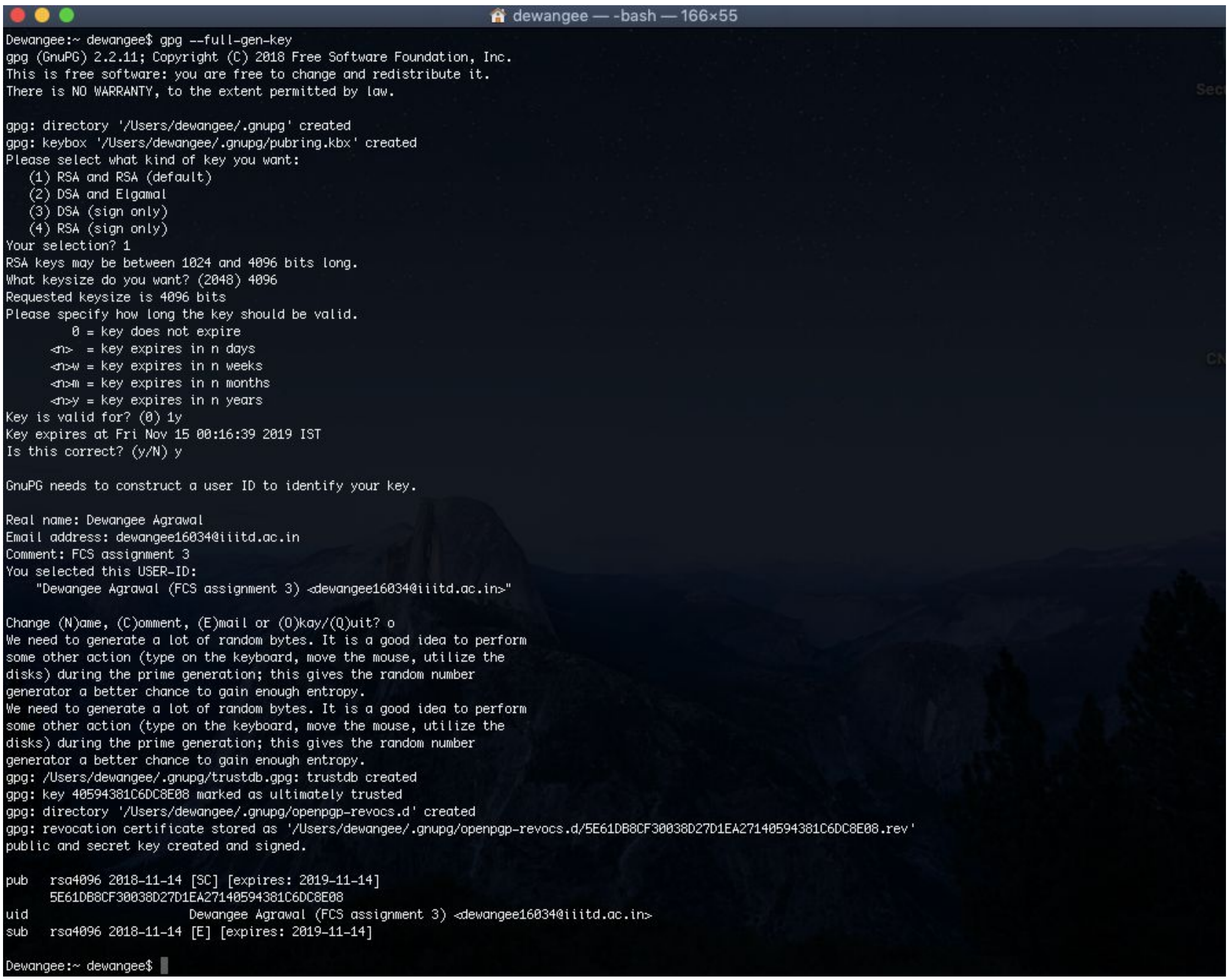
FCS Assignment 3

-Dewangee Agrawal (2016034)

Part I :

