# 2021-10-04 CTF-Reserve

Ch1lkat 于 2021-10-04 17:11:35 发布 1204 收藏

分类专栏： BUUCTF Reserve 文章标签： python windows 网络安全

本文链接：https://blog.csdn.net/m0_56897090/article/details/120605639

版权

BUUCTF 同时被 2 个专栏收录

8 篇文章 0 订阅

订阅专栏

Reserve

1 篇文章 0 订阅

订阅专栏

## 文章目录

# 基础框架

# 指令体系

# 逆向分析

# 算法识别

通常CTF中会出现base64、TEA、AES、RC4、MD5等算法

特征：

| 算法 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| base64 | 64+1位表、右移左移取二进制操作 | | | | | | | |
| TEA | 每轮加密涉及移位 <<4 >>5 与 delta常量(0x9e3779b9) | | | | | | | |
| AES | 主要特征为sbox(0x63)和逆sbox(0x52) | | | | | | | |
| RC4 | 涉及初始化函数和加密函数（交换s[i]与s[j]）、 %256 | | | | | | | |
| MD5 | 散列常量：0x67452301、0xefcdab89、0x98badcfe、0x10325476 | | | | | | | |

# 反调试

## **Windows**

### 自动化反调试

当遇到花指令或其他需要Patch汇编时

1. 使用IDAPatch
2. 使用IDCPython自动化批量Patch

# Linux

## 虚拟机保护

> 通常，VMProtect等商用软件采用了虚拟机保护技术，它们核心都会有一个vm_init阶段完成初始化自己的一套(ISA)指令集架构

## 脚本语言逆向

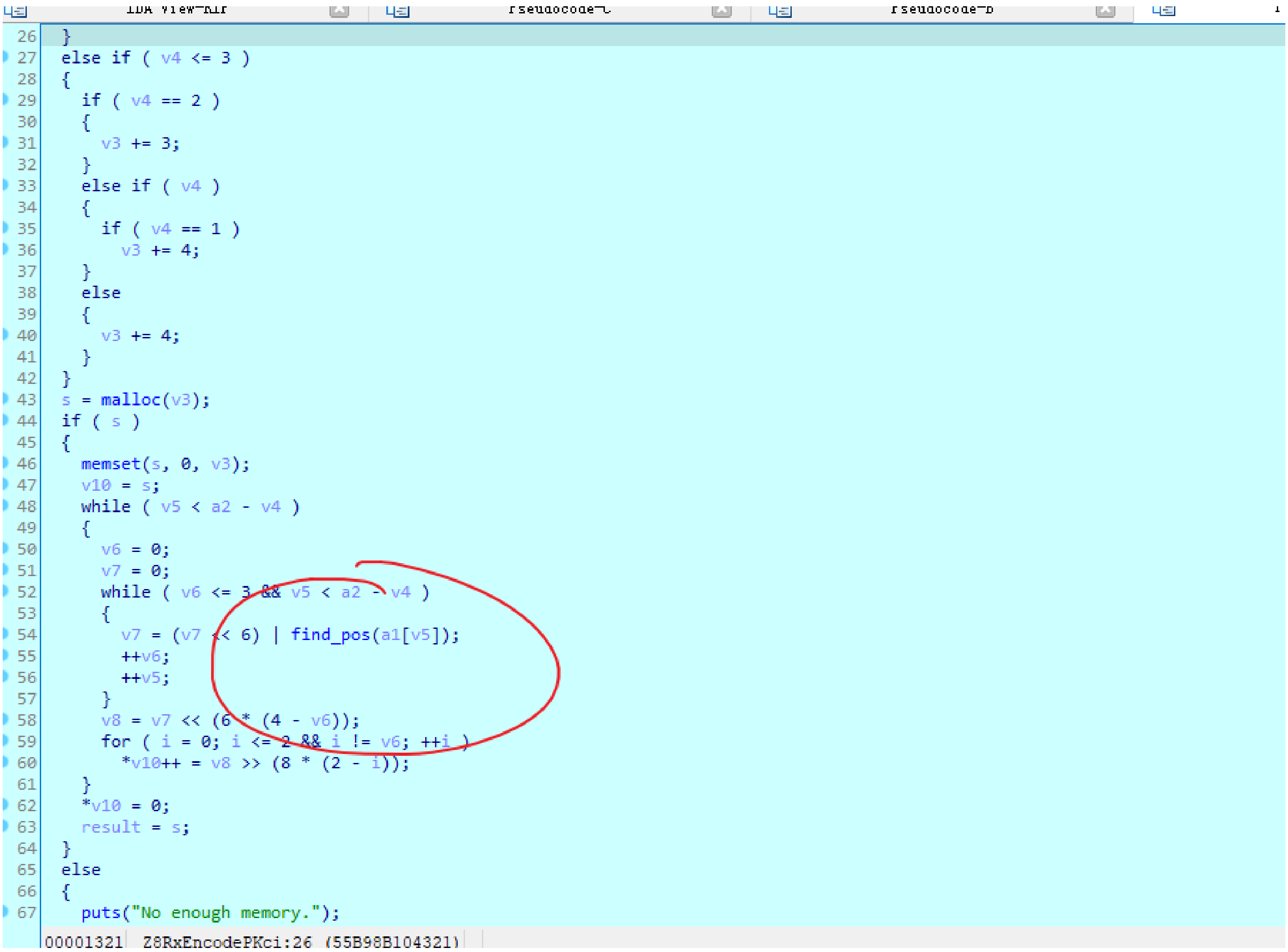
.NET/Python/Java

# RxEncode

## 分析

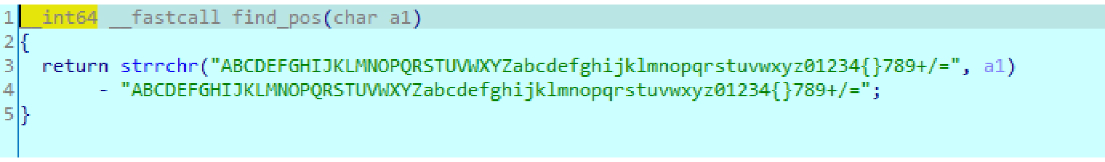
静态分析 很像是换表的BASE64

关键算法：



```
  }
  else if ( v4 <= 3 )
  {
    if ( v4 == 2 )
    {
      v3 += 3;
    }
    else if ( v4 )
    {
      if ( v4 == 1 )
        v3 += 4;
    }
    else
    {
      v3 += 4;
    }
  }
  s = malloc(v3);
  if ( s )
  {
    memset(s, 0, v3);
    v10 = s;
    while ( v5 < a2 - v4 )
    {
      v6 = 0;
      v7 = 0;
      while ( v6 <= 3 && v5 < a2 - v4 )
      {
        v7 = (v7 << 6) | find_pos(a1[v5]);
        ++v6;
        ++v5;
      }
      v8 = v7 << (6 * (4 - v6));
      for ( i = 0; i <= 2 && i != v6; ++i )
        *v10++ = v8 >> (8 * (2 - i));
