	0	0
ff:ff:ff:ff:ff:ff	82	15 k	0	0	82	15 k

☐ Name resolution ☒ Limit to display filter

Endpoint Types

Help Copy

Close

IP Addresses -

log.pcap

eth.addr == ff:ff:ff:ff:ff:ff

No. Time Source Destination Protocol Length Info

Wireshark · Endpoints · log.pcap

Ethernet · 4 IPv4 · 6 IPv6 TCP UDP · 12

Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	AS Number	AS Organization
0.0.0.0	15	5130	15	5130	0	0	—	—	—	—
10.0.2.1	17	3393	17	3393	0	0	—	—	—	—
10.0.2.2	14	3458	14	3458	0	0	—	—	—	—
10.0.2.3	25	3070	25	3070	0	0	—	—	—	—
10.0.2.255	49	8192	0	0	49	8192	—	—	—	—
255.255.255.255	22	6859	0	0	22	6859	—	—	—	—

☐ Name resolution ☒ Limit to display filter

Endpoint Types

Help Copy

Close

2) Since the number of hosts is 3 as seen in the above screenshot - 10.0.2.1, 10.0.2.2, 10.0.2.3.

This seems like a home or small institution's local network. The websites visited frequently include educational and social networking websites. This can be seen from the DNS requests.

3) The IP address of the FTP server is **192.109.21.66**.

As seen below, all the FTP requests are sent by the host to this IP and it responds accordingly. For example, Login Incorrect etc.

The image shows a Wireshark packet capture of an FTP session. The packet list at the top shows a series of requests and responses between 194.109.21.66 and 10.0.2.2. The packet details for frame 14603 show the raw data of an FTP response, which is decoded as ASCII text.

No.	Time	Source	Destination	Protocol	Length	Info
146...	83.830702	194.109.21.66	10.0.2.2	FTP	72	Response: 220-
146...	83.830754	194.109.21.66	10.0.2.2	FTP	136	Response: 220-Welcome to the XS4ALL archive, Please login as 'anon
146...	83.830785	194.109.21.66	10.0.2.2	FTP	130	Response: 220-your E-mail address as the password to access the arc
146...	83.830815	194.109.21.66	10.0.2.2	FTP	72	Response: 220-
146...	83.830846	194.109.21.66	10.0.2.2	FTP	143	Response: 220-All anonymous transfers are logged with your host nam
146...	83.830868	194.109.21.66	10.0.2.2	FTP	144	Response: 220-entered for the password. If you don't like this poli
146...	83.830890	194.109.21.66	10.0.2.2	FTP	73	Response: 220-
146...	83.830911	194.109.21.66	10.0.2.2	FTP	72	Response: 220
146...	83.834608	10.0.2.2	194.109.21.66	FTP	79	Request: AUTH GSSAPI
146...	83.942844	194.109.21.66	10.0.2.2	FTP	104	Response: 530 Please login with USER and PASS.
164...	115.175217	10.0.2.2	194.109.21.66	FTP	89	Request: USER laticia.langhans
164...	115.283635	194.109.21.66	10.0.2.2	FTP	100	Response: 331 Please specify the password.
165...	121.727321	10.0.2.2	194.109.21.66	FTP	83	Request: PASS gobblue3859
165...	122.838427	194.109.21.66	10.0.2.2	FTP	88	Response: 530 Login incorrect.
165...	122.931177	10.0.2.2	194.109.21.66	FTP	72	Request: SYST

