**信息安全实验报告**

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

**孙铁**

**SA20225414**

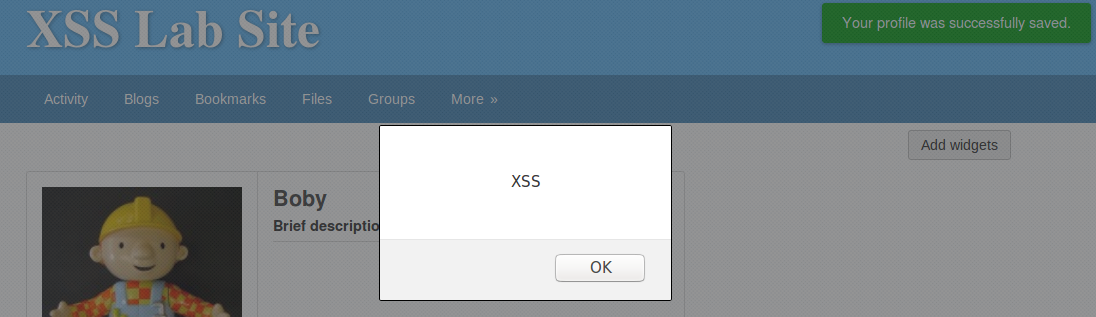
**Task 1**

打开http://www.xsslabelgg.com进入Elgg页面；登录Boby账号。

修改Boby的个人简介：



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：

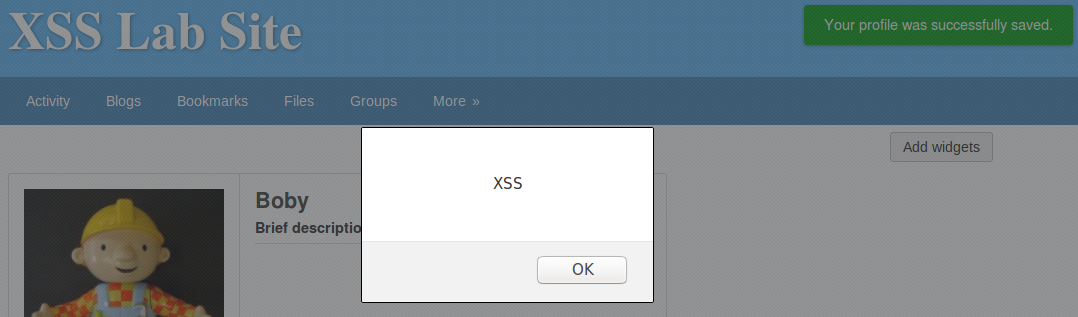
在/var/www/XSS/Elgg/路径下创建文件scripts1.js：



修改Boby的个人简介：

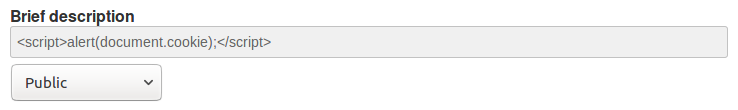


保存之后返回主页后同样会弹出窗口：

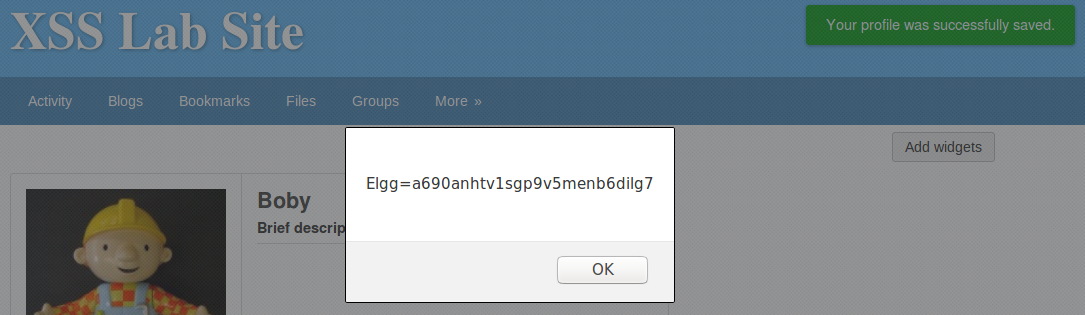


**Task 2**

修改Boby的个人简介：

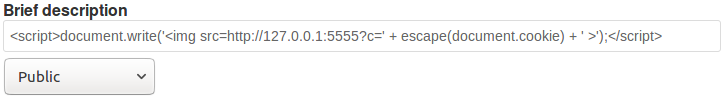


保存之后返回主页，弹出窗口显示用户cookie：



**Task 3**

修改Boby的个人简介：

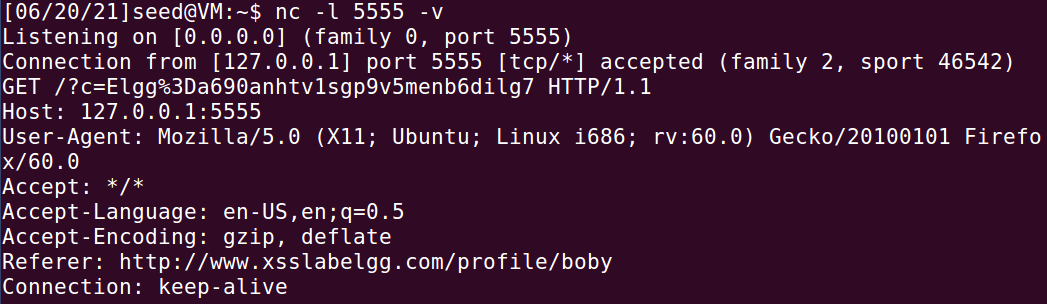


JavaScript代码的作用是将当前用户的cookie发送给攻击者主机的5555号端口。这里使用当前VM作为攻击者主机，使用本地IP地址 127.0.0.1。