    }
    *v10 = 0;
    result = s;
  }
  else
  {
    puts("No enough memory.");
```

```
00001321 Z8RxEncodePKci:26 (55B98B104321)
```



```
__int64 __fastcall find_pos(char a1)
{
  return strrchr("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234{}789+/=", a1)
       - "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234{}789+/=";
}
```

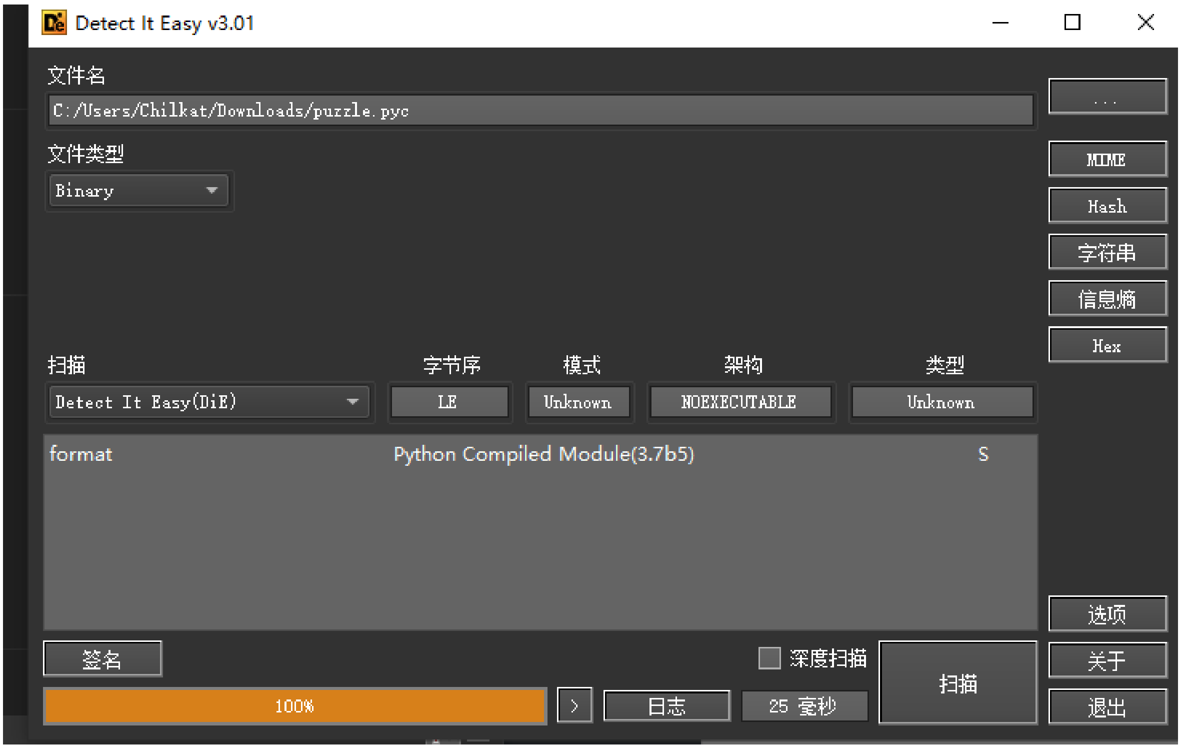
C 库函数 **char *strrchr(const char str, int c)* 在参数 **str** 所指向的字符串中搜索最后一次出现字符 **c**（一个无符号字符）的位置。

已经确定是换表的basee64解码，等待进一步完善RXEncode.py的EXP

## Real_easy_python

### 分析

查壳是 python3.7的pyc文件



文件名
C:/Users/Chilkat/Downloads/puzzle.pyc

文件类型
Binary

扫描 Detect It Easy(DiE)
字节序 LE
模式 Unknown
架构 NOEXECUTABLE
类型 Unknown

format Python Compiled Module(3.7b5) S

100%

25 毫秒

使用uncompyle6工具解码转换py:

```
root@ubuntu:~# uncompyle6 -o puzzle.py puzzle.pyc
```

解码代码:

```
# uncompyle6 version 3.7.4
# Python bytecode 3.7 (3394)
# Decompiled from: Python 3.5.2 (default, Jan 26 2021, 13:30:48)
# [GCC 5.4.0 20160609]
# Embedded file name: ./source.py
# Compiled at: 2020-08-03 05:55:47
# Size of source mod 2**32: 515 bytes
key = [
 115, 76, 50, 116, 90, 50, 116, 90, 115, 110, 48, 47, 87, 48, 103, 50, 106, 126, 90, 48, 103, 116, 126, 90, 85,
126, 115, 110, 105, 104, 35]
print('Input your flag: ', end='')
flag = input()
out = []
for i in flag:
    out.append(ord(i) >> 4 ^ ord(i))

if len(out) != len(key):
    print('TRY AGAIN!')
    exit()
for i in range(len(out)):
    if out[i] != key[i]:
        print('TRY AGAIN!')
        exit()

print('you are right! the flag is : moectf{%s}' % flag)
```