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

gpg --full-gen-key



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***

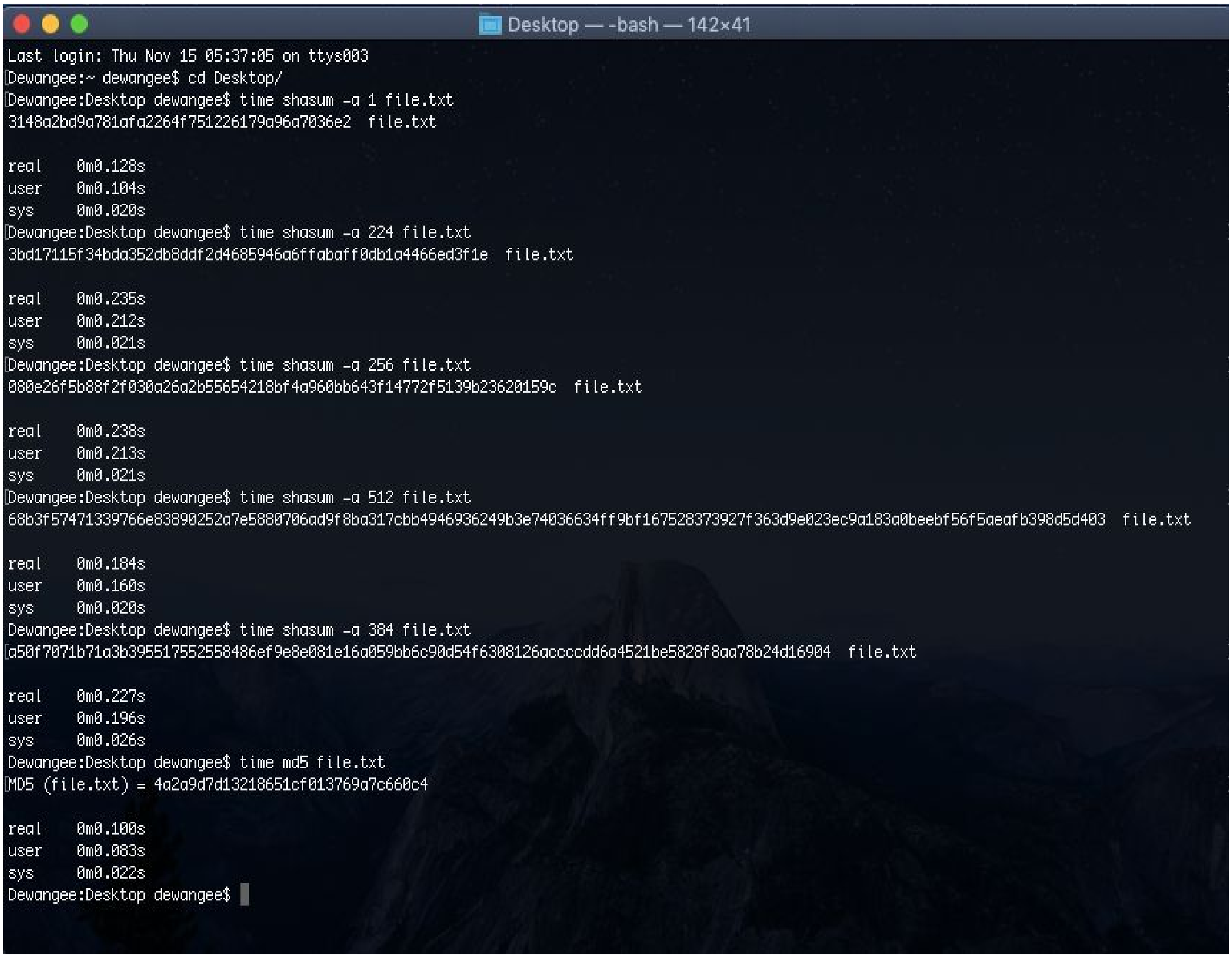
I was able to decrypt the file since I have the private key associated with the public key that was used to encrypt the file. The decrypted file (“file_decrypted”) has been uploaded along with this PDF. But, if someone else has the file, they cannot decrypt it since they do not have my private key.

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 -

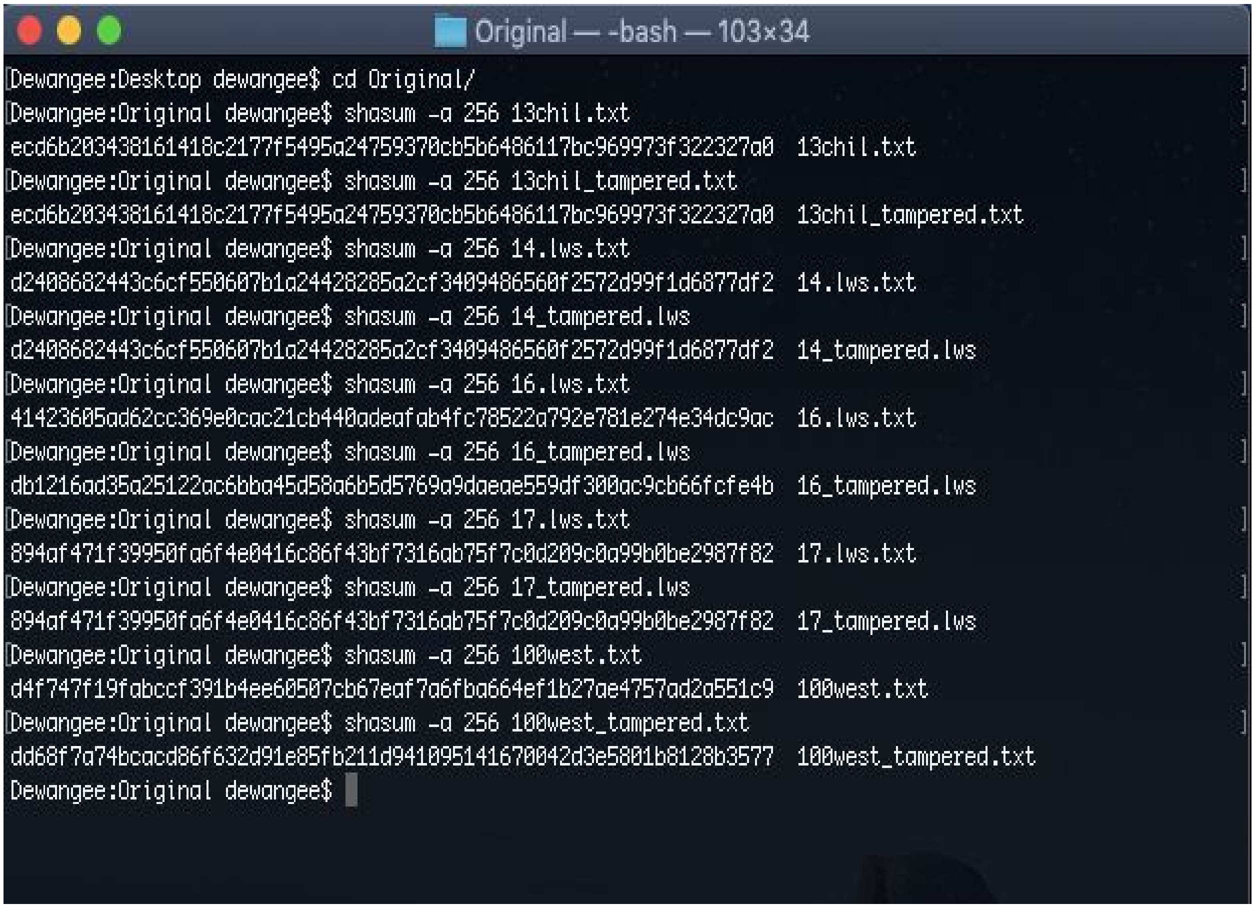
MD5 > SHA-1 > SHA-224 > SHA-256 > SHA-384 > SHA-512 > SHA-3



