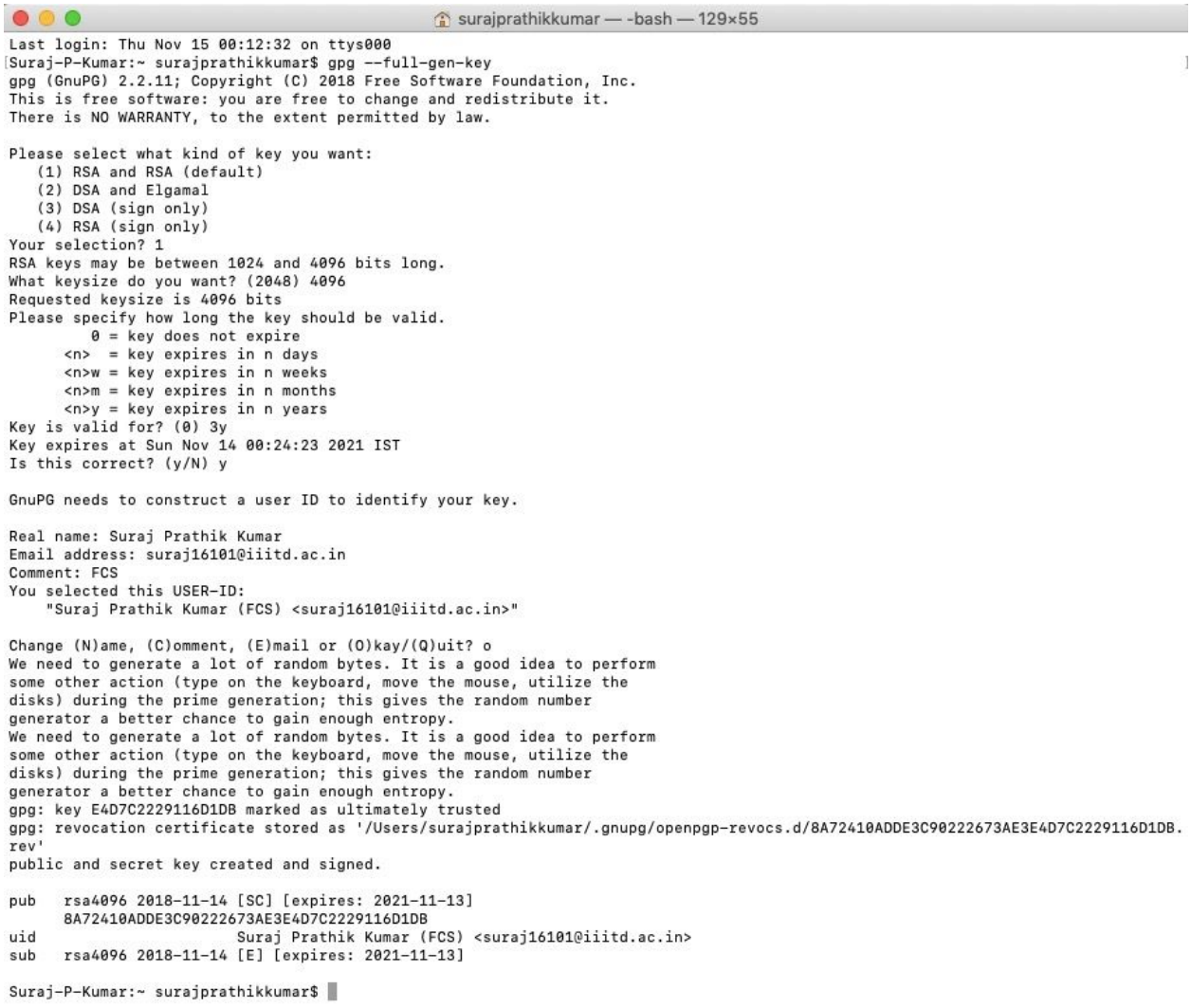
FCS Assignment

-Suraj Prathik Kumar(2016101)

Question 1

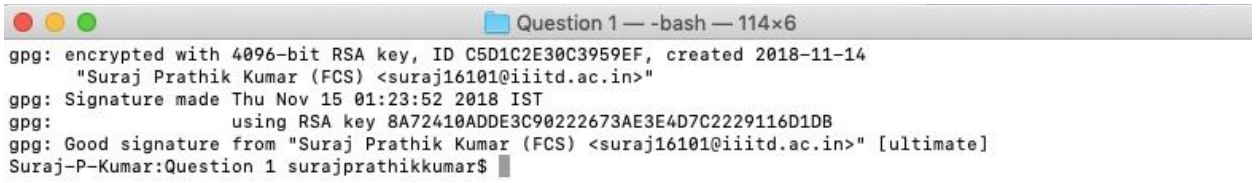
All the file, encrypted, decrypted, Public and the Private key has been uploaded.

1. **gpg --full-gen-key**



```
surajprathikkumar — -bash — 129×55
Last login: Thu Nov 15 00:12:32 on ttys000
Suraj-P-Kumar:~ surajprathikkumar$ 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.

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
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 3y
Key expires at Sun Nov 14 00:24:23 2021 IST
Is this correct? (y/N) y

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

Real name: Suraj Prathik Kumar
Email address: suraj16101@iiitd.ac.in
Comment: FCS
You selected this USER-ID:
    "Suraj Prathik Kumar (FCS) <suraj16101@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: key E4D7C2229116D1DB marked as ultimately trusted
gpg: revocation certificate stored as '/Users/surajprathikkumar/.gnupg/openpgp-revocs.d/8A72410ADDE3C90222673AE3E4D7C2229116D1DB.
rev'
public and secret key created and signed.

pub   rsa4096 2018-11-14 [SC] [expires: 2021-11-13]
      8A72410ADDE3C90222673AE3E4D7C2229116D1DB
uid                      Suraj Prathik Kumar (FCS) <suraj16101@iiitd.ac.in>
sub   rsa4096 2018-11-14 [E] [expires: 2021-11-13]

Suraj-P-Kumar:~ surajprathikkumar$
```

2. **gpg --encrypt --sign --armor -r suraj16101@iiitd.ac.in File**



3. Yes, I can decrypt the file with “gpg -d File.asc”
I can decrypt it as I do have the private key that was associated to the public key that was used to generate the File.asc.
If someone else has the File they cannot decrypt it as they won’t have the private key.

