# CTFHUB——反射型XSS详解

原创

Wuuconix 于 2021-09-01 02:23:26 发布 1093 收藏 3

文章标签： xss ctf

Wuuconix wanna a girlfriend!



## 背景

本来看ctfhub上有xss的题目，打算好好学习一波，结果点开一看，只有一道题2333。

便现在dvwa上熟悉了一波。所谓反射型是相对于存储型来讲的。

如果黑客的xss注入是通过某种方式储存到了数据库中，那就是存储型的，这种xss的特点就是每次访问该页面都会收到xss攻击，因为js语句已经放在数据库里了。

而反射型xss则不是这样，每次触发只能手动输入和点击才能触发。

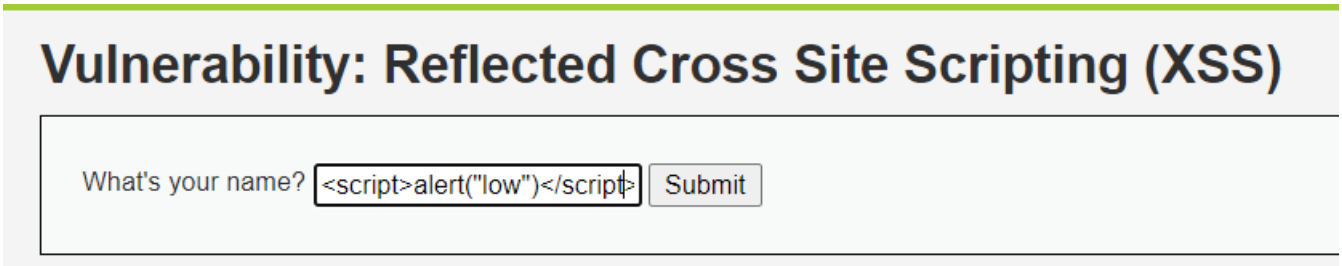
我认为xss产生的原因主要是对html标签审查不严格造成的。

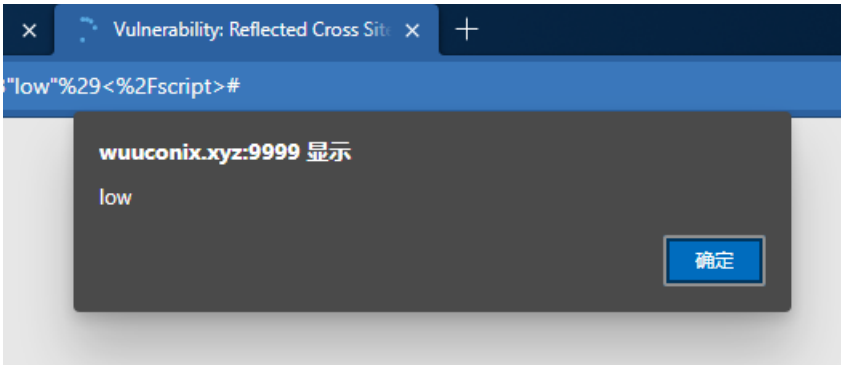
## dvwa xss例题

下面写一下dvwa中的三种难度的反射型xss。

```php
<?php
// Low难度
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
?>
```