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.



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 = hash(m).*
- Second Pre-image resistance - If we are given a message *m1*, then it is difficult to find a message *m2*, such that *hash(m1) = hash(m2).*
- Collision resistance - It should be hard to find messages *m1* and *m2* such that *hash(m1) = hash(m2).*

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.

```c
#include "passwd.h"
#include <crypt.h>
#include <unistd.h>
#include <errno.h>

static int is_registered(char *uname) {

    FILE *db_file;

    if ((db_file = fopen("user_db.txt", "r")) != NULL)
    {
        char user[2000], passwd[2000];
        while (fscanf(db_file, "%s", user) == 1) {
            fscanf(db_file, " ");
            fscanf(db_file, "%s\n", passwd);
            if (strcmp(uname, user) == 0) {
                return 1;
            }
        }

        fclose(db_file);
    }
    else
    {
        printf("Error\n");
        exit(1);
    }



    return 0;
}

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

    if (access("user_db.txt", R_OK) != 0) {
        if (errno == ENOENT) {
            printf("File does not exist\n");
            return -1;
        }

        if (errno == EACCES) {
            printf("User does not have permissions to access database\n");
            return -1;
        }
```

```
        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;
```

```c
    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;
}
```

```c
}

int main(int argc, char *argv[])
{
    int register_flag = 0;
    if (strcmp(argv[1], "-r") == 0) {
        register_flag = 1;
    }

    else if (strcmp(argv[1], "-a") == 0)
    {
        register_flag = 0;
    }
    char uname[2000];

    printf("Enter Username: ");
    scanf("%s", uname);
    unsigned long seed[2];
    char salt[] = "$1$........";
    char temp[] = "abcdef";
    const char *const seedchars =
        "./0123456789ABCDEFGHIJKLMNOPQRST"
        "UVWXYZabcdefghijklmnopqrstuvwxyz";
    seed[0] = 0;
    seed[1] = 0;
    char *password;
    int i;

    for (i = 0; i < 8; i++)
        salt[3 + i] = seedchars[(seed[i / 5] >> (i % 5) * 6) & 0x3f];

    password = crypt(getpass("Password:"), salt);
    if (register_flag == 1) {
        int a = register_user(uname, password);
    }
    else {
        int a = auth_user(uname, password);

        if (a == 1) {
            printf("Authorised\n");

        }
        else {
            printf("Incorrect\n");
        }
    }
```