在攻击者主机监听5555号端口：

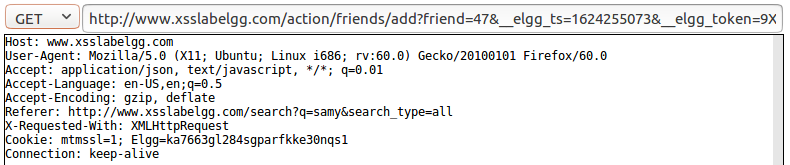


保存修改的个人简介并返回主页，此时攻击者终端成功获得受害者用户cookie：



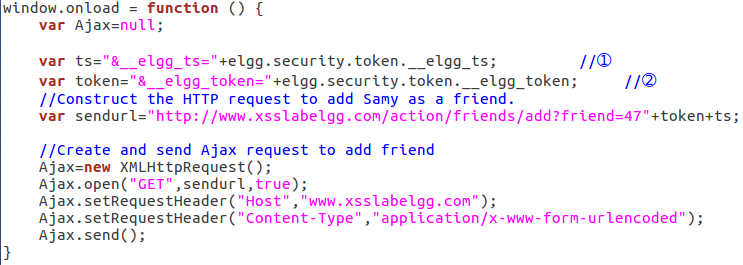
**Task 4**

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



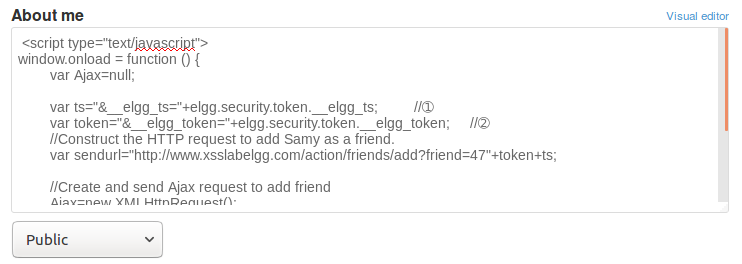
由此可以得到Samy的GUID为47。

本次攻击需要使用的JavaScript代码如下：

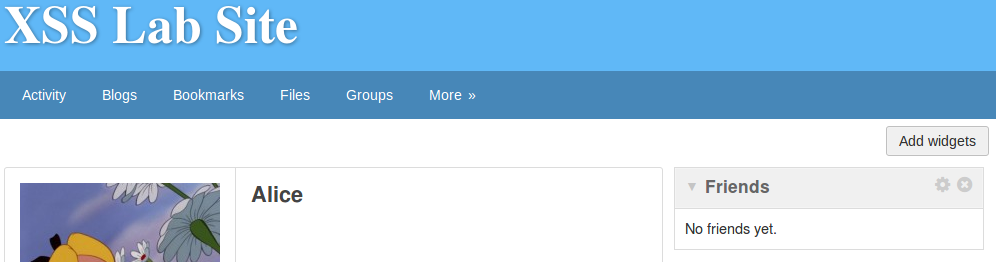


代码的作用是伪造一个向Samy发送好友申请的HTTP请求，并使用Ajax在后台发送。

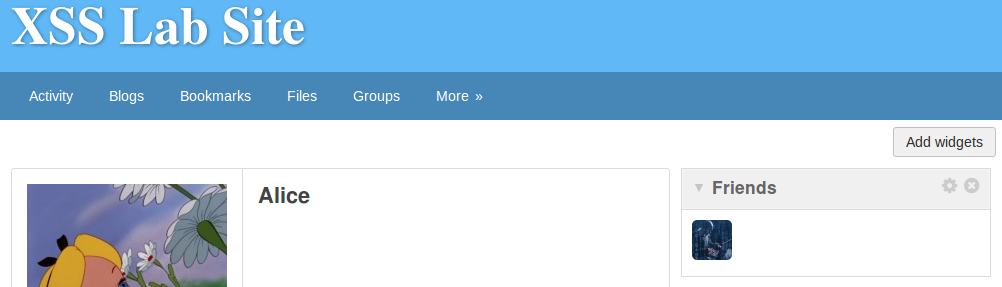
登录Samy账号，修改“About me”区域，点击右上角的Edit HTML来进行明文编辑，将JavaScript代码复制进来：



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：**

➀与➁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，这就让它可以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：