Question 2

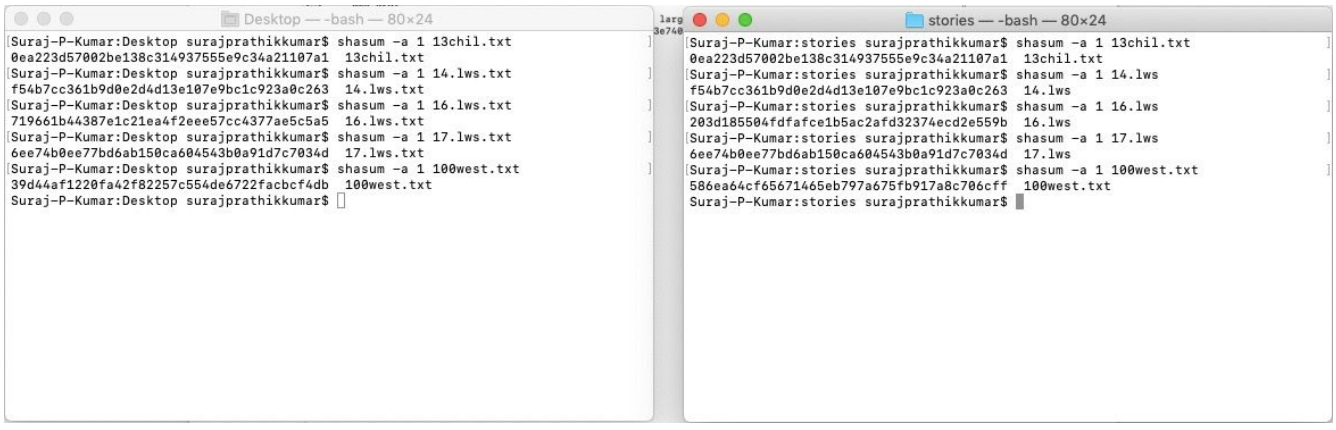
1. Commands -
    - **md5 <filename>**
    - **shasum -a 1 <filename>**
    - **shasum -a 224 <filename>**
    - **shasum -a 256 <filename>**
    - **shasum -a 384 <filename>**
    - **shasum -a 512 <filename>**
    - **sha3sum -a <filename>**

Speed
md5 > sha1 > sha224 >sha256 > sha384 >sha512 > sha3

```
Suraj-P-Kumar:Desktop surajprathikkumar$ time md5 large_file.txt
MD5 (large_file.txt) = 4a2a9d7d13218651cf013769a7c660c4

real    0m0.110s
user    0m0.086s
sys     0m0.027s
Suraj-P-Kumar:Desktop surajprathikkumar$ time shasum -a 1 large_file.txt
3148a2bd9a781afa2264f751226179a96a7036e2  large_file.txt

real    0m0.222s
user    0m0.122s
sys     0m0.045s
Suraj-P-Kumar:Desktop surajprathikkumar$ time shasum -a 224 large_file.txt
3bd17115f34bda352db8ddf2d4685946a6ffabaff0db1a4466ed3f1e  large_file.txt

real    0m0.233s
user    0m0.210s
sys     0m0.019s
Suraj-P-Kumar:Desktop surajprathikkumar$ time shasum -a 256 large_file.txt
080e26f5b88f2f030a26a2b55654218bf4a960bb643f14772f5139b23620159c  large_file.txt

real    0m0.237s
user    0m0.213s
sys     0m0.021s
Suraj-P-Kumar:Desktop surajprathikkumar$ time shasum -a 384 large_file.txt
a50f7071b71a3b395517552558486ef9e8e081e16a059bb6c90d54f6308126accccdd6a4521be5828f8aa78b24d16904  large_file.tx
t

real    0m0.181s
user    0m0.158s
sys     0m0.019s
Suraj-P-Kumar:Desktop surajprathikkumar$ time shasum -a 512 large_file.txt
68b3f57471339766e83890252a7e5880706ad9f8ba317cbb4946936249b3e74036634ff9bf167528373927f363d9e023ec9a183a0beebf5
6f5aeafb398d5d403  large_file.txt

real    0m0.196s
user    0m0.164s
sys     0m0.024s
Suraj-P-Kumar:Desktop surajprathikkumar$
```

2. a) File Names 14.lws.txt and 100west.txt have were modified by the third party.2 out of 5 were modified.

Methodology - Checksum used to verify the files integrity as it is very unlikely that any two non-identical files in the real world will have the same MD5 hash and Sha1 hash unless they have been specifically created to have the same hash.



b) The detection technique might also fail in case of MD5 collisions, it is possible for the third party to create a second file with the same checksum, so this technique cannot protect against some forms of malicious tampering.

c) Cryptographic Hash functions are mostly used instead of hash functions to solve 3 fundamental problems associated with hash functions.
Pre-image resistance, Second Pre-image resistance, Collision resistance (Hard to find messages hash(m1) = hash(m2))
Security property violated is **Collision resistance.**
Since MD5 collisions are possible in 2b and they take away the use case of cryptographic hash functions over hash functions.
Due to this Priyab will have to use some other technique to find the integrity of the files.

Question 3

1. The changes in the functions are -

```
static int is_registered(char *uname) {

        FILE *db_file;

        if ((db_file = fopen("user_db.txt", "r")) == NULL) {
                printf("Error\n");
                exit(1);
        }

        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);


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

        fprintf(db_file, "%s", uname);
        fprintf(db_file, " ");
        fprintf(db_file, "%s\n", passwd);

        fclose(db_file);


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

        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)
                {
                        if (strcmp(strdup(passwd), strdup(password)) == 0) {
                                return 1;
                        }
                }
        }

        fclose(db_file);

        return 0;
}
```

```
int main(int argc, char *argv[])
{

        int register_flag = 0;

        if (strcmp(argv[1], "-r") == 0) {
                printf("Register User\n");
                register_flag = 1;
        }

        else if (strcmp(argv[1], "-a") == 0)
        {
                printf("Authorise\n");
                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");
                }
        }

        return 0;
}
```

The code hashes the password using a salt. The file permissions have also been defined as above.