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

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

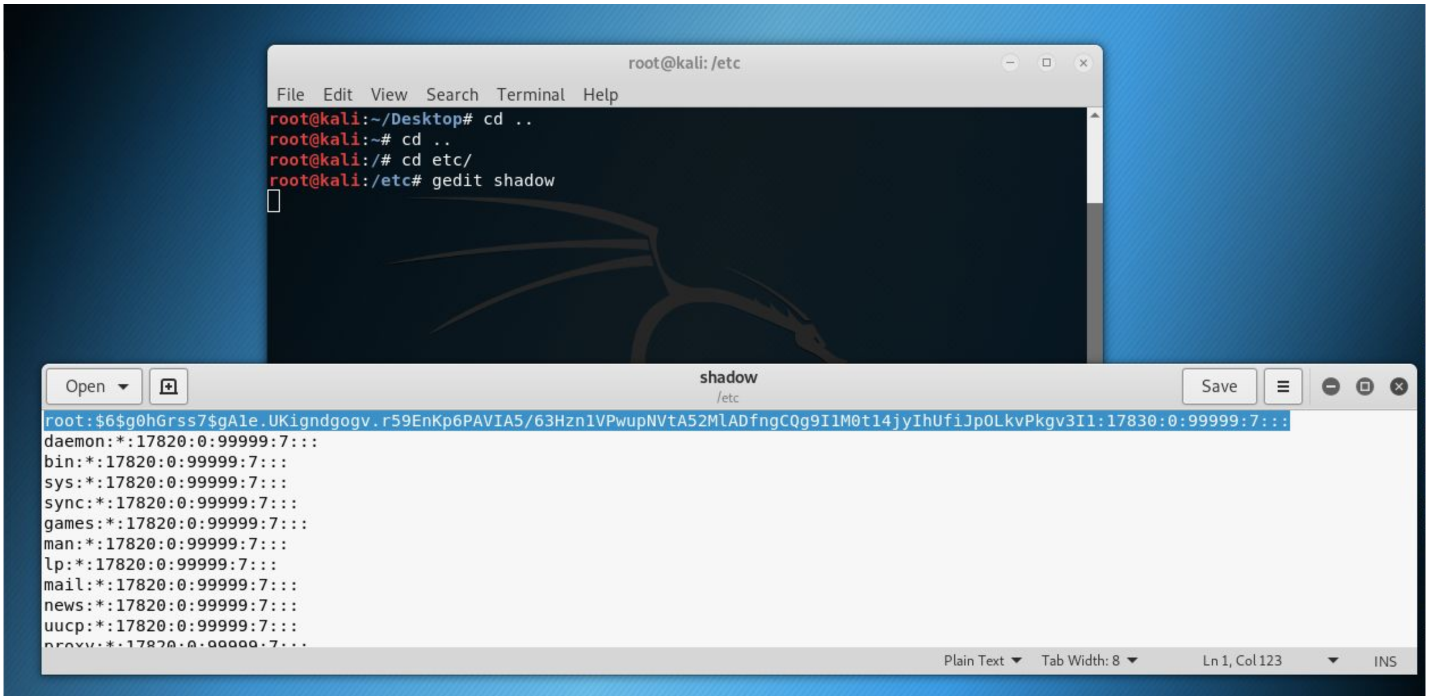
brute_force.c — Edited

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char username[200];

    if (argc!=2)
    {
        exit(1);
    }
    else
    {
        strcpy(username, argv[1]);

        FILE *fptr1;

        fptr1 = fopen("./passwd.txt","r");

        if (fptr1  != NULL)
        {
            char password[1000];

            while (fgets(password, 1000, fptr1) != NULL)
            {
                FILE *fp;

                fp = popen("./main","w");
                fprintf(fp, "%s\n", username);
                fprintf(fp, "%s\n", password);

                pclose(fp);
            }
        }
        else
        {
            exit(1);
        }

    }

    return 0;
}
```

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.



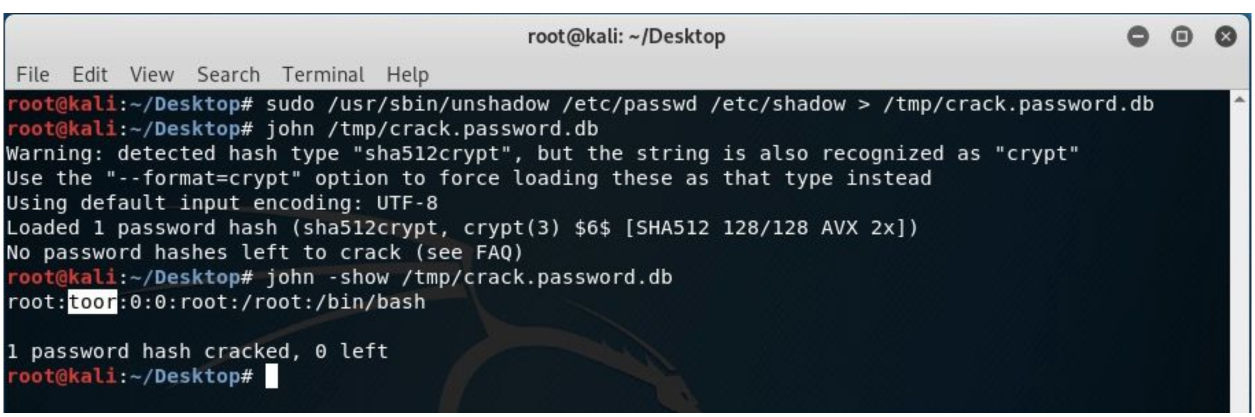
5) The commands used for the same include -

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.

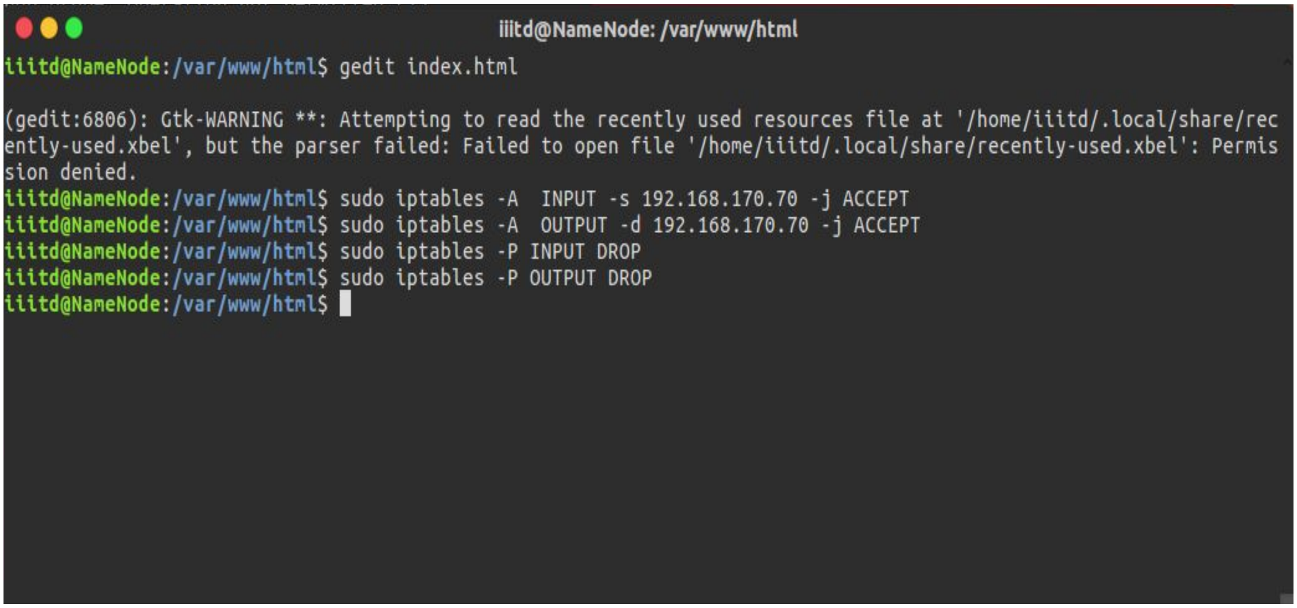
To check whether traffic passes from my PC to other websites, I ran the command ***ping google.com***
Which is working.