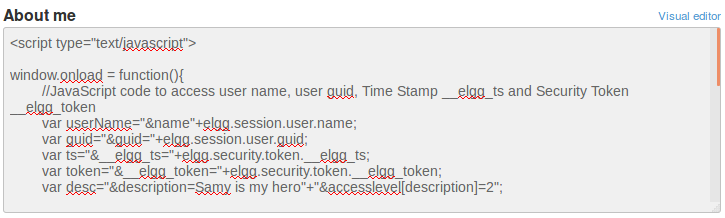


为了修改个人简介，需要使用的JavaScript代码如下：

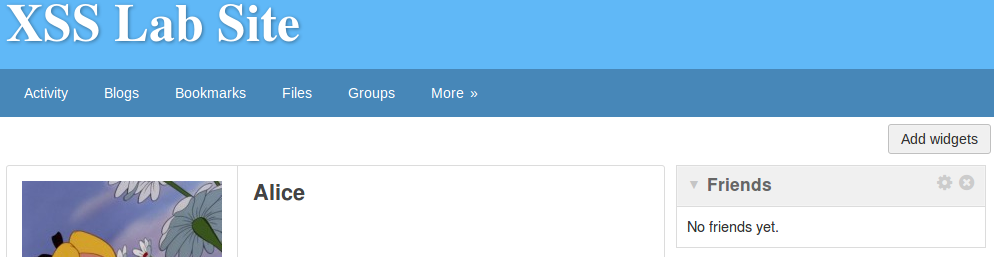


代码的作用是伪造一个修改个人信息的Ajax请求。

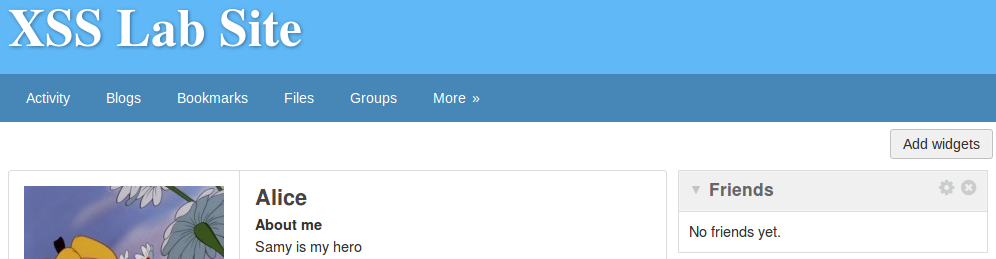
登录Samy账号，修改“About me”区域，点击右上角的Edit HTML来进行明文编辑，将JavaScript代码复制进来：



保存之后退出Samy账号，登录Alice账号，此时Alice个人介绍为空：



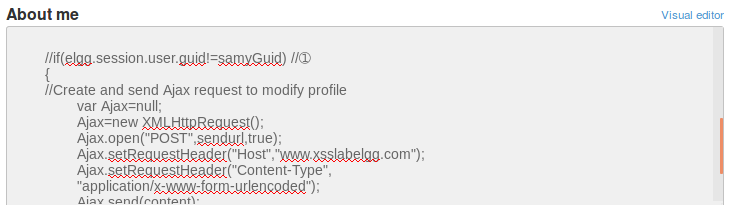
在Alice账号中搜索Samy并打开其主页，返回Alice主页发现个人简介被修改：



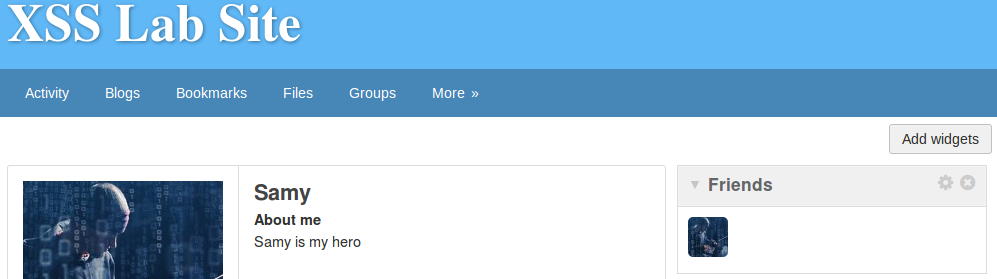