## EXP

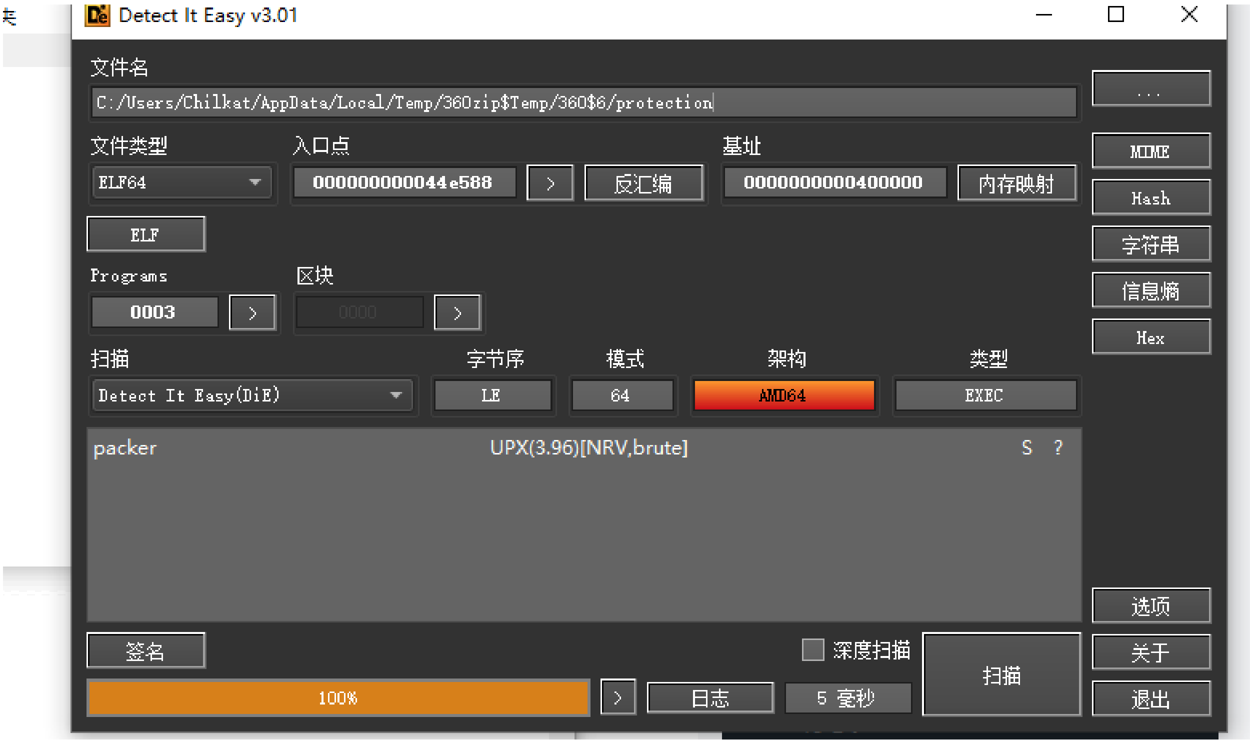
顺加倒解

```
key = [
 115, 76, 50, 116, 90, 50, 116, 90, 115, 110, 48, 47, 87, 48, 103, 50, 106, 126, 90, 48, 103, 116, 126, 90, 85,
126, 115, 110, 105, 104, 35]
for i in range(len(key)):
    out.append(chr(key[i] >>4  ^ key[i]))

print(''.join(out))#moectf{tH1s_1s_th3-R3a1ly_3asy_Python!}
```

# Protection

## 分析

查壳UPX3.96



Detect It Easy v3.01

文件名: C:/Users/Chilkat/AppData/Local/Temp/360zip$Temp/360$6/protection

文件类型: ELF64

入口点: 000000000044e588

基址: 0000000000400000

Programs: 0003

扫描: Detect It Easy(DiE)