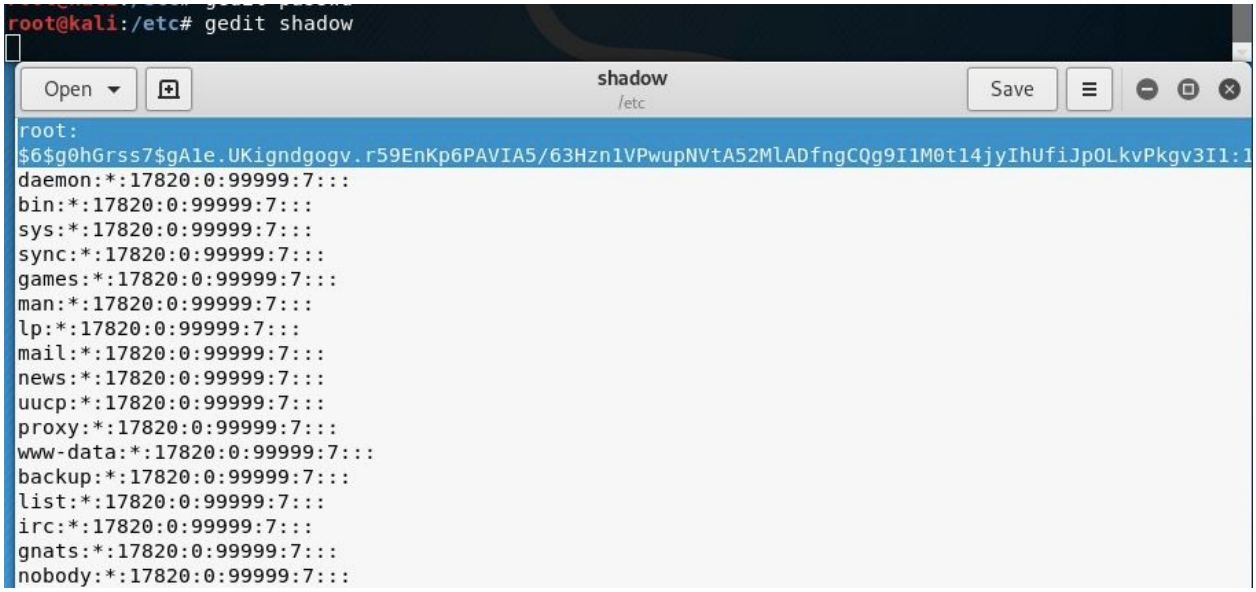
2. The following code is used for the Brute force attack. The password list is saved in a file -

commands to run -

gcc -o ./attack brute_force.c
./attack <user>

```
#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)
        {
            printf("Error");
            exit(1);
        }
        else
        {
            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);
            }
        }

    }

        return 0;
}
```

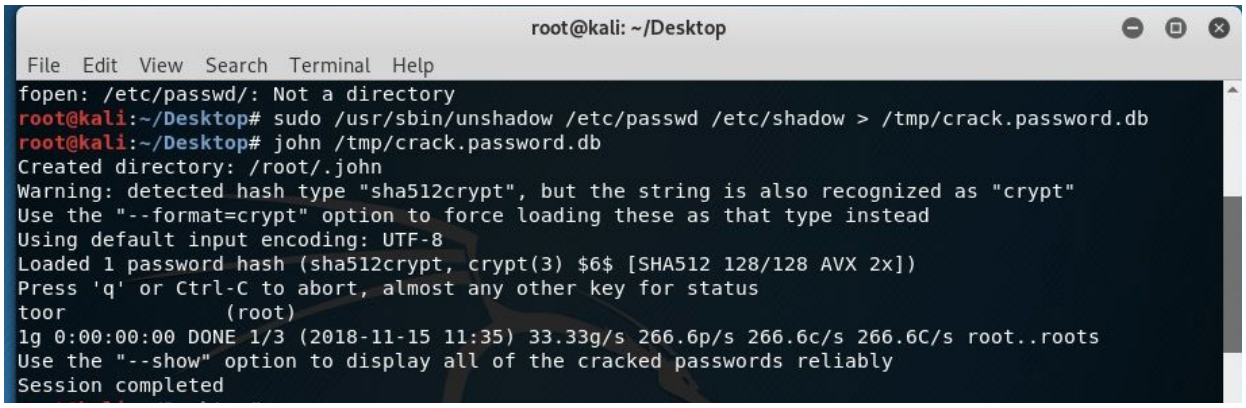
3. The password and the username is saved in the /etc/shadow file.



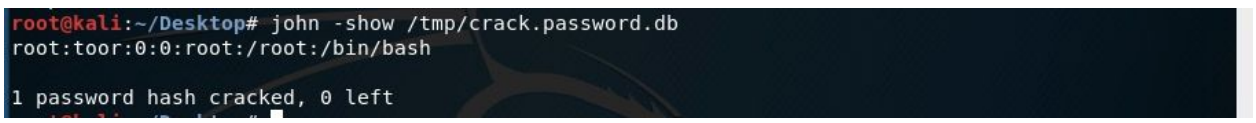
The passwd stores general user information and shadow stores user passwd info. passwd is the file where the user information and in shadow file important information (like an encrypted form of the password of a user, the day the password expires) are stored.