**Question3：**

➀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➀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：



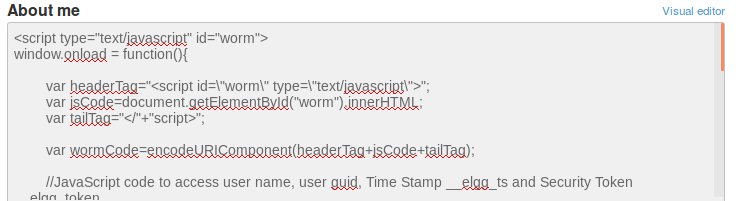
**Task 6**

使用JavaScript代码如下：

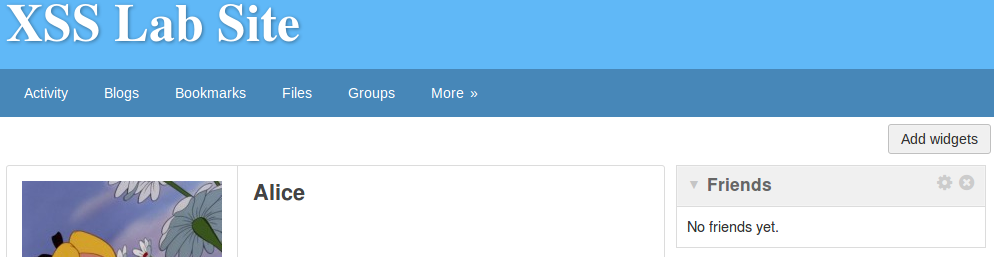


代码通过DOM方法实现了JavaScript代码的自我传播。在页面加载完成之后，浏览器会把页面的内容存放在名为DOM的树状数据结构中。将script代码块取名为“worm”，使用document.getElementById函数就可以找到此代码块，用它的innerHTML属性获取代码的内容，再加上一对script开始标签和结束标签就构成了一份恶意代码拷贝。然后构造一个Ajax请求将“Samy is my hero”和一份恶意代码的拷贝放入受害者的个人介绍，使得受害者主页也能感染他人，这就实现了自我传播。