字节序: LE

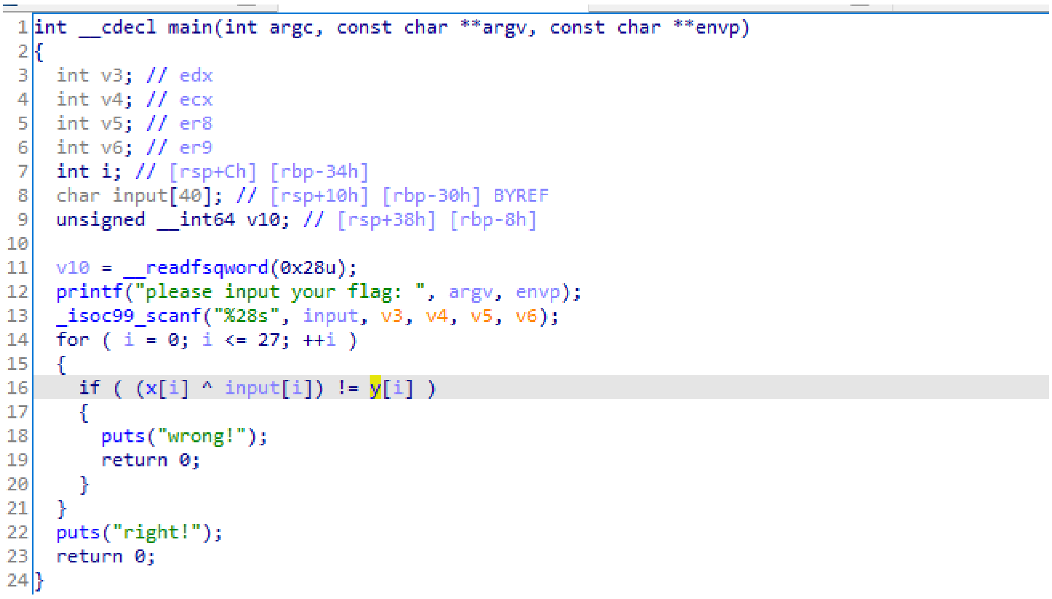
模式: 64

架构: AMD64

类型: EXEC

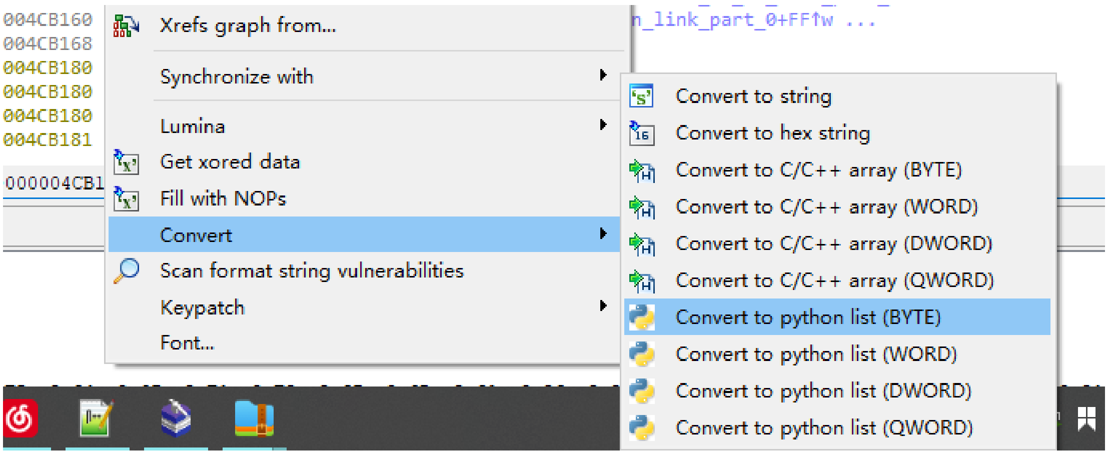
packer UPX(3.96)[NRV,brute]

脱壳后，发现是xor，照葫芦还原x^y=flag



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // edx
  int v4; // ecx
  int v5; // er8
  int v6; // er9
  int i; // [rsp+Ch] [rbp-34h]
  char input[40]; // [rsp+10h] [rbp-30h] BYREF
  unsigned __int64 v10; // [rsp+38h] [rbp-8h]

  v10 = __readfsqword(0x28u);
  printf("please input your flag: ", argv, envp);
  _isoc99_scanf("%28s", input, v3, v4, v5, v6);
  for ( i = 0; i <= 27; ++i )
  {
    if ( (x[i] ^ input[i]) != y[i] )
    {
      puts("wrong!");
      return 0;
    }
  }
  puts("right!");
  return 0;
}
```

快速转换数组（convert）


004CB160
004CB168
004CB180
004CB180
004CB180
004CB181
000004CB1
Xrefs graph from...
Synchronize with
Lumina
Get xored data
Fill with NOPs
Convert
Scan format string vulnerabilities
Keypatch
Font...
n_link_part_0+FF↑w ...
Convert to string
Convert to hex string
Convert to C/C++ array (BYTE)
Convert to C/C++ array (WORD)
Convert to C/C++ array (DWORD)
Convert to C/C++ array (QWORD)
Convert to python list (BYTE)
Convert to python list (WORD)
Convert to python list (DWORD)
Convert to python list (QWORD)

丝滑~moectf{upx_1S_simp1e-t0_u3e}

## EXP

```
x=[0x61, 0x6F, 0x75, 0x76, 0x23, 0x40, 0x21, 0x56, 0x30, 0x38, 0x61, 0x73, 0x64, 0x6F, 0x7A, 0x70, 0x6E, 0x6D, 0
x61, 0x26, 0x2A, 0x23, 0x25, 0x21, 0x24, 0x5E, 0x26, 0x2A, 0x00]
y=[0x0C, 0x00, 0x10, 0x15, 0x57, 0x26, 0x5A, 0x23, 0x40, 0x40, 0x3E, 0x42, 0x37, 0x30, 0x09, 0x19, 0x03, 0x1D, 0
x50, 0x43, 0x07, 0x57, 0x15, 0x7E, 0x51, 0x6D, 0x43, 0x57, 0x00, 0x00, 0x00, 0x00]

ans=''
for i in range(len(x)):
    ans+=chr(x[i]^y[i])

print(ans)
```

## SimpleRe

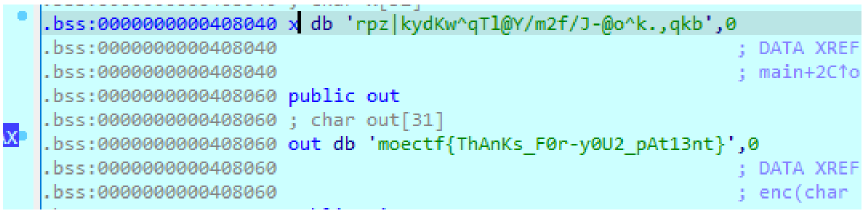
### 分析

MOE2020

简单的XOR，在动态中解密

顺加逆解，xor一遍是加密，xor第二遍就是解密
满足公式：

