Reverse 500

这题一开始没做出来，后经 zw 的指点才补上，也在此感谢 zw 和大姐头的帮助。

和上一道题一样都是 Python，第一想法是去网上找 pyc 反编译的工具，根据推荐找到了 uncompyle，直接反编译报错如下：

```
Syntax error at or near `NOP' token at offset 0
# decompiled 0 files: 0 okay, 1 failed, 0 verify failed
# 2015.09.27 17:06:53 CST
```

那么看来是一个被损坏或者被处理过的 pyc，需要人工分析。参考了 Python 2.6.2的.pyc文件格式 这篇文章，尝试人工反编译难度太大，注意到有一个 **mtime** 的文件头，在 import 的时候会比较这个 mtime 和相对应 Python 源文件的 mtime 是否一致，如果不一致的话会用源文件重新生成一份 pyc 覆盖。这样我就可以新建一个空的源文件，并修改其 mtime 使得可以直接导入该 pyc。遗憾的是这种方法并不正确，导入了函数以后无法顺利执行。

回过头再看 uncompyle 的报错信息

```
ParserError: --- This code section failed: ---
0   NOP                 ''
1   LOAD_CONST          "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x1
2%\x1d-"
4   LOAD_CONST          "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x1
2%\x1d-"
7   NOP                 ''
8   LOAD_CONST          "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x1
2%\x1d-"
11  LOAD_CONST          'I\x7fM(I{I\x7fJ.\x16wWcRj\x0e6\x0f
n'
14  BINARY_ADD          ''
15  LOAD_CONST          'Zo\nn\x0fk\t1R7\x03g\x067\x00eUb\x
043'
18  BINARY_ADD          ''
19  LOAD_CONST          '\x014\x071Rr\x14x\x19~D?q"a5s,A%'
22  BINARY_ADD          ''
23  LOAD_CONST          "\x10'\x11uLyA%\x1d|DrFv\x12t\x11#B
&"
26  BINARY_ADD          ''
27  LOAD_CONST          'GsKzK*O)\x1c%GuC>\x1e\x7f\x1b+\x19
*'
30  BINARY_ADD          ''
31  LOAD_CONST          '\x1e&\x14-\x1f/\x1axAqBq@yO-LtE}'
```

```
Syntax error at or near `NOP' token at offset 0
# decompiled 0 files: 0 okay, 1 failed, 0 verify failed
```

可以看到是 NOP 指令的解析上出了问题，根据 Python 字节码指令集 提供的列表找到 NOP 指令对应的 Bytecode 是 09，于是用二进制编辑软件打开，将对应位置的 09 随意修改为另一单字节码指令值，例如 01。根据报错信息以此定位 NOP 的位置，总共出现了 4 次，可用 `09 64` 以及 `09 74` 去定位。

修改完了以后再尝试反编译依旧不行，于是换一个工具 pycdc，就可以成功反编译得到如下代码：

```
# Source Generated with Decompyle++
# File: Reverse04_test.pyc (Python 2.7)

data = "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x12%\x1d-" + 'I\x
```

```
7fM(I{I\x7fJ.\x16wWcRj\xe6\xfn' + 'Zo\nn\xfk\t1R7\x3g\x67
\x0eUb\x43' + '\x14\x71Rr\x14x\x19~D?q"a5s,A%' + "\x10'\x
11uLyA%\x1d|DrFv\x12t\x11#B&" + 'GsKzK*O)\x1c%GuC>\x1e\x7
f\x1b+\x19*' + '\x1e&\x14-\x1f/\x1axAqBq@yO-LtE}' + '\x1b
,MuBp\x12'
import os
import sys
import struct
import cStringIO
import string
import dis
import marshal
import types
import random
count = 0

def reverse(string):
    return string[::-1]

data_list = list(reverse(data)[1:])

def decrpyt(c, key2):
    global count
    data_list[count] = c ^ key2
    count += 1


def GetFlag1():
    key = struct.unpack('B', data[len(data) - 8])[0]
    for c in data_list:
        if count == 0:
            decrpyt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 3])[0]
        decrpyt(struct.unpack('B', c)[0], key)

    for c in data_list[::-1]:
        print chr(c),



def GetFlag2():
    key = struct.unpack('B', data[len(data) - 11])[0]
```

```
    for c in data_list:
        if count == 0:
            decrpyt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 4 - cou
nt])[0]
        decrpyt(struct.unpack('B', c)[0], key)

    for c in data_list[::-1]:
        print chr(c),



def GetFlag3():
    key = struct.unpack('B', data[len(data) - 5])[0]
    for c in data_list:
        if count == 0:
            decrpyt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 2 - cou
nt])[0]
        decrpyt(struct.unpack('B', c)[0], key)

    for c in data_list[::-1]:
        print chr(c),



def GetFlag4():
    global count
    key = struct.unpack('B', data[len(data) - 1])[0]
    for c in data_list:
        if count == 0:
            decrpyt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 1 - cou
nt])[0]
        decrpyt(struct.unpack('B', c)[0], key)

    count = 0
    for c in data_list[::-1]:
        print chr(c),
```

```
def GetFlag5():
    key = struct.unpack('B', data[len(data) - 9])[0]
    for c in data_list:
        if count == 0:
            decrpyt(struct.unpack('B', c)[0], key)
            continue
            key = struct.unpack('B', data[len(data) - 3 -
 count])[0]
            decrpyt(struct.unpack('B', c)[0], key)
    for c in data_list[::-12]:
        print chr(c),


GetFlag1()
```

这里还有一个坑是 pycdc 在处理例如 `0f aa` 这样的指令时会变成 `\xfa\xa*` 这样，就会导致错位而解不出 flag。这个解决办法可以自己修改源码，也可以手动将 data 抠出来并去掉连接符。我抠出来的 data 是这样的

```
data = "4D2C1D2D187D45271E7A4E7E1B2A192B12251D2D497F4D284
97B497F4A2E16775763526A0E360F6E5A6F0A6E0F6B09315237036706
37006555620433013407315272147819 7E443F71226135732C4125102
711754C7941251D7C447246761274112342264773 4B7A4B2A4F291C25
4775433E1E7F1B2B192A1E26142D1F2F1A784171427140794F2D4C744
57D1B2C4D75427012".decode('hex')
```

用这个替换掉原来的 data 并依次尝试 5 个 GetFlag，最终在 GetFlag4 中得到 flag

```
flag:{NSCTF_md576d958d8a8640dfe2ada4811aef59b26}
```