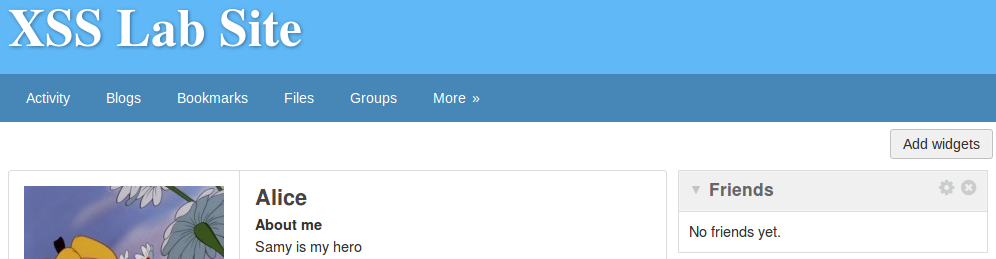
登录Samy账号，修改“About me”区域，点击右上角的Edit HTML来进行明文编辑，将JavaScript代码复制进来：



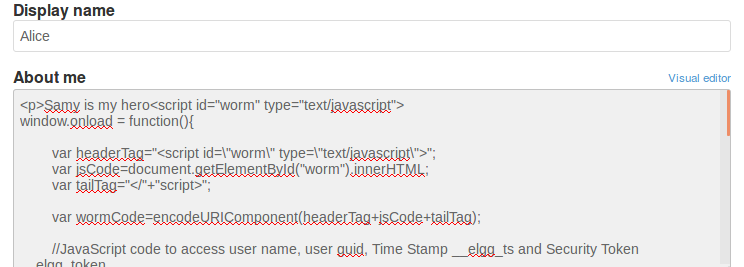
保存之后退出Samy账号，登录Alice账号，此时Alice个人介绍为空：



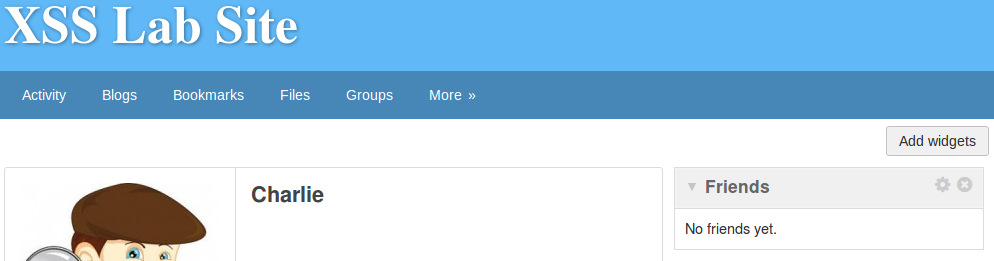
在Alice账号中搜索Samy并打开其主页，返回主页发现个人简介被修改：



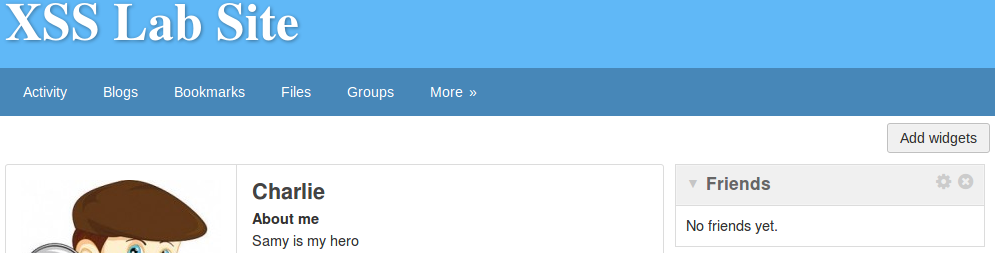
进入Alice的个人介绍的修改页面，点击右上角的Edit HTML来使用明文编辑器查看“About me”区域，发现成功复制了恶意代码：