```
a^b=c
c^a=b
a^b^c=0
```



```
.bss:0000000000408040 x db 'rpz|kydKw^qTl@Y/m2f/J-@o^k.,qkb',0
.bss:0000000000408040                                        ; DATA XREF
.bss:0000000000408040                                        ; main+2C↑o
.bss:0000000000408060 public out
.bss:0000000000408060 ; char out[31]
.bss:0000000000408060 out db 'moectf{ThAnKs_F0r-y0U2_pAt13nt}',0
.bss:0000000000408060                                        ; DATA XREF
.bss:0000000000408060                                        ; enc(char
```

# Thank you JavaScript

## 分析

去在线解混淆JS代码
http://edit.89tool.com/

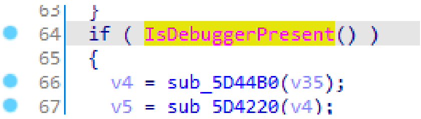
```
const io = require('console-read-write');
async
function main() {
    io.write('MoeCTF 2020 ThankYouJavaScript --written by Reverier');
    io.write(await io.read());
    io.write(`Hello $ {
        await io.ask('Who are you?')
    } ! `);
    let saidHi = false;
    while (!saidHi) {
        io.write('Please input the true flag:');
        saidHi = await io.read() === 'moectf{Fx' + 'c' + 'k_' + 'Y' + '0' + 'u-' + 'Jav' + 'aS' + 'cr' + 'ipt' +
 '!}'
    }
    io.write('Congratulations! You find the flag!')
}
main();
```

得到flag:moectf{Fxck_Y0u-JavaScript!}

# EasyCPP

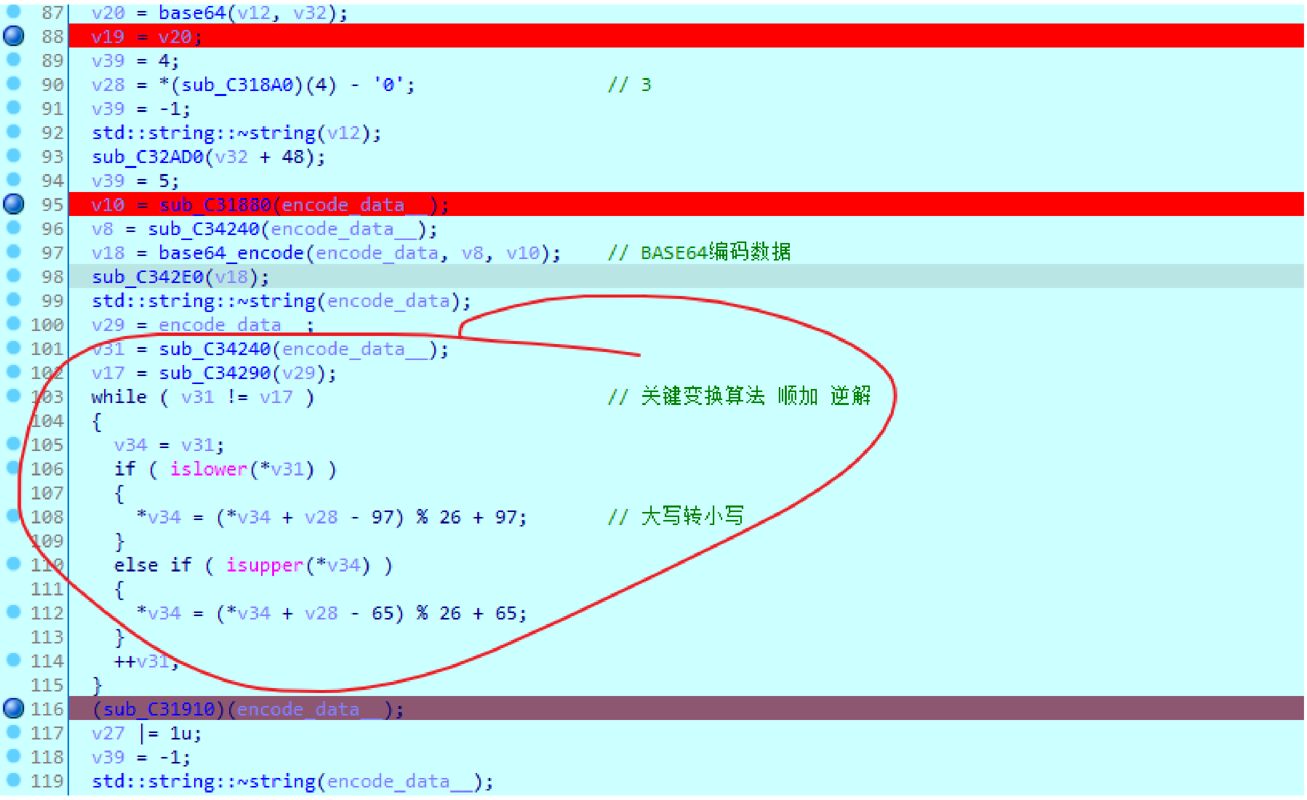
## 分析

nop



```
  }
  if ( IsDebuggerPresent() )
  {
    v4 = sub_5D44B0(v35);
    v5 = sub 5D4220(v4);
```

继续分析发现是一个简单的变换算法，其余的都是混淆：



```
v20 = base64(v12, v32);
v19 = v20;
v39 = 4;
v28 = *(sub_C318A0)(4) - '0';                   // 3
v39 = -1;
std::string::~string(v12);
sub_C32AD0(v32 + 48);
v39 = 5;
v10 = sub_C31880(encode_data__);
v8 = sub_C34240(encode_data__);
v18 = base64_encode(encode_data, v8, v10);     // BASE64编码数据
sub_C342E0(v18);
std::string::~string(encode_data);
v29 = encode_data__;
v31 = sub_C34240(encode_data__);
v17 = sub_C34290(v29);
while ( v31 != v17 )                            // 关键变换算法 顺加 逆解
{
  v34 = v31;
  if ( islower(*v31) )
  {
    *v34 = (*v34 + v28 - 97) % 26 + 97;         // 大写转小写
  }
  else if ( isupper(*v34) )
  {
    *v34 = (*v34 + v28 - 65) % 26 + 65;
  }
  ++v31;
}
(sub_C31910)(encode_data__);
v27 |= 1u;
v39 = -1;
std::string::~string(encode_data__);
```

笔记

1. unknown lib name
2. 对于调试保护可以用Patch修改

## EXP

