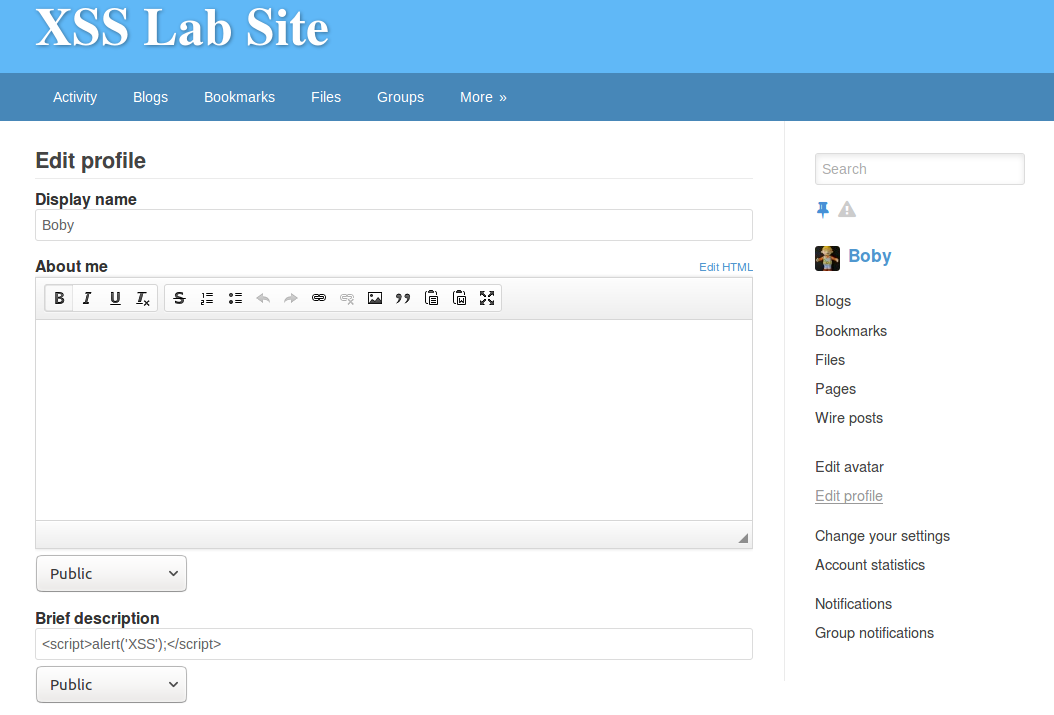
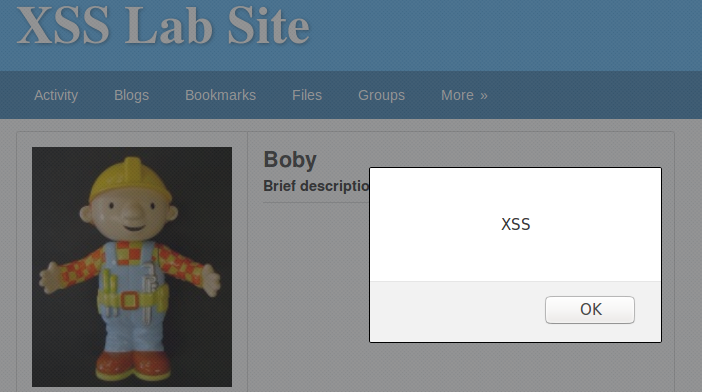
**Lab[5]-report**