```
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.

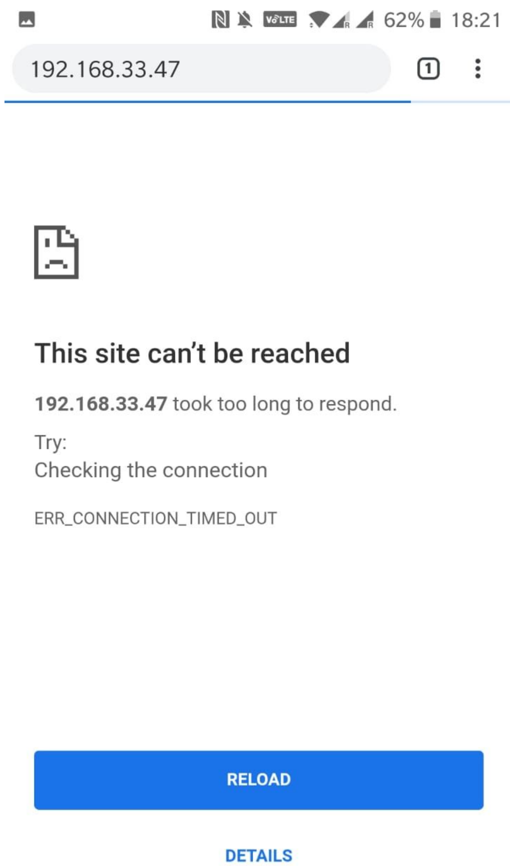
```
Dewangee:Original dewangee$ ping 192.168.33.47
PING 192.168.33.47 (192.168.33.47): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
Request timeout for icmp_seq 5
Request timeout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

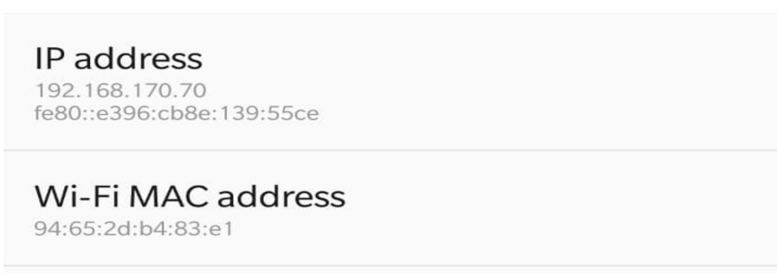