```python
import binascii


def encode(old):
#old="YWFhYQ=="
    ans=""
    for i in range(len(old)):
        i=old[i]
        num=ord(i)
        if(i.islower()):
            ans+=chr((num+3-97)%26+97)
        elif i.isupper():
            ans+=chr((num+3-65)%26+65)
        else:ans+=chr(num)


def decode(old):
    #old="BZIkBT=="#YWFhYQ=="
    ans=""
    for i in range(len(old)):
        i=old[i]
        num=ord(i)
        if(i.islower()):
            anx=(num-97)%26 +97-3
            if(anx<97):
                anx=97+26-(97-anx)

            ans+=chr(anx)
            print((num-97)%26 +97-3)

        elif i.isupper():
            anx=(num-65)%26 +65-3
            if(anx<65):
                anx=65+26-(65-anx)

            ans+=chr(anx)
            print((num-65)%26 +65-3)
        else:ans+=chr(num)
    return ans

print(decode("eZ9oB3Uph0QTXI9FBYQIQmUiT2IoX0EbAcIWA3PzA2YkBZIkf3o9"))
print(chr((89+3-65)%26+65))
```

# AlgorithmTask-HardTask

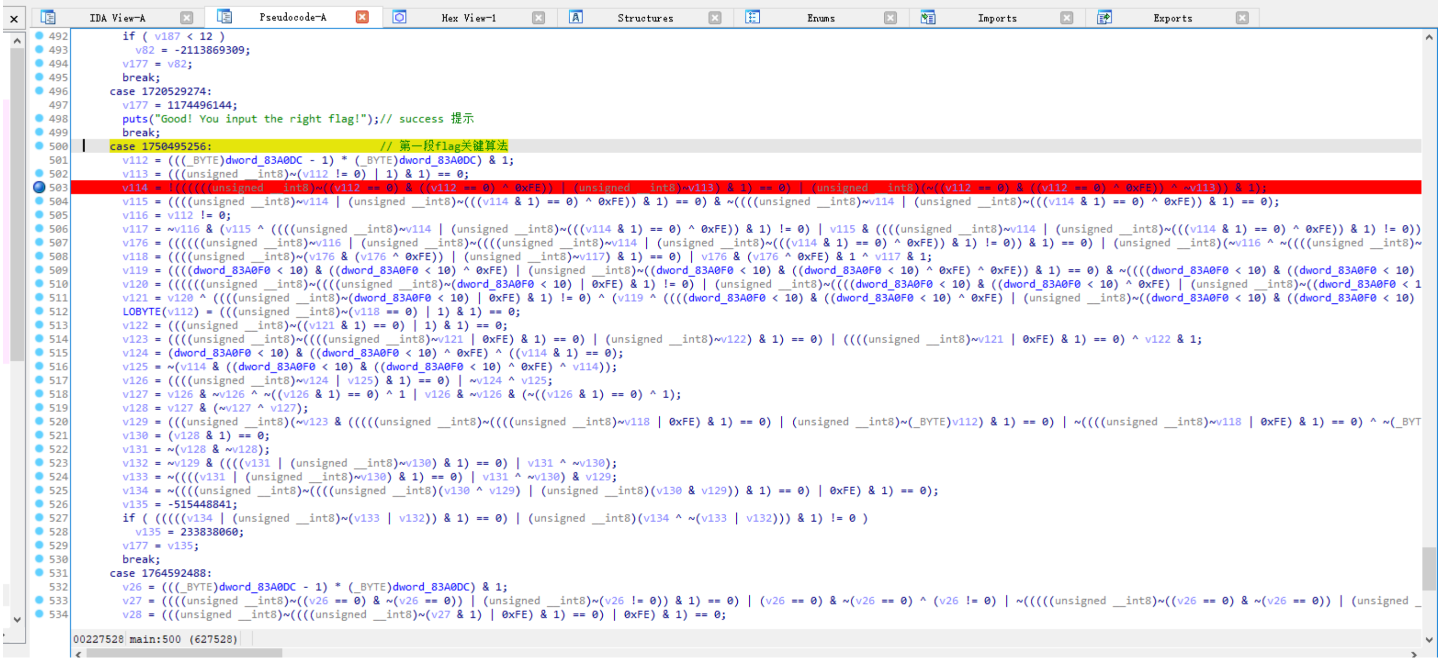
## 分析

MOE2019

strncmp

> 若str1与str2的前n个字符相同，则返回0；

程序流程被混淆，稍微分析了下、将字符串用a转换了一下，重新反编译恢复字符串：