57118123刘康辉

**Task 1**

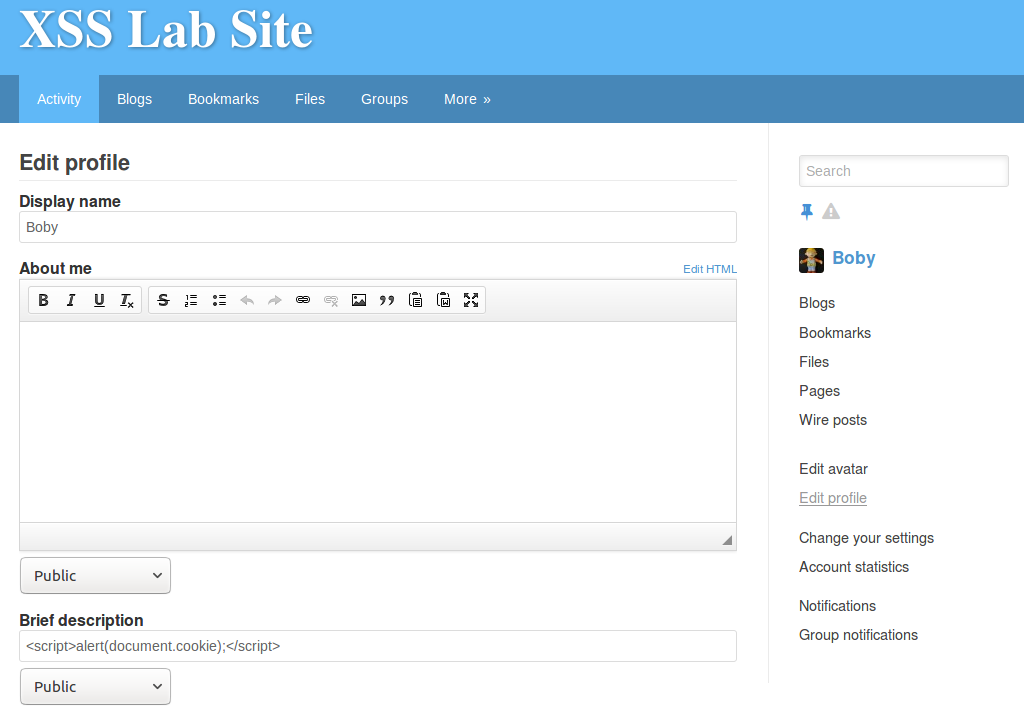


登录Boby的账户，修改签名如上图所示。

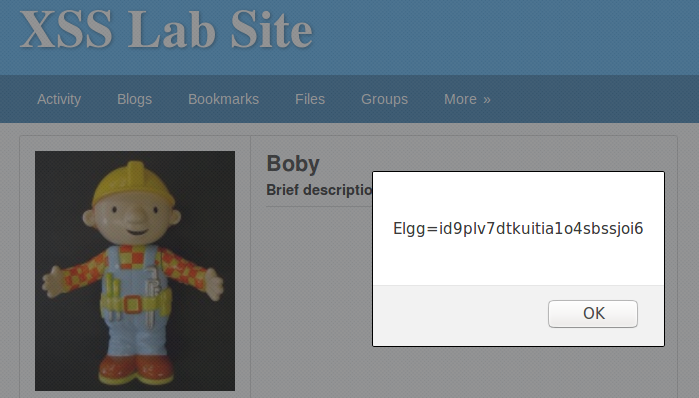


登录Alice的账户，点击查看Boby的个人信息界面，出现警报窗口。根据实验结果可知，Javascript语句成功执行。

**Task 2**

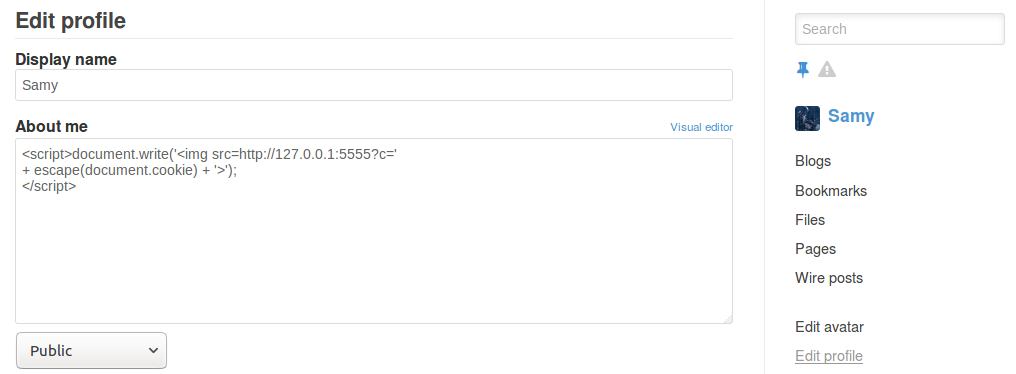


登录Boby的账户，修改签名如上图所示。

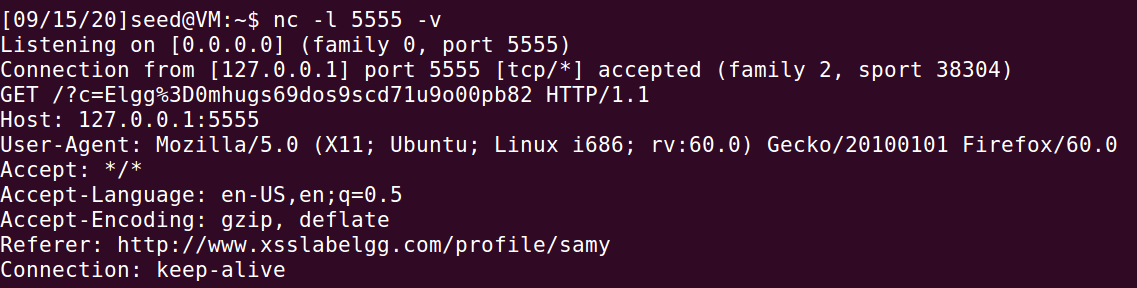


登录Alice的账户，点击查看Boby的个人信息界面，出现警报窗口并显示用户的cookies。根据实验结果可知，Javascript语句成功执行。

**Task 3**

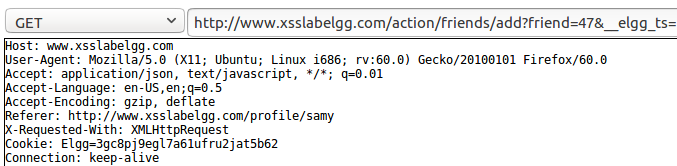


登录Samy的账户，进入Edit HTML界面修改签名如上图所示。