5. Command - **sudo /usr/sbin/unshadow /etc/passwd /etc/shadow > /tmp/crack.password.db**

For cracking Password: john /tmp/crack.password.db



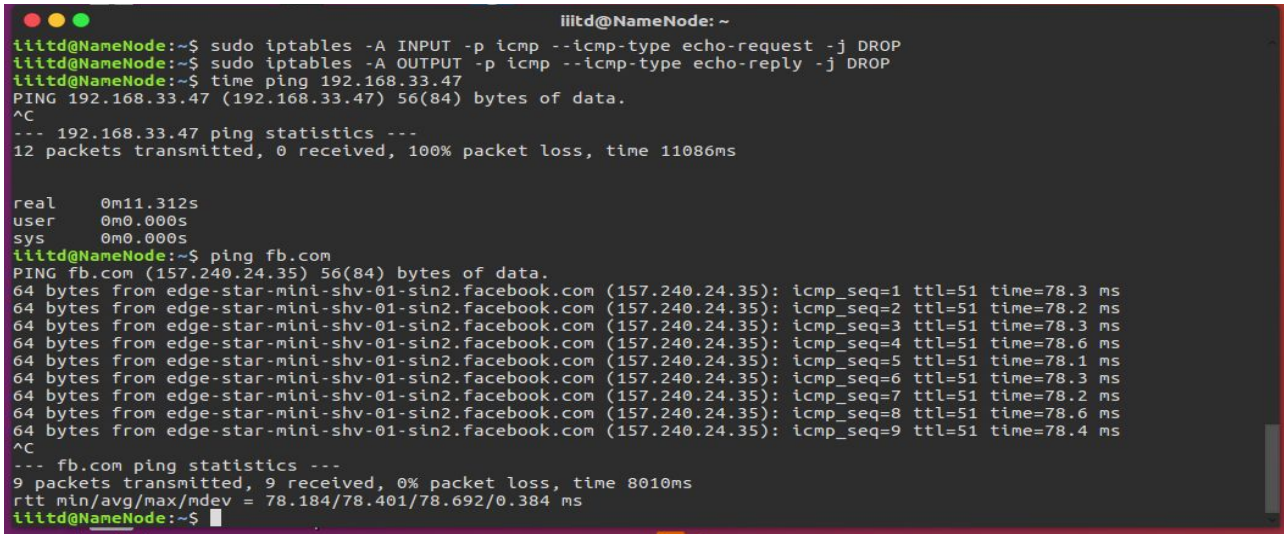
To show the cracked File: john -show /tmp/crack.password.db



Question 4

Note: Since i have a mac the 4th Question is done on LAB PC as the iptables dont work on it.

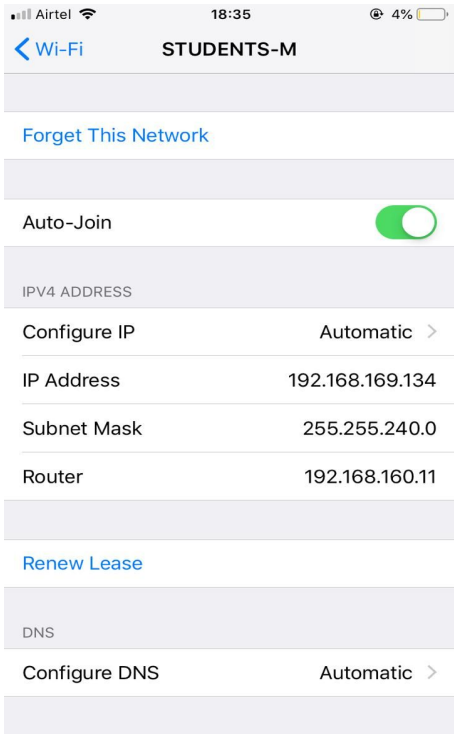
1. **a) Command -**
   Sudo iptables -A INPUT -p icmp --icmp-type-echo-request -j DROP
   Sudo iptables -A OUTPUT -p icmp --icmp-type-echo-reply -j DROP



Checking if the ping to system is blocked. “**ping IP address**”
No response and checked for 11sec.

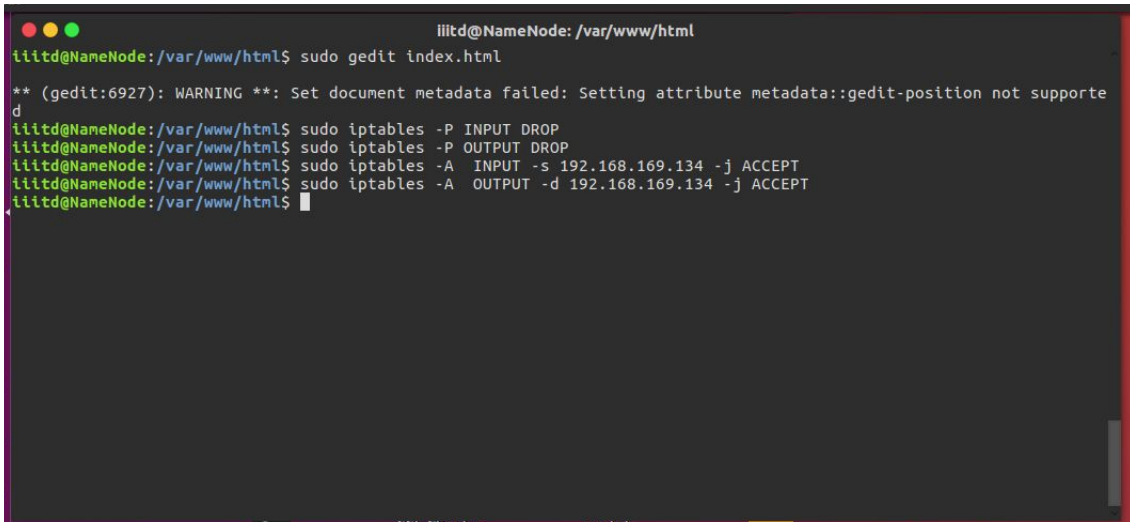
Checking if the ping to any other devices is working by hitting “**ping fb.com**” and it is working.

b) IP address of my Phone



Command for dropping all the packets

sudo iptables -P INPUT DROP && sudo iptables -P OUTPUT DROP



Hosted the HTML webpage on apache2 on the Lab pc with IP address - 192.168.33.47

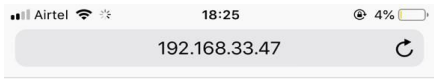
After Dropping all the packets i could not get the webpage on my phone or laptop

After that i allowed my IP address (phone) .