这里没有对输入`$_GET['name']`做任何限制，我们完全可以在这个变量里写一个script标签。

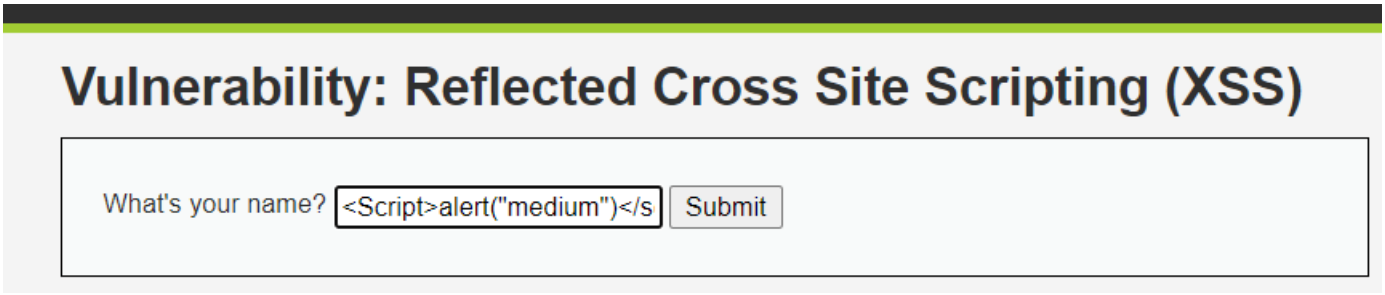


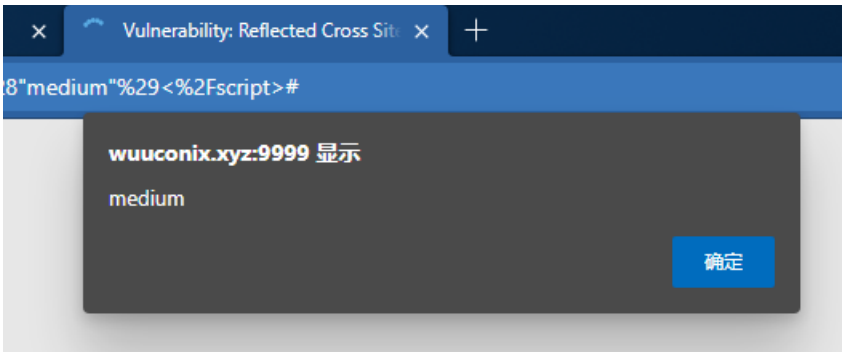



```php
<?php
// Medium 难度
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );
    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}
?>
```

这里把输入里的`<script>`替换为了空字符。但是这里是大小写敏感的，我们完全可以大写绕过。

```
<Script>alert("medium")</script>
```




或者双拼绕过。

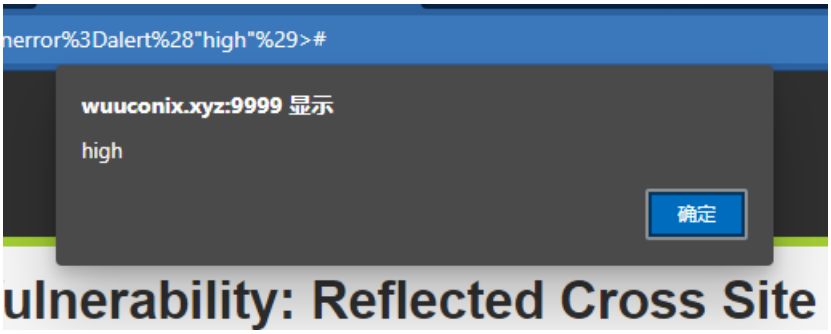
```
<scri<script>pt>alert("medium")</script>
```

```php
<?php
// High 难度
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );
    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}
?>
```

最高难度用了正则匹配，并且大小写不敏感。上面两种方法都失效了。

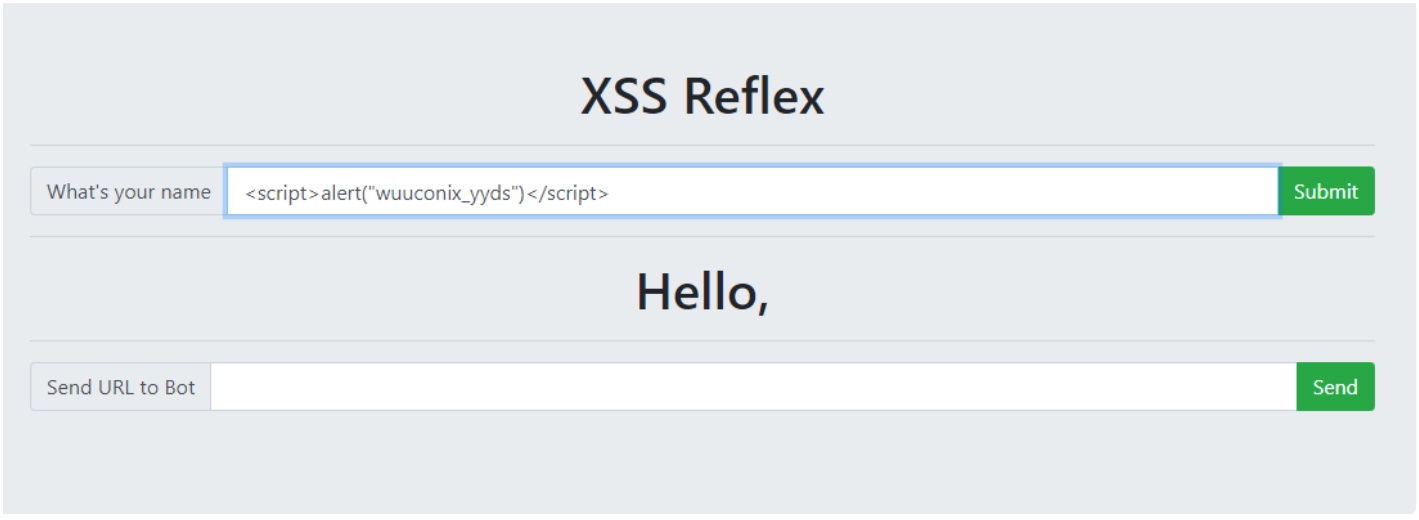
但是它只过滤了`script`标签这种xss，还可以利用img标签报错来实现弹窗。