在终端输入指令监听端口5555的信息，然后登录Alice的账户，点击查看Samy的个人信息界面，发现获取的信息如上图所示。根据实验结果可知，Javascript语句成功执行。

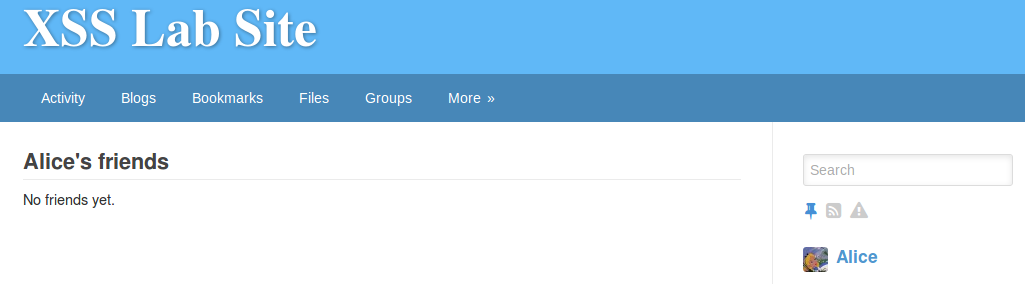
**Task 4**

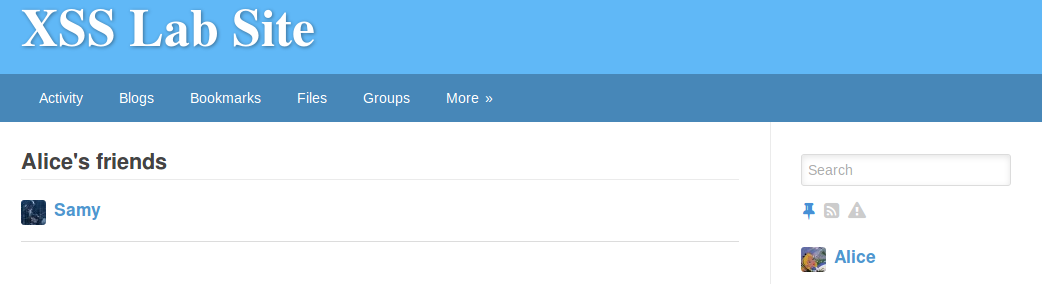


抓取的添加好友的HTTP GET请求如上图所示，可知Samy的用户id为47。



登录Samy的账户，进入Edit HTML界面修改签名如上图所示。

登录Alice的账户，进入好友界面发现未添加好友。



点击查看Samy的个人信息界面，然后返回自己的好友界面，发现已经添加了Samy为好友。根据实验结果可知，XSS攻击成功。

**Question 1**

Line1和Line2这两行的代码是为了获取秘密令牌\_elgg\_ts和\_elgg\_token，因为发送的请求只有提供正确的秘密令牌才不会被拒绝，才能实现XSS攻击，否则它会被丢弃。