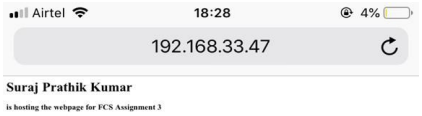
Command - **sudo iptables -A INPUT -s 192.168.169.134 -j ACCEPT**

sudo iptables -A OUTPUT -d 192.168.168.134 -j ACCEPT

Before Allowing



After Allowing



2. a) **File.txt has been uploaded along with the document.**

IP - 192.168.33.47
Subnet mask for lab B519- 255.255.240.0/20

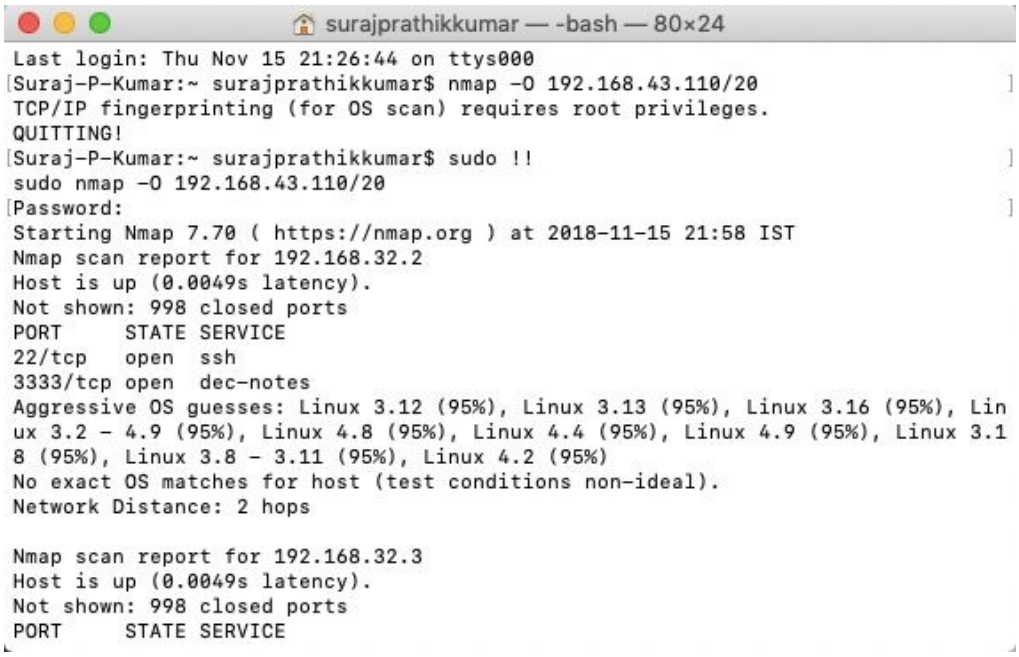
subnet ID - 192.168.33.47/20.

ssh run on PORT - 22

Command - **nmap -sS 192.168.33.47/20 -p 22 > file.txt**

b) **Complete File has been uploaded with pdf**

Command - nmap -O 192.168.43.110/20 > File_OSFingerprint.txt



Stats - 168 out of which only 6 were found to be windows and rest linux users thereby proving these stats wrong of 70% windows and 20% Linux