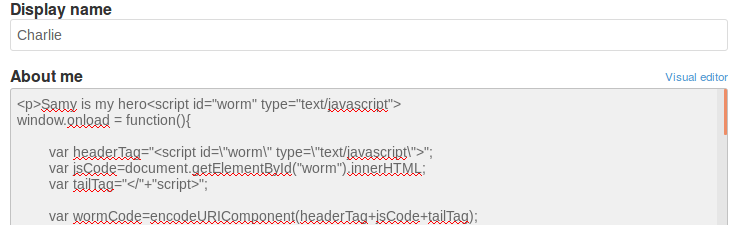
退出Alice账号，登录Charlie账号，此时Charlie个人介绍为空：



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：



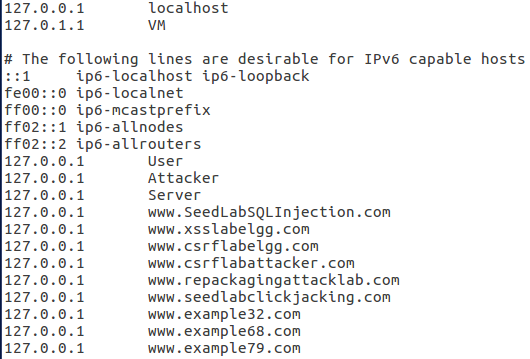
进入Charlie的个人介绍的修改页面，点击右上角的Edit HTML来使用明文编辑器查看“About me”区域，发现成功复制了恶意代码：