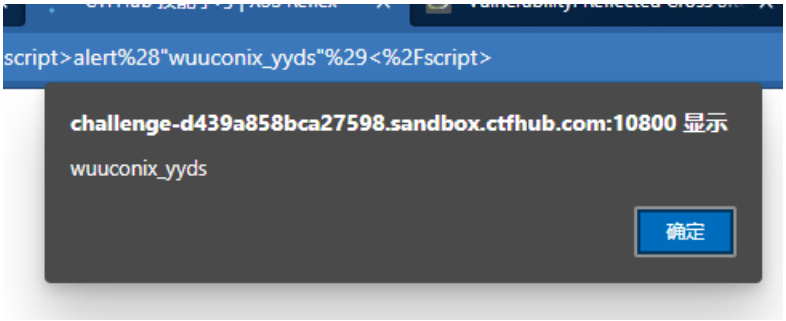
```
<img src=0 onerror=alert("high")>
```



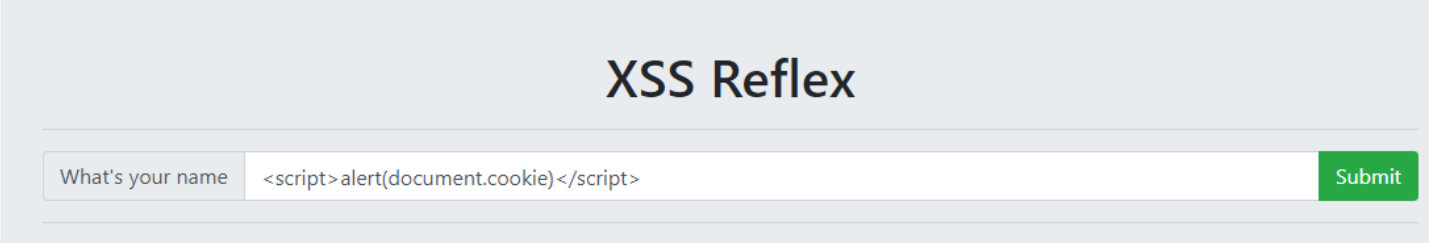

## CTFHUB的题目

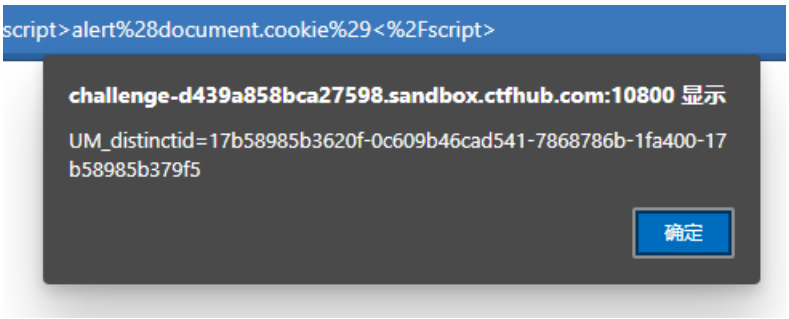
然后我便开始做ctfhub的题目了。我试了一下，发现它没有任何验证，可以直接xss。