3. Install openvpn
Commands - sudo apt-get install openvpn easy-rsa

Setup ca Directory

make-cadir ~/openvpn-ca
nano vars

Then you create your certificate

Build the Certificate Authority

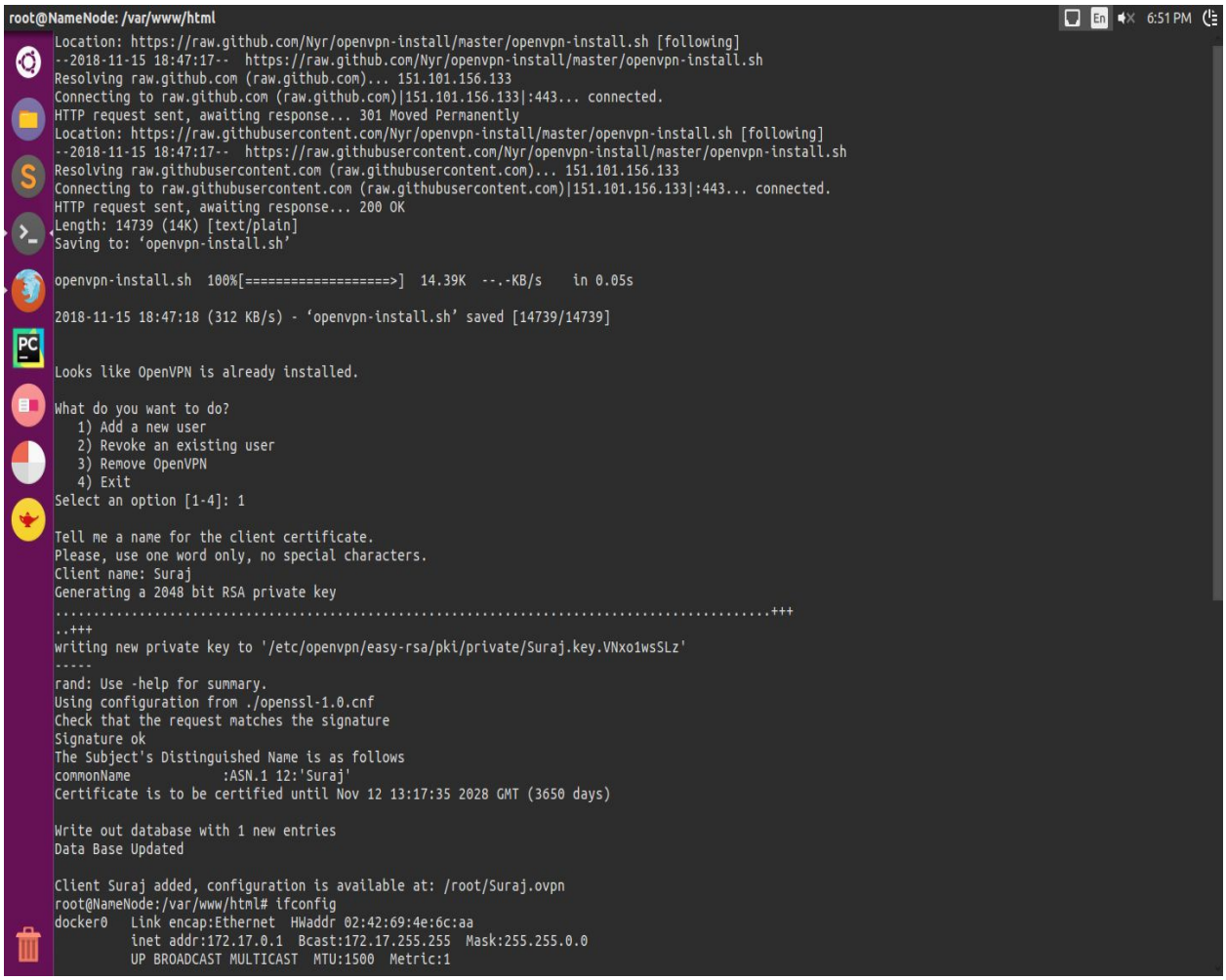
cd ~/openvpn-ca
source vars

Create the server Key

./build-key-server server

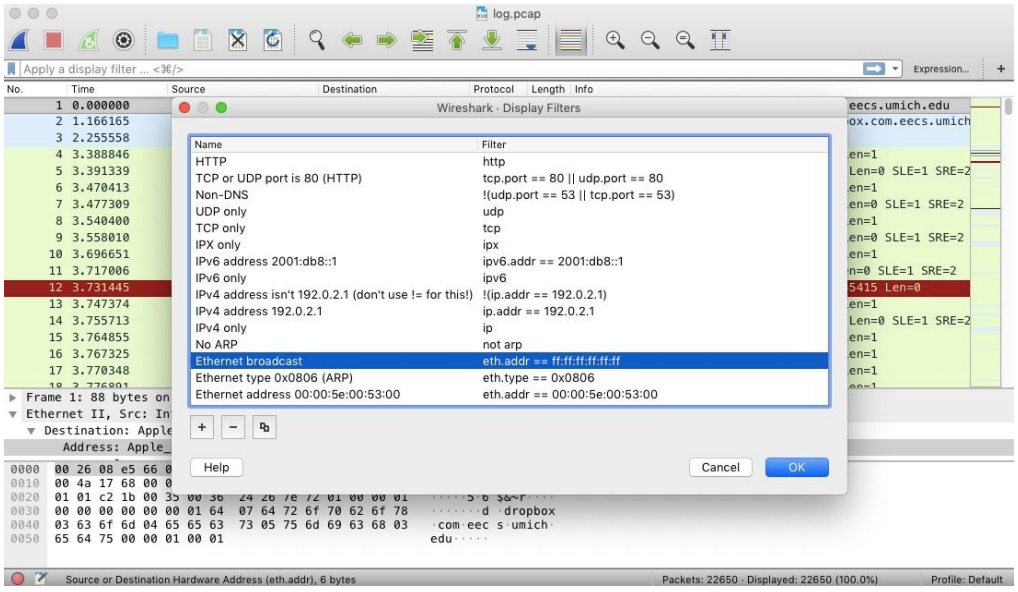
Generate a Client Certificate and Key Pa