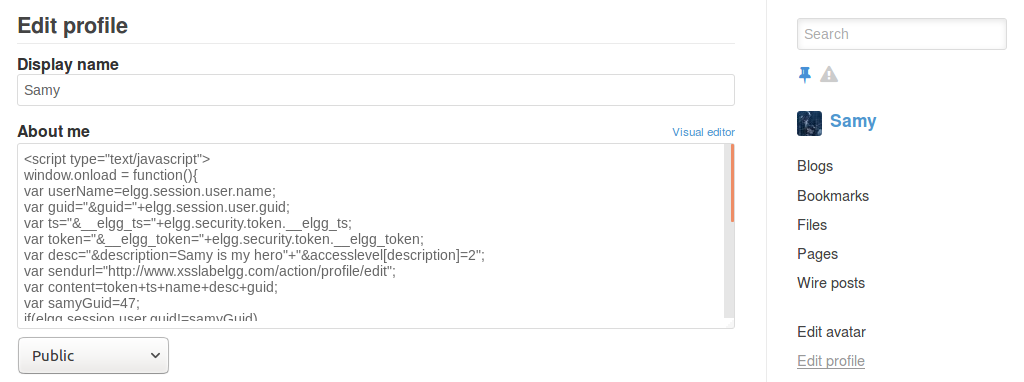
**Question 2**

能；可以使用浏览器扩展来删除HTTP请求中的格式化数据。

**Task 5**



登录Alice的账户，修改签名如上图所示。



登录Samy的账户，进入Edit HTML界面修改签名如上图所示。

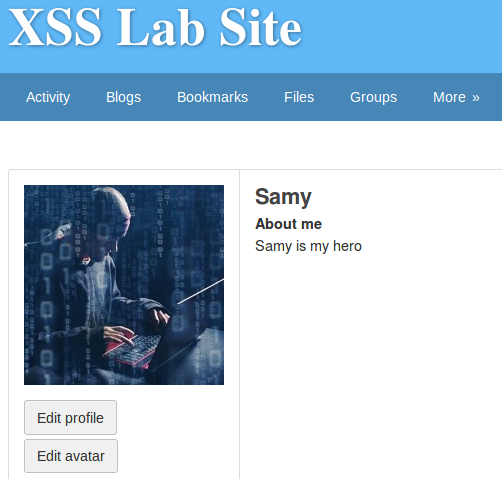


登录Alice的账户，点击查看Samy的个人信息界面，然后返回自己的个人信息界面，发现签名已经被修改为Samy is my hero。根据实验结果可知，XSS攻击成功。

**Question 3**



登录Samy的账户，进入Edit HTML界面修改签名，删除Line1这一行的代码，结果如上图所示。



返回Samy的个人信息界面发现自己签名已经被修改为Samy is my hero，而其他用户点击查看Samy的个人信息界面后签名不会被修改。原因是Line1的代码负责判断用户是否为Samy本人，避免自己的签名被修改。现在删除该行代码，返回Samy的个人信息界面时自己的签名就已经被修改，之前的代码被覆盖，因此无法对其他用户实现XSS攻击。