Frame 14603: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits)
Ethernet II, Src: Apple_e5:66:07 (00:26:08:e5:66:07), Dst: Apple_d8:0f:fa (04:0c:ce:d8:0f:fa)
Internet Protocol Version 4, Src: 194.109.21.66, Dst: 10.0.2.2
Transmission Control Protocol, Src Port: 21, Dst Port: 50952, Seq: 7, Ack: 1, Len: 70
File Transfer Protocol (FTP)
[Current working directory:]

```
0000 04 0c ce d8 0f fa 00 26 08 e5 66 07 08 00 45 00  ....&..f...E-
0010 00 7a aa b1 40 00 35 06 b7 1b c2 6d 15 42 0a 00  .z..@.5...m.B..
0020 02 02 00 15 c7 08 b8 62 2c a7 08 77 b9 ab 80 18  ....b...w....
0030 00 72 1e 66 00 00 01 01 08 0a 55 c0 86 12 35 ae  .r.f.....U...5-
0040 51 65 32 32 30 2d 57 65 6c 63 6f 6d 65 20 74 6f  Qe220-We lcome to
0050 20 74 68 65 20 58 53 34 41 4c 4c 20 61 72 63 68  the XS4 ALL arch
0060 69 76 65 2c 20 20 50 6c 65 61 73 65 20 6c 6f 67  ive, Pl ease log
0070 69 6e 20 61 73 20 60 61 6e 6f 6e 79 6d 6f 75 73  in as 'a nonymous
0080 27 20 77 69 74 68 0d 0a                          ' with..
```

The DNS hostname of this IP is **xs4all** -

```

Type: A (Host Address) (1)
Class: IN (0x0001)
▼ Answers
▶ ftp.mirror.nl: type CNAME, class IN, cname download.xs4all.nl
▶ download.xs4all.nl: type CNAME, class IN, cname dl.xs4all.nl
▶ dl.xs4all.nl: type A, class IN, addr 194.109.21.66
▼ Authoritative nameservers
▶ xs4all.nl: type NS, class IN, ns ns.xs4all.nl
▶ xs4all.nl: type NS, class IN, ns ns2.xs4all.nl
▼ Additional records
▶ ns.xs4all.nl: type A, class IN, addr 194.109.6.67
▶ ns2.xs4all.nl: type A, class IN, addr 194.109.9.100
[Request In: 14362]
[Time: 0.674762000 seconds]

```

FTP is not a secure method of transferring packets since the data is sent in plain text and can be viewed by anyone who intercepts this information. This is susceptible to man in the middle attack.

Thus, more secure mechanisms like HTTPS and FTPS should be used which allow encryption of packets via SSL or TLS.

4) One HTTPS website surfed is **pnc.com**. The screenshot attached below shows encrypted application data.

This has an advantage over FTP since the data is encrypted via SSL or TLS and not susceptible to man in the middle attack.

The screenshot shows a Wireshark packet capture of a TLS session. The packet list on the left shows a series of TLSv1 packets. The selected packet (No. 21682) is a TLSv1 application data packet (1071 bytes) from 171.385581 to 10.0.2.3. The packet details on the right show the TLS record layer with content type 'Application Data' and length 1012. The packet bytes at the bottom show the encrypted application data.

No.	Time	Source	Destination	Protocol	Length	Info
216...	171.381854	www.pnc.com	10.0.2.3	TLSv1	91	Encrypted Handshake Message
216...	171.382056	10.0.2.3	www.pnc.com	TLSv1	97	Change Cipher Spec, Encrypted Handshake Message
216...	171.383451	10.0.2.3	www.pnc.com	TLSv1	1053	Application Data
216...	171.383834	10.0.2.3	www.pnc.com	TLSv1	1054	Application Data
216...	171.384409	10.0.2.3	www.pnc.com	TCP	54	55609 → 443 [ACK] Seq=215 Ack=123 Win=66304 Len=0
216...	171.385110	10.0.2.3	www.pnc.com	TLSv1	1062	Application Data
216...	171.385581	10.0.2.3	www.pnc.com	TLSv1	1071	Application Data
216...	171.386134	10.0.2.3	www.pnc.com	TLSv1	1066	Application Data
216...	171.388255	10.0.2.3	www.pnc.com	TLSv1	97	Change Cipher Spec, Encrypted Handshake Message
216...	171.389064	10.0.2.3	www.pnc.com	TLSv1	1056	Application Data
216...	171.404043	statse.webtrends.a...	10.0.2.3	TCP	54	443 → 55611 [ACK] Seq=2689 Ack=538 Win=4917 Len=0
216...	171.406297	statse.webtrends.a...	10.0.2.3	TLSv1	97	Change Cipher Spec, Encrypted Handshake Message
216...	171.425841	10.0.2.3	statse.webtrends.a...	TLSv1	1166	Application Data
216...	171.438956	www.pnc.com	10.0.2.3	TCP	54	443 → 55605 [ACK] Seq=123 Ack=258 Win=35216 Len=0
216...	171.440976	www.pnc.com	10.0.2.3	TCP	54	443 → 55606 [ACK] Seq=123 Ack=258 Win=35216 Len=0