sudo nano /etc/sysctl.conf

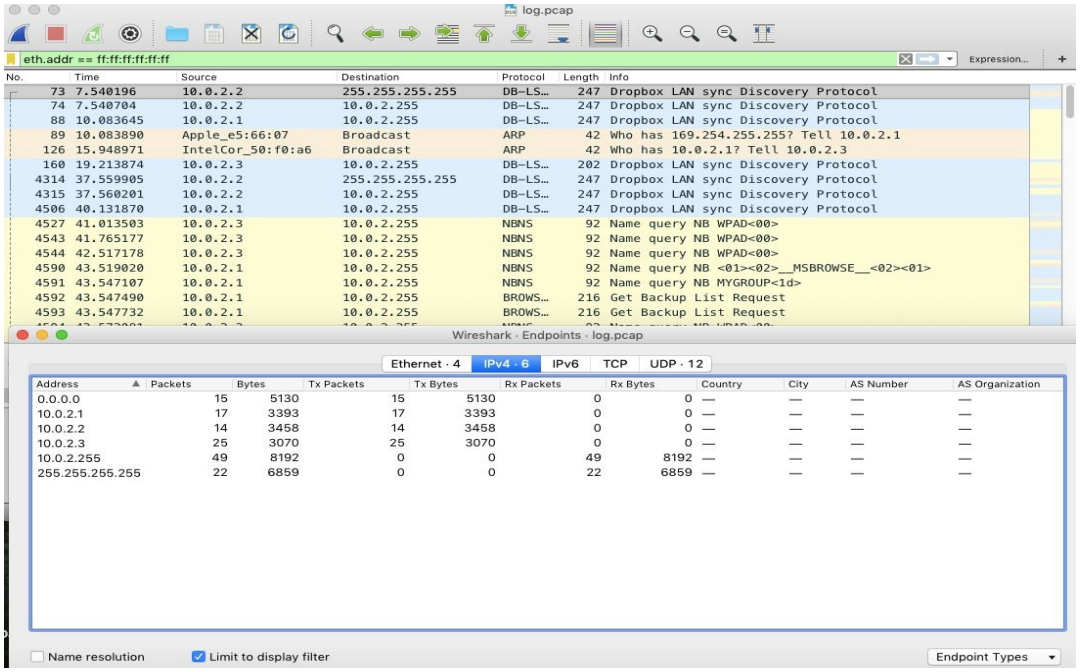


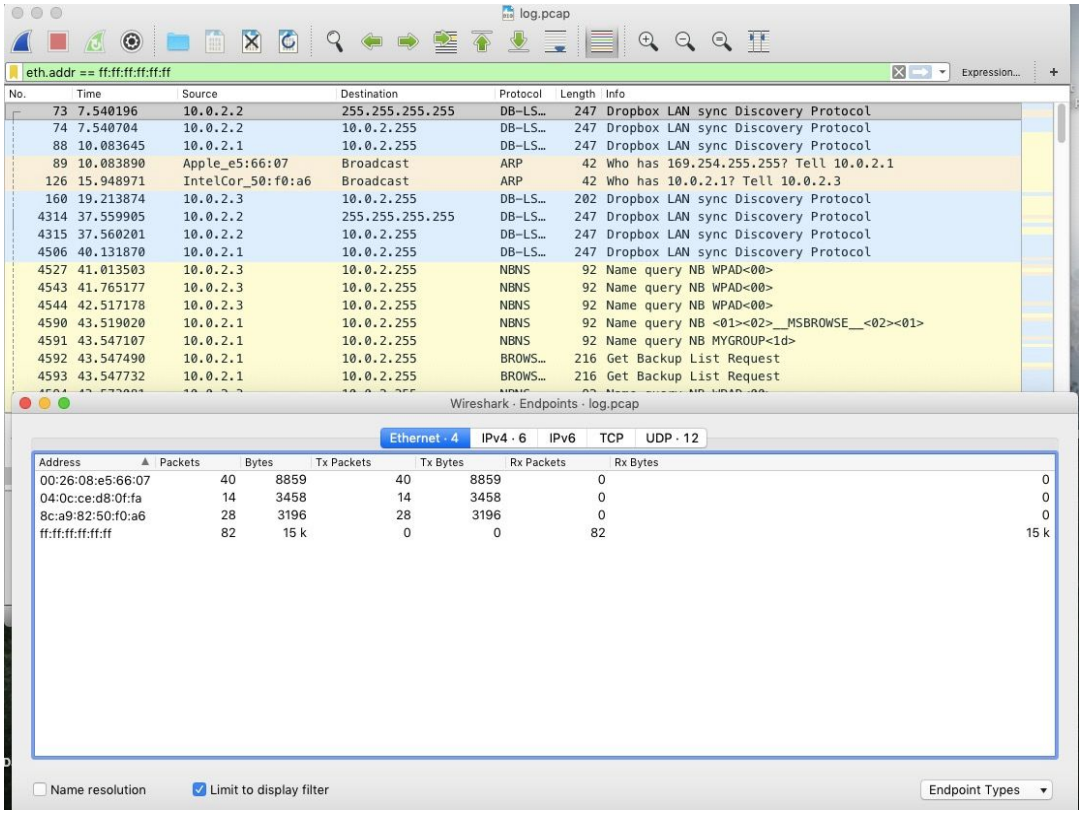
Question 5

1. **a)** The ethernet broadcast is eth.addr == ff:ff:ff:ff:ff:ff



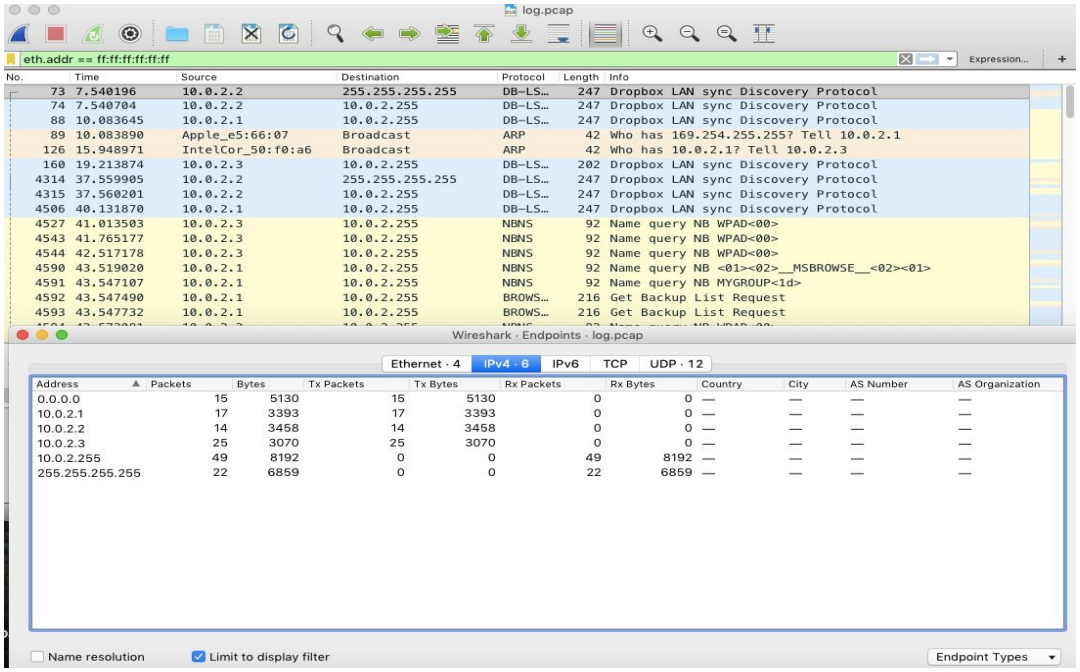
Associated IP and Mac Addresses are :