**Task 6**

**DOM Approach**



登录Samy的账户，进入Edit HTML界面修改签名如上图所示。

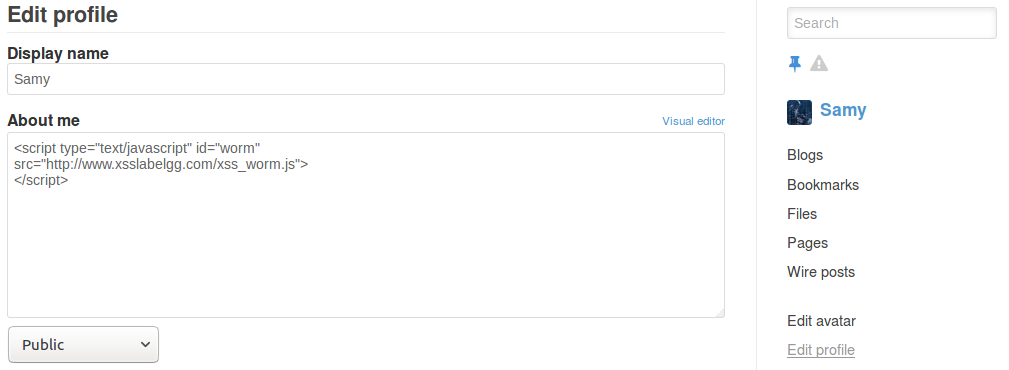


登录Alice的账户，点击查看Samy的个人信息界面，然后返回自己的个人信息界面，发现签名已经被修改为Samy is my hero，并且在Edit HTML界面内存在Javascript代码。



登录Boby的账户，点击查看Alice的个人信息界面，然后返回自己的个人信息界面，发现签名已经被修改为Samy is my hero。根据实验结果可知，通过DOM方法，XSS蠕虫攻击成功并且具有自我传播能力。

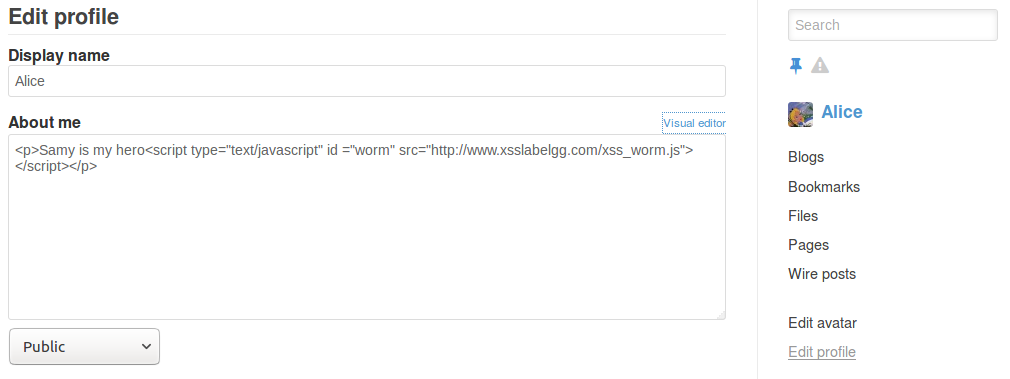
**Link Approach**



登录Samy的账户，进入Edit HTML界面修改签名如上图所示。



在/var/www/XSS/Elgg下创建文件xss\_worm.js，并且添加代码如上图所示。

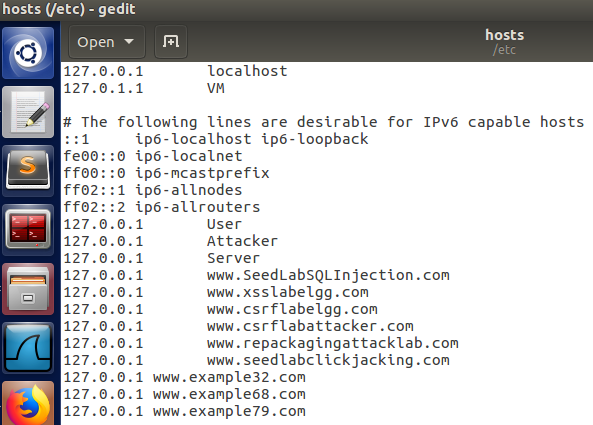


登录Alice的账户，点击查看Samy的个人信息界面，然后返回自己的个人信息界面，发现签名已经被修改为Samy is my hero，并且在Edit HTML界面内存在Javascript代码。



登录Boby的账户，点击查看Alice的个人信息界面，然后返回自己的个人信息界面，发现签名已经被修改为Samy is my hero。根据实验结果可知，通过Link方法，XSS蠕虫攻击成功并且具有自我传播能力。