Frame 21682: 1071 bytes on wire (8568 bits), 1071 bytes captured (8568 bits)

Ethernet II, Src: IntelCor_50:f0:a6 (8c:a9:82:50:f0:a6), Dst: Apple_e5:66:07 (00:26:08:e5:66:07)

Internet Protocol Version 4, Src: 10.0.2.3 (10.0.2.3), Dst: www.pnc.com (170.201.60.3)

Transmission Control Protocol, Src Port: 55608, Dst Port: 443, Seq: 258, Ack: 123, Len: 1017

Secure Sockets Layer

▼ TLSv1 Record Layer: Application Data Protocol: http-over-tls

Content Type: Application Data (23)

Version: TLS 1.0 (0x0301)

Length: 1012

Encrypted Application Data: df2927f4d2a98b3d6f5d66cc68ccd685053acec64e197687...

0030 01 03 11 df 00 00 17 03 01 03 f4 df 29 27 f4 d2)'..

0040 a9 8b 3d 6f 5d 66 cc 68 cc d6 85 05 3a ce c6 4e ...=o]f.hN

0050 19 76 87 a1 84 5d 26 06 6c d1 49 3b 00 79 ab 9e ...v...]&.l.I;y..

0060 95 e0 4b 6b c5 dd a8 19 55 f4 a5 44 00 8c 83 f5 ...Kk...U..D....

0070 e5 f9 cf 9b 97 9b 71 c1 d6 11 49 63 6c 74 52 04q...IcltR

0080 57 4f b2 50 05 b4 cc 68 c2 b8 2a 4e 39 92 e5 18 W0.P...h...N9...

0090 21 92 ce ea cd 40 cf 97 29 0f e1 fb f1 bf f6 ca !...@...).....

00a0 ae c5 e5 52 f3 a6 75 ef 45 3c ab 53 2e b2 42 05 ...R..u..E<.S..B

00b0 cc 27 f2 4e f4 47 b8 39 b4 ed 92 09 c8 ea 22 13 ...'N.G.9"

00c0 b5 4b c7 f0 01 f0 1a 34 8d 47 2c bd ae 59 2f af .K.....4 .G.,.Y/..

Payload is encrypted application data (ssl.app_data), 1012 bytes

Packets: 22650 - Displayed: 22650 (100.0%) Profile: Default

5) The security property violated in the case of Facebook is Authentication. This is because it has no proper mechanism to authenticate the identity of the users. A user can have multiple accounts with fake names and other credentials and there is no way by which Facebook can make ensure identity of the person via the browser as even authentic people might change their browser and people with fake accounts might use different accounts for the same purpose. No CA signed certificates are issues to the users.

2) The code has been uploaded along with the submission - the file name is -

test.py


Language used - Python 2.7

- The code uses **dpkt** library to read the pcap file.
- The eth data is extracted to find ip packets.
- The ip data is extracted to find the tcp packets.
- The IPs sending the SYN packets with 2 way handshake are then extracted and printed.

Guidelines to run the code - ***python test.py***

This should be in a virtual environment with the dpkt package installed.

Output - The malicious IPs found are - **10.0.2.3** and **10.0.2.2**



```
Run: test x
/Users/dewangee/Desktop/FCSAssignment3/Part5/venv/bin/python /Users/dewangee/Desktop/FCSAssignment3/Part5/test.py
['10.0.2.3', '10.0.2.2']
Process finished with exit code 0
```