Initially I blocked all IPs from accessing the file.
iptables -P INPUT DROP
iptables -P OUTPUT DROP

The error message observed was -



The IP address of my phone is -



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

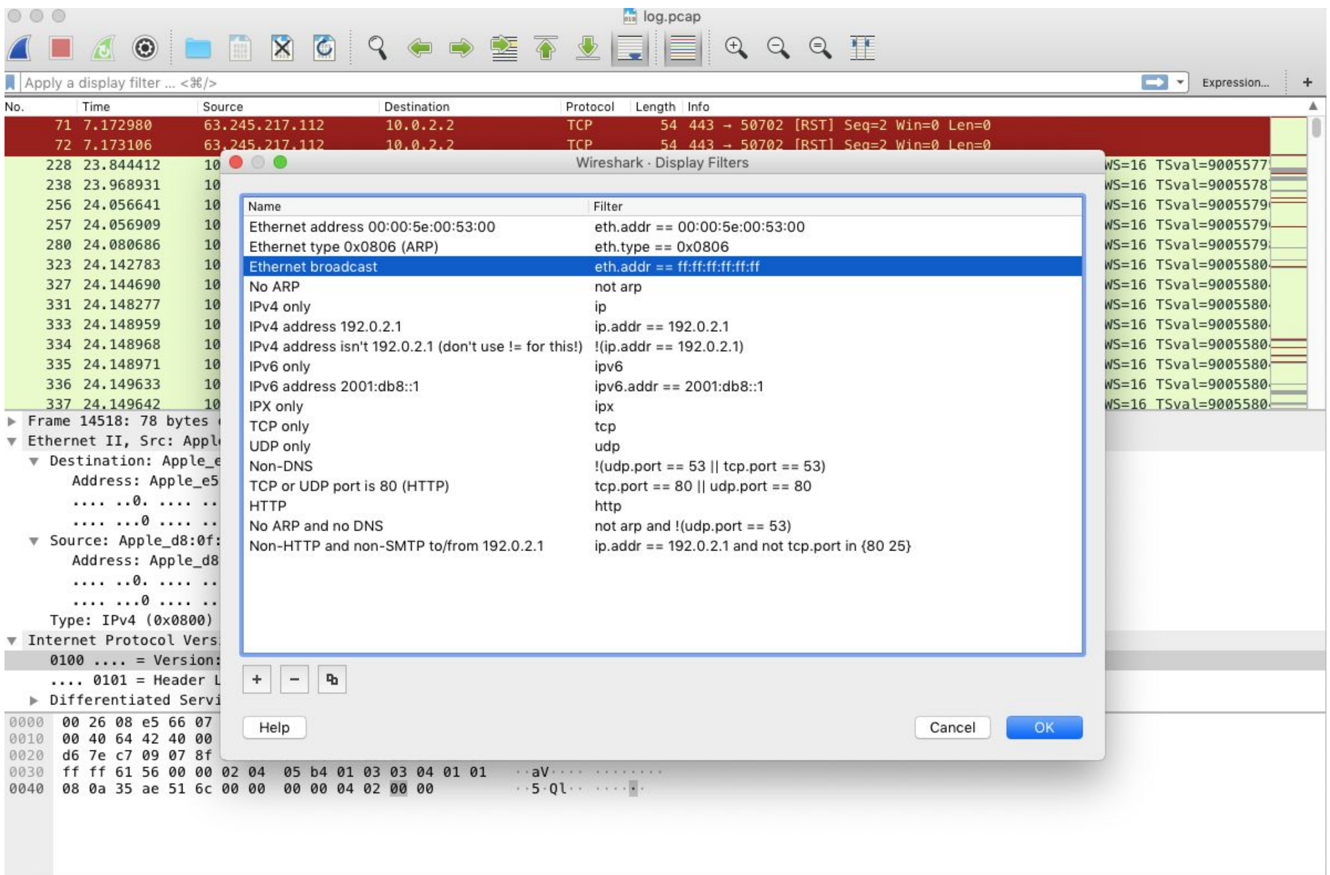
The file has been uploaded along with the submission.
Out of 164 scanned IPs, the ones using windows turned out to be 8 and the ones using linux were 156.