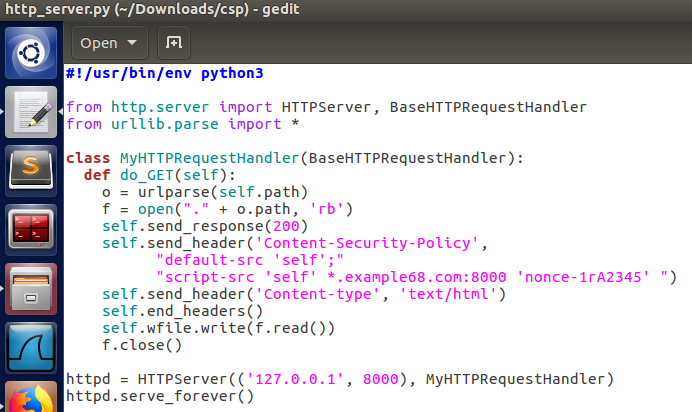
**Task 7**

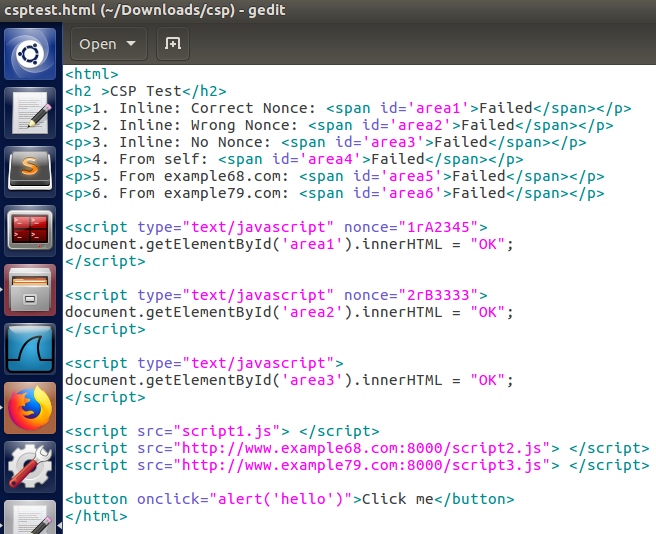


修改/etc/hosts文件，添加语句如上图所示。

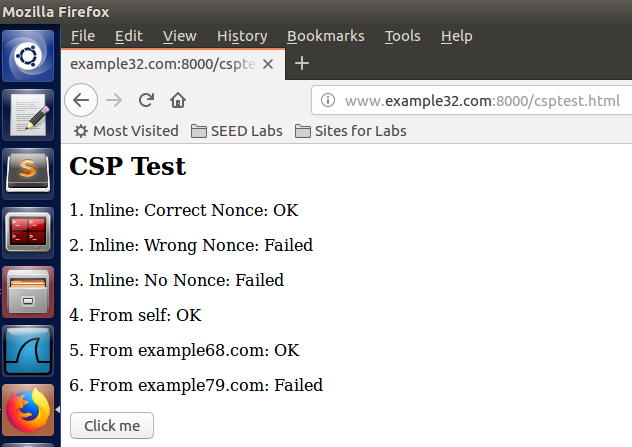


在<https://seedsecuritylabs.org/Labs_16.04/Web/Web_XSS_Elgg/>中下载csp.zip文件，然后进行解压，再运行其中的http\_server.py文件。