```
      if ( v187 < 12 )
        v82 = -2113869309;
      v177 = v82;
      break;
    case 1720529274:
      v177 = 1174496144;
      puts("Good! You input the right flag!");// success 提示
      break;
    case 1750495256:                          // 第一段flag关键算法
      v112 = (((_BYTE)dword_83A0DC - 1) * (_BYTE)dword_83A0DC) & 1;
      v113 = (((unsigned __int8)~(v112 != 0) | 1) & 1) == 0;
      v114 = !((((((unsigned __int8)~((v112 == 0) & ((v112 == 0) ^ 0xFE)) | (unsigned __int8)~v113) & 1) == 0) | (unsigned __int8)(~((v112 == 0) & ((v112 == 0) ^ 0xFE)) ^ ~v113)) & 1);
      v115 = ((((unsigned __int8)~v114 | (unsigned __int8)~(((v114 & 1) == 0) ^ 0xFE)) & 1) == 0) & ~((((unsigned __int8)~v114 | (unsigned __int8)~(((v114 & 1) == 0) ^ 0xFE)) & 1) == 0);
      v116 = v112 != 0;
      v117 = ~v116 & (v115 ^ ((((unsigned __int8)~v114 | (unsigned __int8)~(((v114 & 1) == 0) ^ 0xFE)) & 1) != 0) | v115 & ((((unsigned __int8)~v114 | (unsigned __int8)~(((v114 & 1) == 0) ^ 0xFE)) & 1) != 0))
      v176 = ((((((unsigned __int8)~v116 | (unsigned __int8)~((((unsigned __int8)~v114 | (unsigned __int8)~(((v114 & 1) == 0) ^ 0xFE)) & 1) != 0)) & 1) == 0) | (unsigned __int8)(~v116 ^ ~((((unsigned __int8)~
      v118 = ((((unsigned __int8)~(v176 & (v176 ^ 0xFE)) | (unsigned __int8)~v117) & 1) == 0) | v176 & (v176 ^ 0xFE) & 1 ^ v117 & 1;
      v119 = ((((dword_83A0F0 < 10) & ((dword_83A0F0 < 10) ^ 0xFE) | (unsigned __int8)~((dword_83A0F0 < 10) & ((dword_83A0F0 < 10) ^ 0xFE) ^ 0xFE)) & 1) == 0) & ~((((dword_83A0F0 < 10) & ((dword_83A0F0 < 10)
      v120 = ((((((unsigned __int8)~((((unsigned __int8)~(dword_83A0F0 < 10) | 0xFE) & 1) != 0) | (unsigned __int8)~((((dword_83A0F0 < 10) & ((dword_83A0F0 < 10) ^ 0xFE) | (unsigned __int8)~((dword_83A0F0 < 1
      v121 = v120 ^ ((((unsigned __int8)~(dword_83A0F0 < 10) | 0xFE) & 1) != 0) ^ (v119 ^ ((((dword_83A0F0 < 10) & ((dword_83A0F0 < 10) ^ 0xFE) | (unsigned __int8)~((dword_83A0F0 < 10) & ((dword_83A0F0 < 10)
      LOBYTE(v112) = (((unsigned __int8)~(v118 == 0) | 1) & 1) == 0;
      v122 = (((unsigned __int8)~((v121 & 1) == 0) | 1) & 1) == 0;
      v123 = ((((unsigned __int8)~((((unsigned __int8)~v121 | 0xFE) & 1) == 0) | (unsigned __int8)~v122) & 1) == 0) | ((((unsigned __int8)~v121 | 0xFE) & 1) == 0) ^ v122 & 1;
      v124 = (dword_83A0F0 < 10) & ((dword_83A0F0 < 10) ^ 0xFE) ^ ((v114 & 1) == 0);
      v125 = ~(v114 & ((dword_83A0F0 < 10) & ((dword_83A0F0 < 10) ^ 0xFE) ^ v114));
      v126 = ((((unsigned __int8)~v124 | v125) & 1) == 0) | ~v124 ^ v125;
      v127 = v126 & ~v126 ^ ~((v126 & 1) == 0) ^ 1 | v126 & ~v126 & (~((v126 & 1) == 0) ^ 1);
      v128 = v127 & (~v127 ^ v127);
      v129 = (((unsigned __int8)(~v123 & (((((unsigned __int8)~((((unsigned __int8)~v118 | 0xFE) & 1) == 0) | (unsigned __int8)~(_BYTE)v112) & 1) == 0) | ~((((unsigned __int8)~v118 | 0xFE) & 1) == 0) ^ ~(_BYT
      v130 = (v128 & 1) == 0;
      v131 = ~(v128 & ~v128);
      v132 = ~v129 & ((((v131 | (unsigned __int8)~v130) & 1) == 0) | v131 ^ ~v130);
      v133 = ~((((v131 | (unsigned __int8)~v130) & 1) == 0) | v131 ^ ~v130) & v129;
      v134 = ~((((unsigned __int8)~((((unsigned __int8)(v130 ^ v129) | (unsigned __int8)(v130 & v129)) & 1) == 0) | 0xFE) & 1) == 0);
      v135 = -515448841;
      if ( (((((v134 | (unsigned __int8)~(v133 | v132)) & 1) == 0) | (unsigned __int8)(v134 ^ ~(v133 | v132))) & 1) != 0 )
        v135 = 233838060;
      v177 = v135;
      break;
    case 1764592488:
      v26 = (((_BYTE)dword_83A0DC - 1) * (_BYTE)dword_83A0DC) & 1;
      v27 = ((((unsigned __int8)~((v26 == 0) & ~(v26 == 0)) | (unsigned __int8)~(v26 != 0)) & 1) == 0) | (v26 == 0) & ~(v26 == 0) ^ (v26 != 0) | ~(((((unsigned __int8)~((v26 == 0) & ~(v26 == 0)) | (unsigned _
      v28 = (((unsigned __int8)~((((unsigned __int8)~(v27 & 1) | 0xFE) & 1) == 0) | 0xFE) & 1) == 0;
```

00227528 main:500 (627528)

> case 1750495256: // 第一段flag关键算法

一共分2段flag

1. dcba3261b6ef0d77(16位)
2. X1l0X0YxYWdfWTB1XzRyZV9TdHlwbmd9

第二段base64可以解出来：_R4_F1ag_Y0u_4re_Str0ng}
看了WP，`盲猜` 是MD5算法，在cmd5查询是`enj0y`

要大胆猜RE，就那几种算法，不用想太复杂~

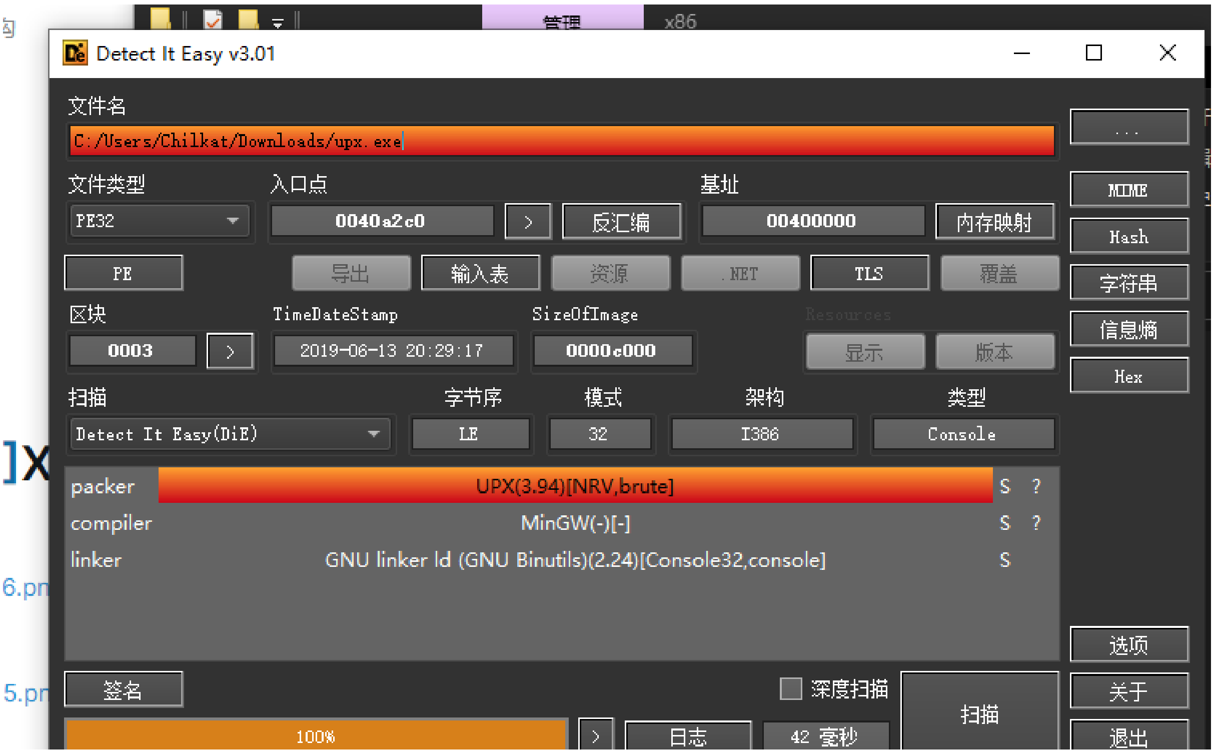
flag:moectf{enj0y_R4_F1ag_Y0u_4re_Str0ng}

## EasyShell

### 分析

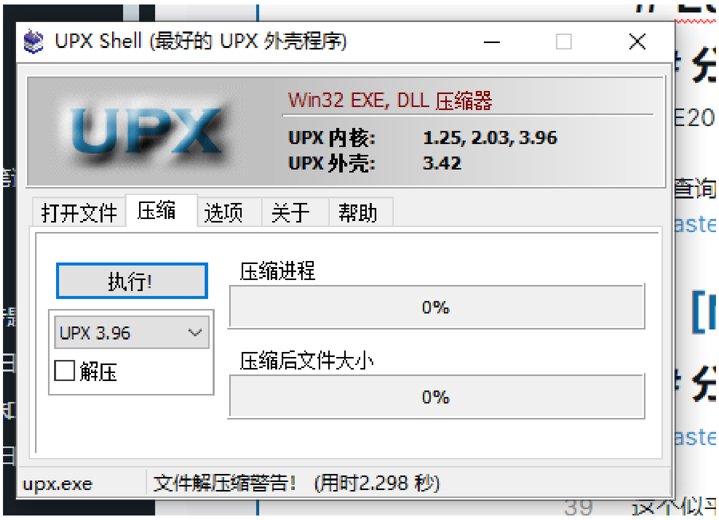
MOE2019 RE

DIE查询是UPX3.94