3) OpenVpn is installed by the commands 0
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.

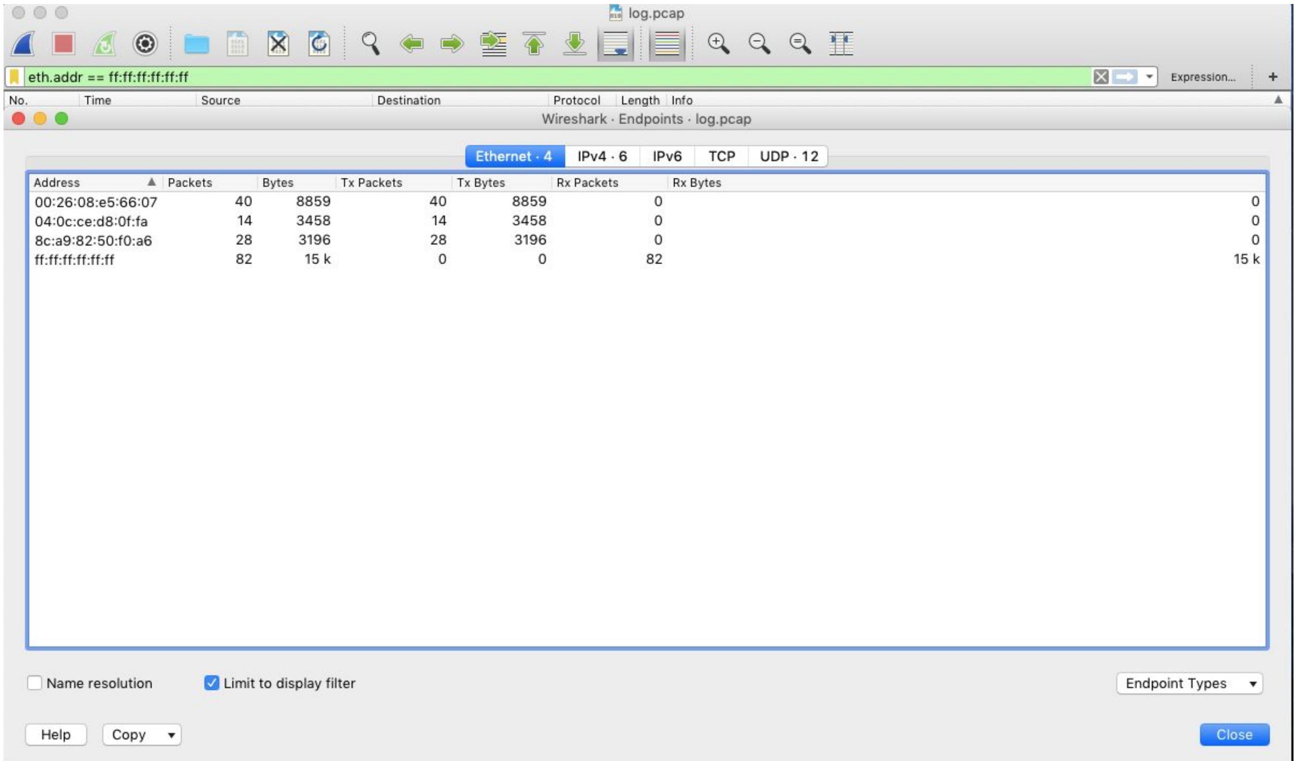
Part V :

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

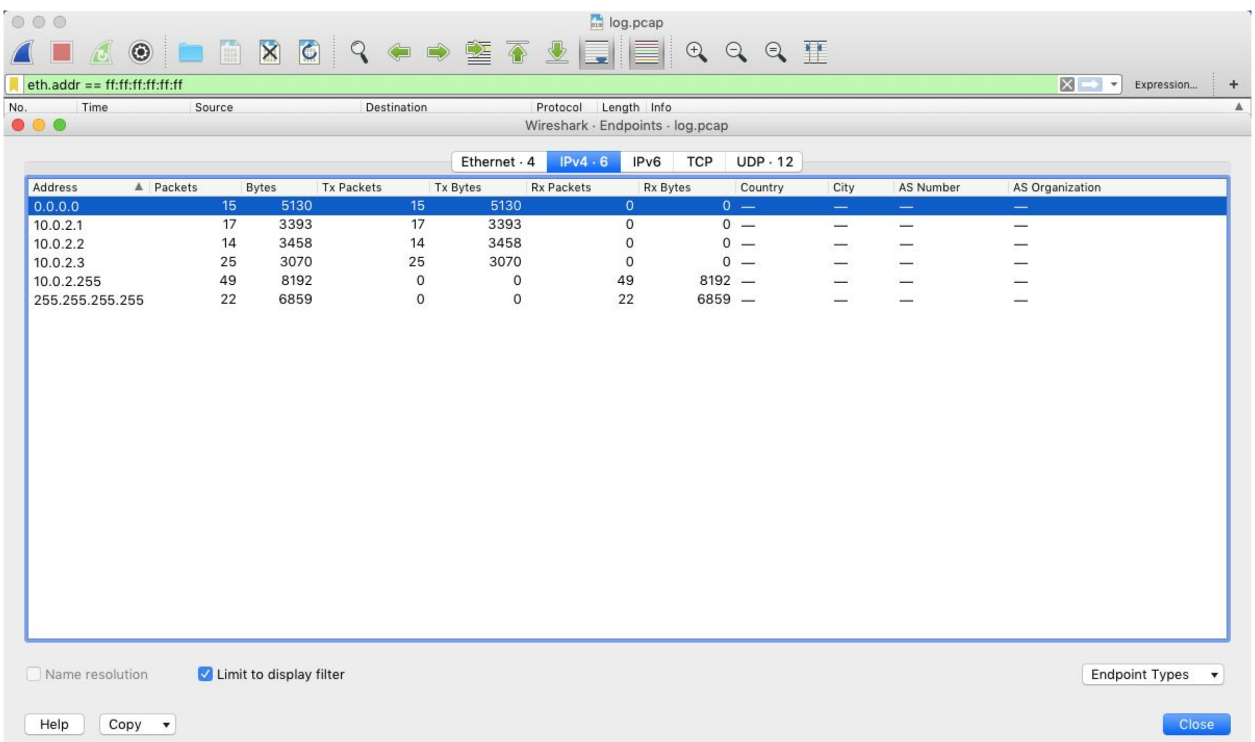


The same is used as a display filter and the resulting MAC and IP Addresses are as follows. -

MAC Addresses -



IP Addresses -