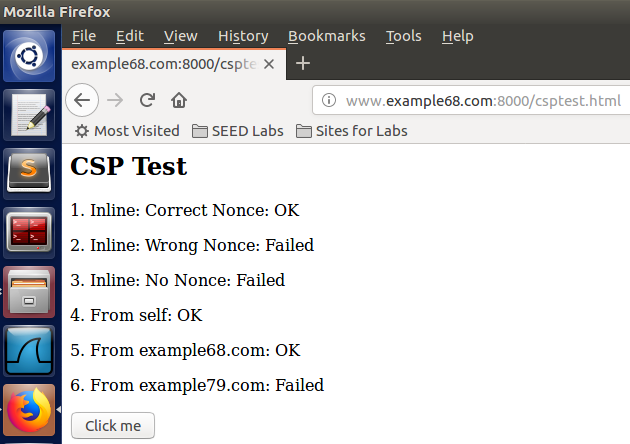




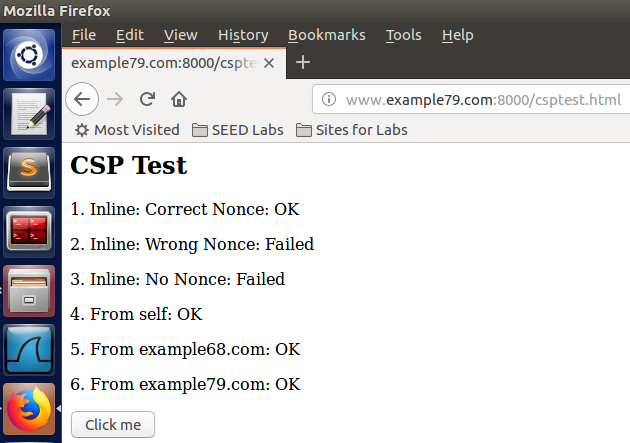
在csp.zip中的http\_server.py和csptest.html文件如上图所示。



在浏览器中输入<http://www.example32.com:8000/csptest.html>，页面如上图所示，Fields 1、4、5显示为OK。

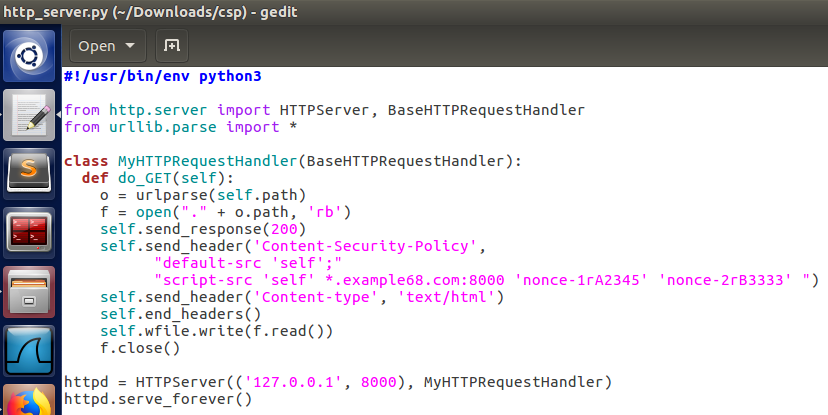


在浏览器中输入<http://www.example68.com:8000/csptest.html>，页面如上图所示，Fields 1、4、5显示为OK。

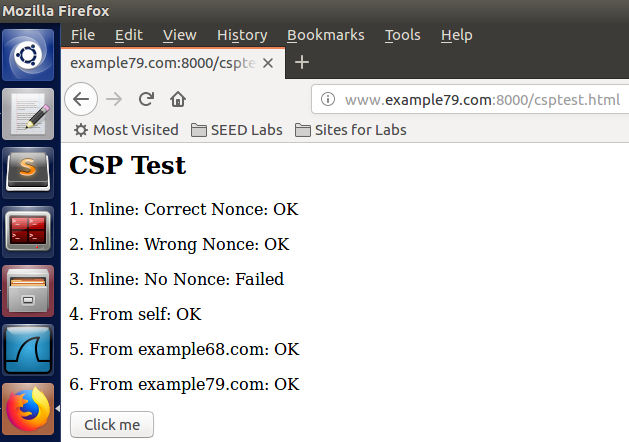


在浏览器中输入<http://www.example79.com:8000/csptest.html>，页面如上图所示，Fields 1、4、5、6显示为OK。

这三个网站显示不同结果的原因是http\_server.py只允许含nonce=1rA2345的代码（对应area 1）、来自example68.com:8000的代码（对应area 5）以及网站自身产生的代码（对应area 4）被执行。



修改http\_server.py文件，在script-src后添加nonce-2rB3333。



重新编译运行修改后的http\_server.py，再次在浏览器中输入<http://www.example79.com:8000/csptest.html>，页面如上图所示，Fields 1、2、4、5、6显示为OK。