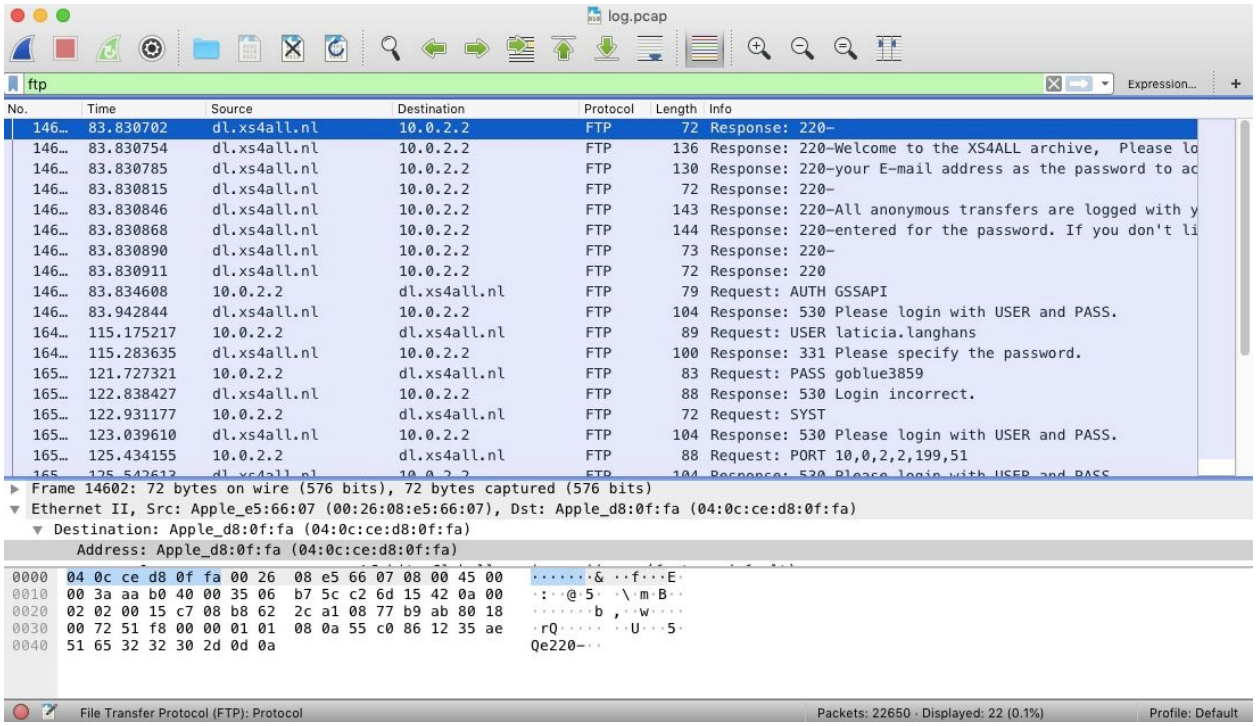
b) Since the number of host is 3 as seen in the picture. This seems like a home network or of a small institute.The websites visited frequently include social

networking sites like facebook and some educational sites.



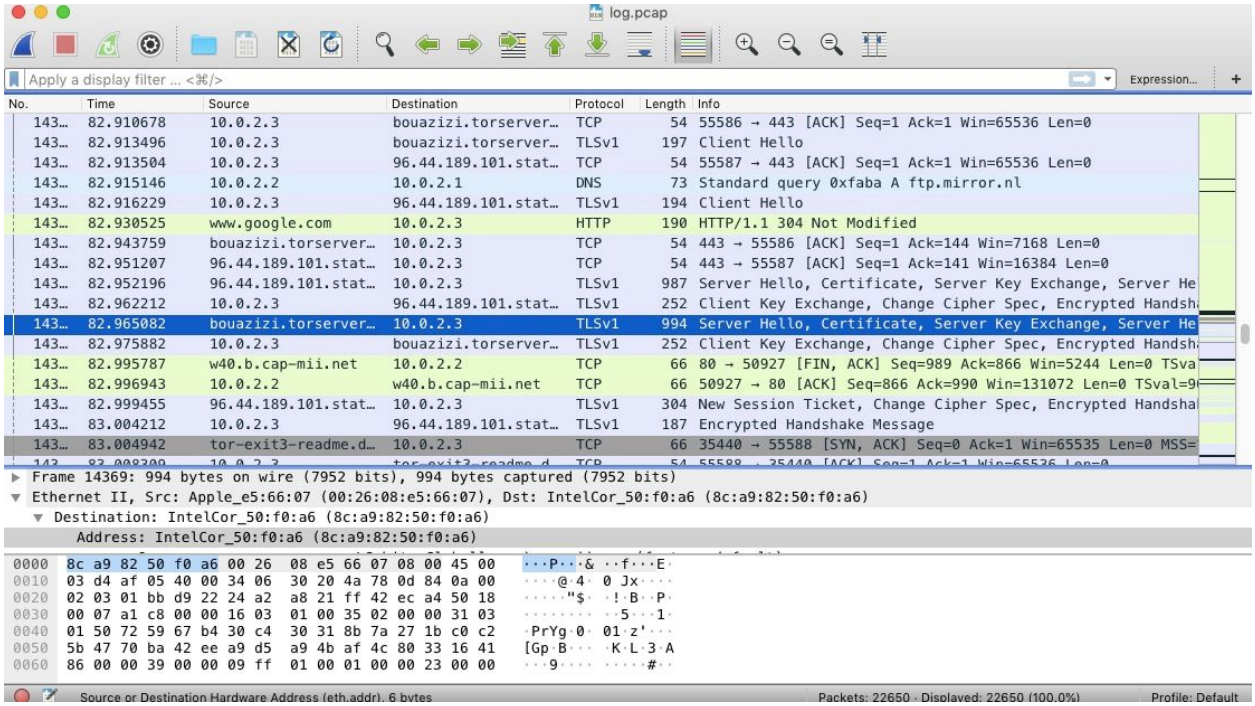
c) The DNS host name is xs4all and 192.109.21.66

All the ftp requests are sent to this ip by the other hosts and they get a response like Login incorrect etc.



FTP - It is not the correct method to transfer packets as no encryption is there therefore anyone can intercept messages and This is also susceptible to man in the middle attack.
The more secure method is HTTPS and FTPS as it does encryption

d) Another HTTPS site other than facebook is bouazizi.torservers.net



Trust, Integrity and SEO is there with https.
It is better as it is not susceptible to man in the middle attack as it does encryption and provides proper security protocols.

e) Even though logins are processed over HTTPS but the security property is violated in case of the facebook at the time of authentication of the user. As the user doesn't have to provide a valid proof of himself. NO CA assigned certificates to the user. A user can have multiple accounts and claim to be different in all of them. With only credentials fb can verify the user and if they are compromised no check is there.

2. **Code has been uploaded with the pdf**
   To Run : install dpkt and run on python 3 and above

The eth data is extracted from the file to find IP Packets. This IP packets are filtered to get TCP packets. TCP packets with SYN packets with 2 way handshake are filtered and printed in the output.

10.0.2.3 and **10.0.2.2 are the malicious IP's**