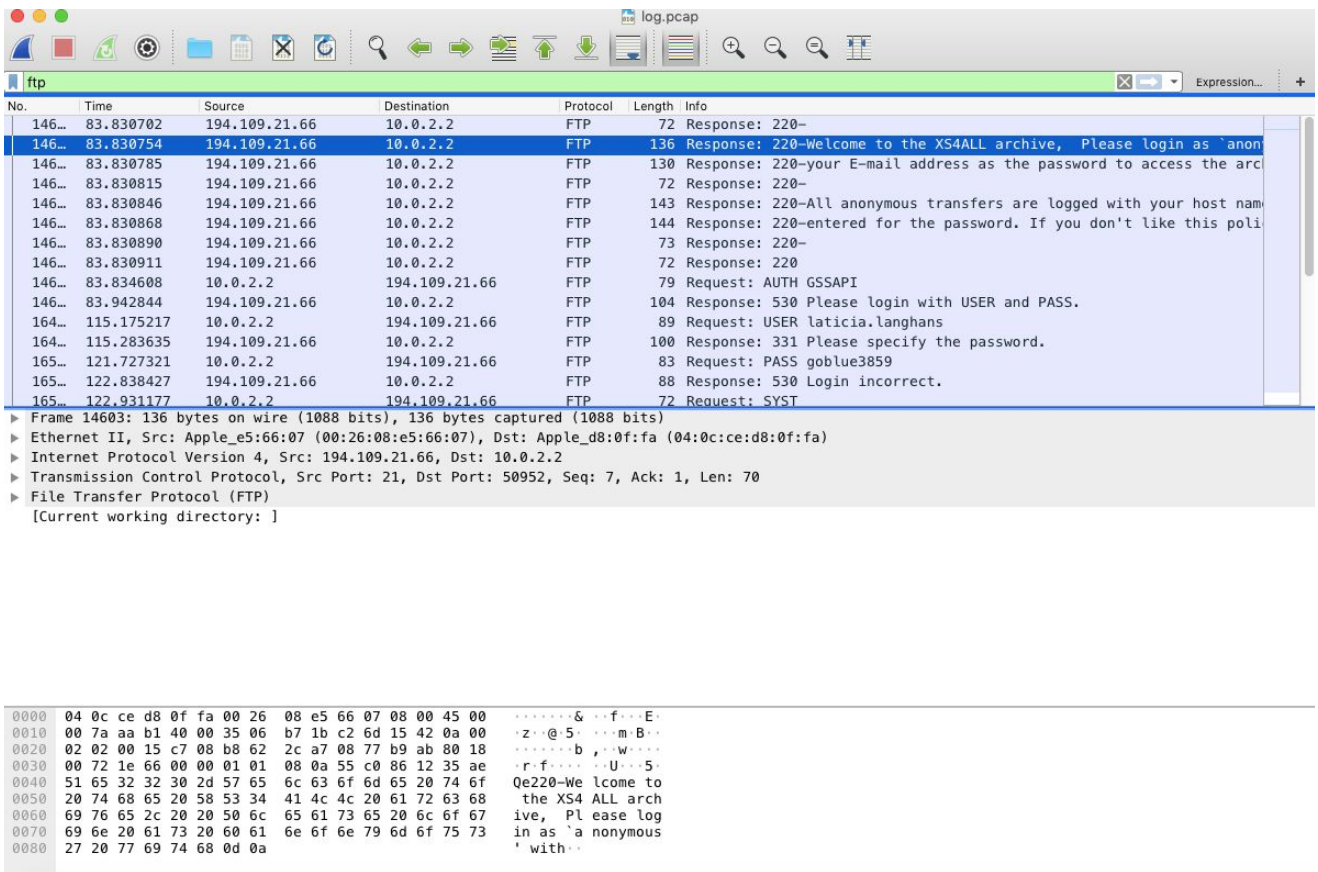


2) Since the number of hosts is 3 as seen in the above screenshot - 10.0.2.1, 10.0.2.2. 10.0.2.3.

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.

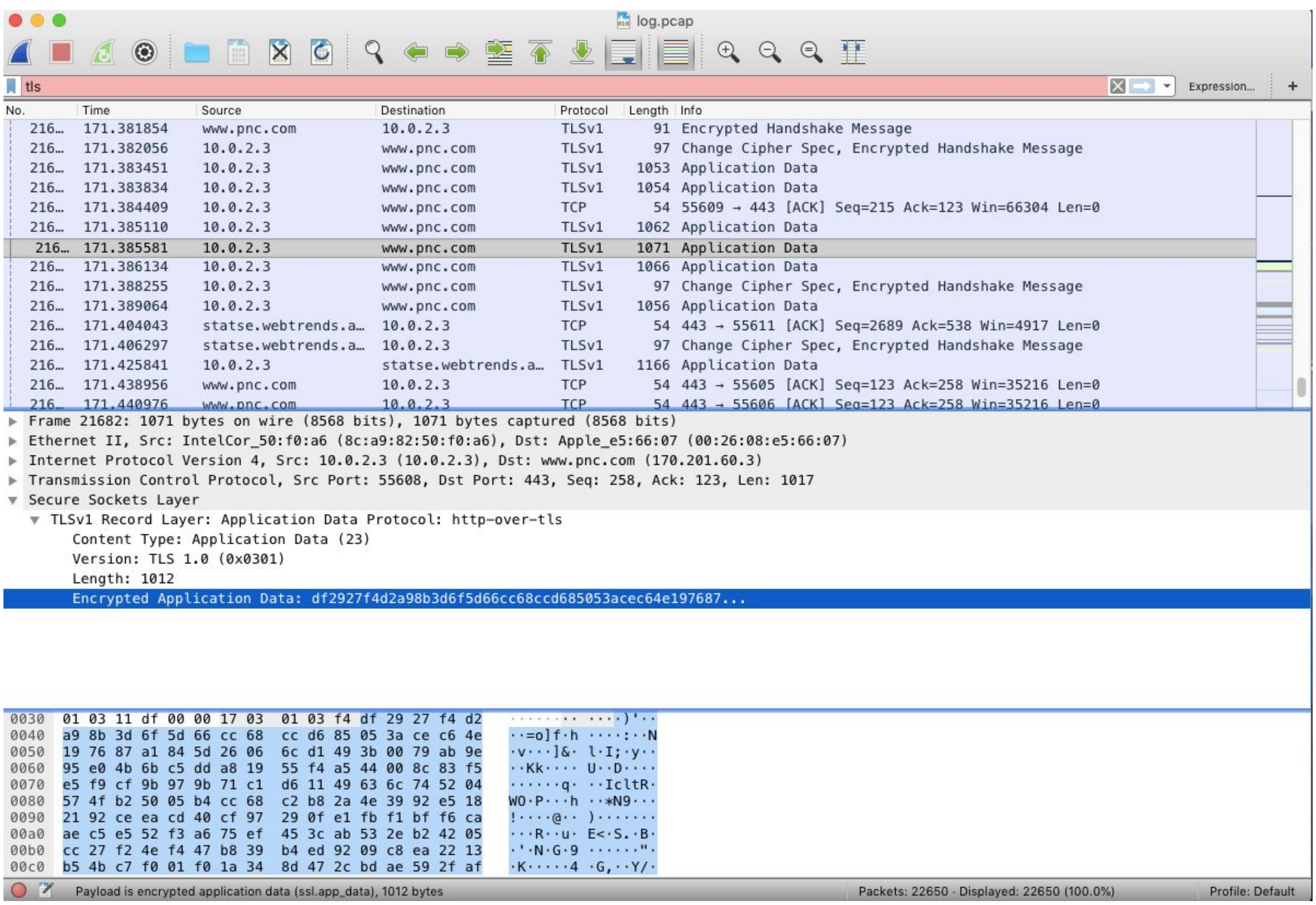


The DNS hostname of this IP is **xS4all** -

```
        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.



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**