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：

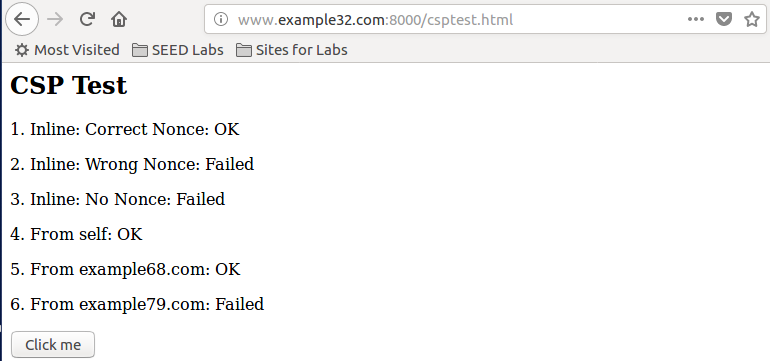


根据http\_server.py中的Content-Security-Policy字段可以得到CSP规则：

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。

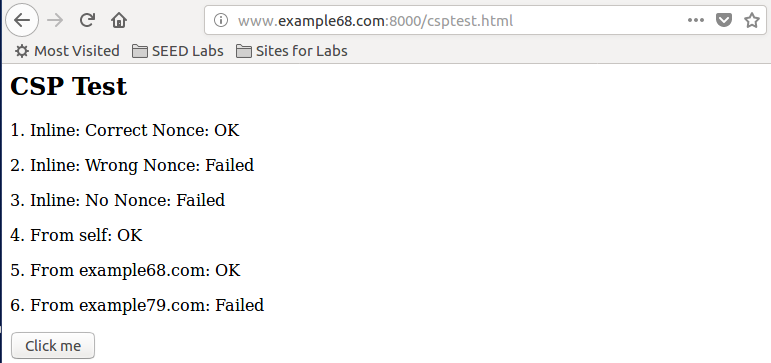
**Task 7.1**

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



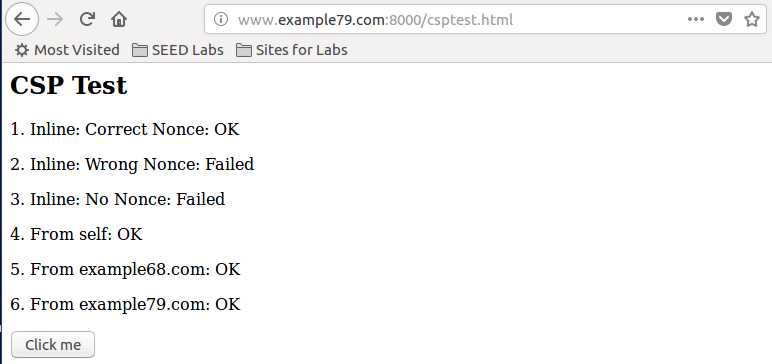
根据CSP规则：只有Nonce正确的JavaScript代码可以执行，所以area1可以执行，area2和area3不能；area4来自网页同一网站，可以执行；area5来自CSP规则允许的example68.com网站，可以执行；area6来自不同网站且没有CSP允许，不能运行。

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。

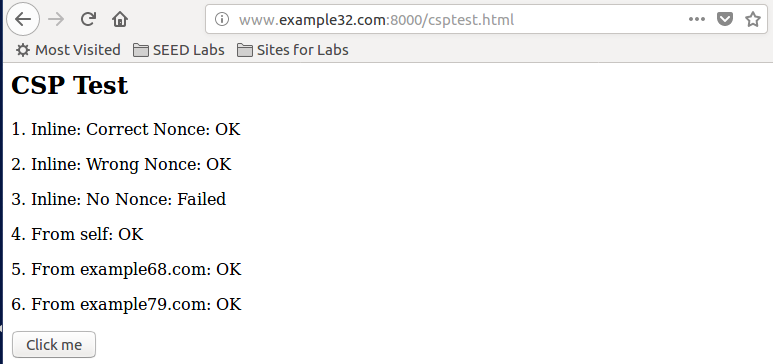
**Task 7.2**

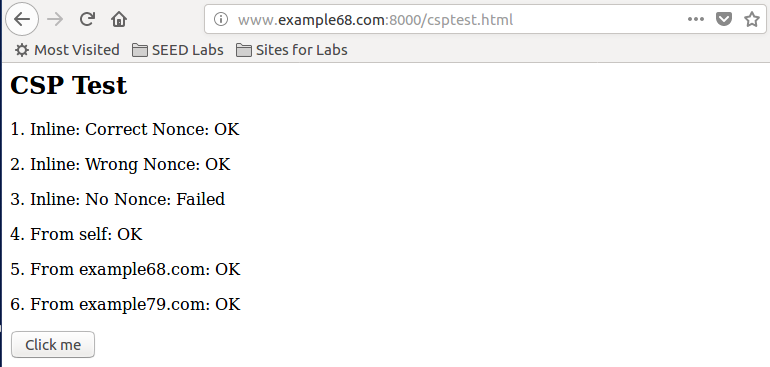
修改http\_server.py：

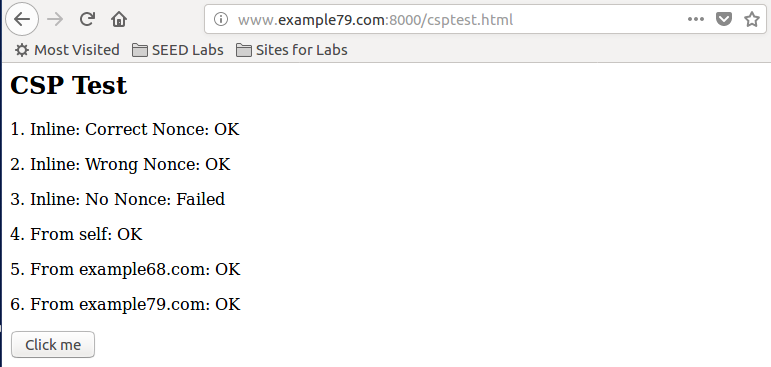


加入规则：允许包含Nonce=2rB3333的代码执行；允许来自example79.com：8000的代码执行。

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。