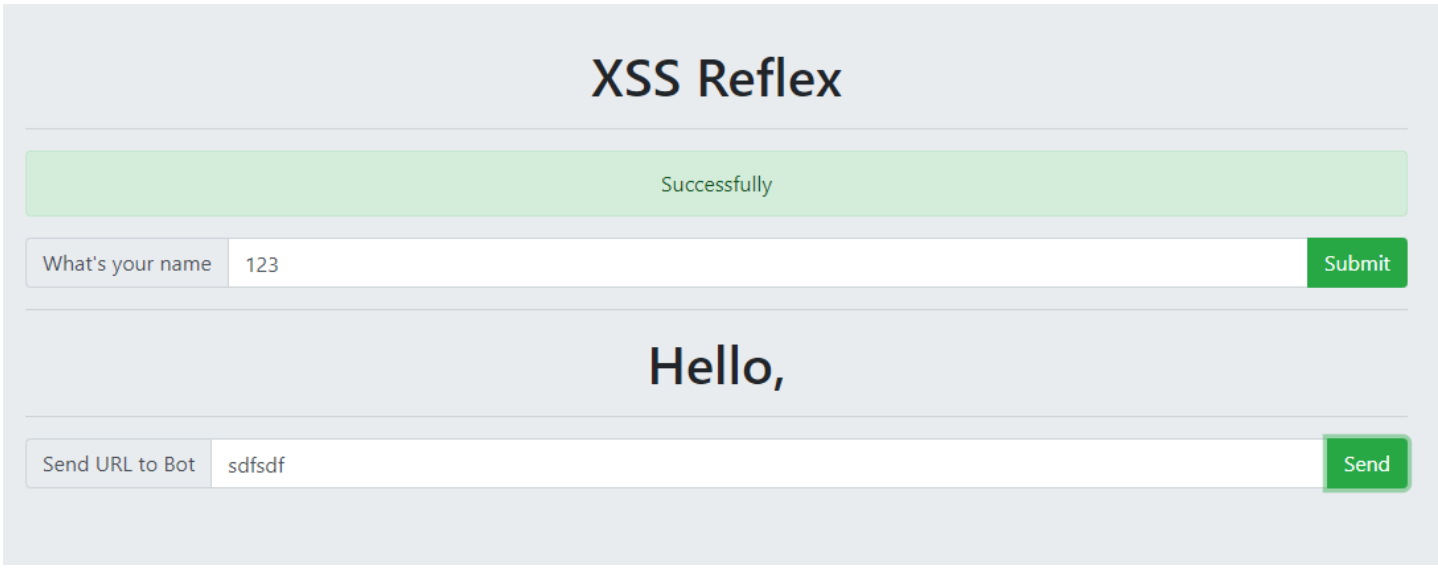

但是我不知道flag会藏在哪里，xss的作用只是操控js，会不会藏在cookie里呢？




很不幸，没有flag。我陷入了人生和社会的大思考。

最终没法，看了writeup。发现需要利用到第二个输入框。



第二个输入框点击send之后就会显示`successfully`，但是这个它发送到哪里无法确定，这个网页用到Bootstrap，我不太熟悉。这可以肯定的是它有一个后端。

然后可以利用xss platform来进行获得它与后端的信息。

在xss platform里新建一个项目然后复制其中的实例代码。


把payload在第一个输入框提交，然后复制url到第二个输入框提交后，就会在xss platform里得到相应。


下面进行战术总结

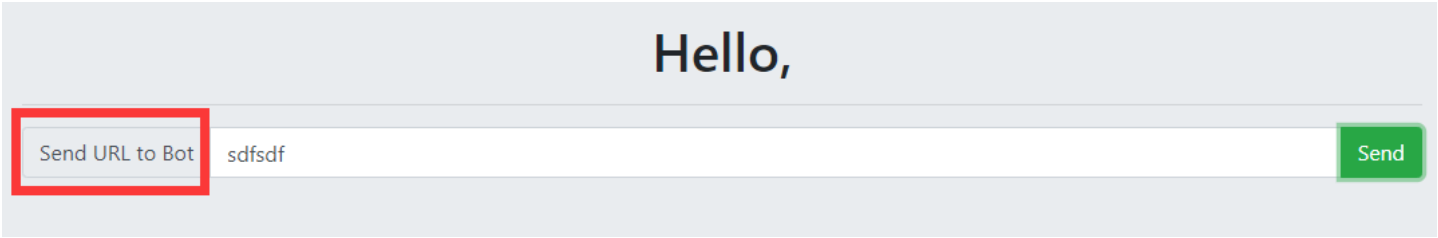
我们一开始直接用xss来看cookie，发现没有flag。我一开始觉得奇怪，觉得flag就应该藏到这个地方，不然还能藏哪呢？

我这里犯了一个原则性的错误。我们用xss一般的用途是什么？是获取cookie嘛？

是获取cookie，但更准确的说，是获取别人的cookie。

cookie相当于每个人的登录凭证，如果得到了别人的cookie，我们将可以不用输账号密码，直接登录。

所以flag一定是不可能藏在自己的cookie里的，自己的cookie没有意义，自己的cookie能直接浏览器控制台里知道，也不需要xss。ctf的题目应该是让我们获得别人的cookie，但是这是ctf的题目，不是公共的服务，没有其他用户，所以ctf模拟了一个机器人。




那就很清楚了，我们的目标就是获得这个机器人的Cookie，然后"盗它的号"，所以获取了这个机器人的Cookie就意味着成功。所以理所应当的，flag也就藏在cookie里了。

所以第二个文本框就是模拟别人点击这个包含xss的链接的情形。