```
文件名: C:/Users/Chilkat/Downloads/upx.exe
文件类型: PE32    入口点: 0040a2c0    基址: 00400000
区块: 0003    TimeDateStamp: 2019-06-13 20:29:17    SizeOfImage: 0000c000
扫描: Detect It Easy(DiE)    字节序: LE    模式: 32    架构: I386    类型: Console
packer      UPX(3.94)[NRV,brute]                                   S  ?
compiler    MinGW(-)[-]                                            S  ?
linker      GNU linker ld (GNU Binutils)(2.24)[Console32,console]  S
100%    42 毫秒
```

使用UPX Shell脱壳



Win32 EXE, DLL 压缩器

UPX 内核: 1.25, 2.03, 3.96

UPX 外壳: 3.42

UPX 3.96

□ 解压

0%

0%

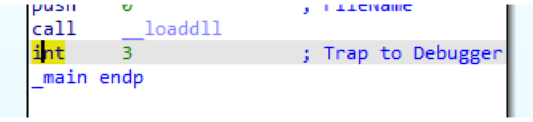
upx.exe | 文件解压缩警告! (用时2.298 秒)

## [MRCTF2020]Xor

### 分析

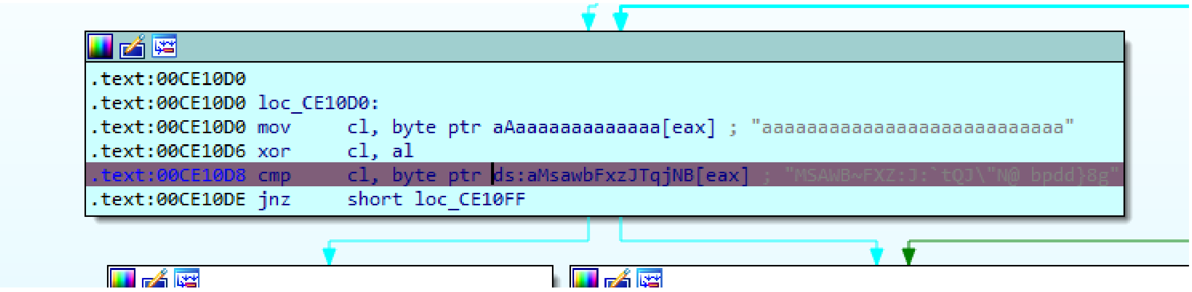
```
call    __loaddll
int     3                       ; Trap to Debugger
_main endp
```

这个似乎会阻止伪代码转换



```
push    0               ; FileName
call    __loaddll
int     3               ; Trap to Debugger
_main endp
```

很明显，在动态调试算法在这：



```
.text:00CE10D0
.text:00CE10D0 loc_CE10D0:
.text:00CE10D0 mov     cl, byte ptr aAaaaaaaaaaaaa[eax] ; "aaaaaaaaaaaaaaaaaaaaaaaaaaa"
.text:00CE10D6 xor     cl, al
.text:00CE10D8 cmp     cl, byte ptr ds:aMsawbFxzJTqjNB[eax] ; "MSAWB~FXZ:J:`tQJ\"N@ bpdd}8g"
.text:00CE10DE jnz     short loc_CE10FF
```

在动态调试中，发现算法是还原xor真实密码，并且逐个比较
还原公式=(变形flag ^ 字符索引)

笔记：

1. 未转换的变量为字符串，再按A重新转换（第一个字符）

## EXP

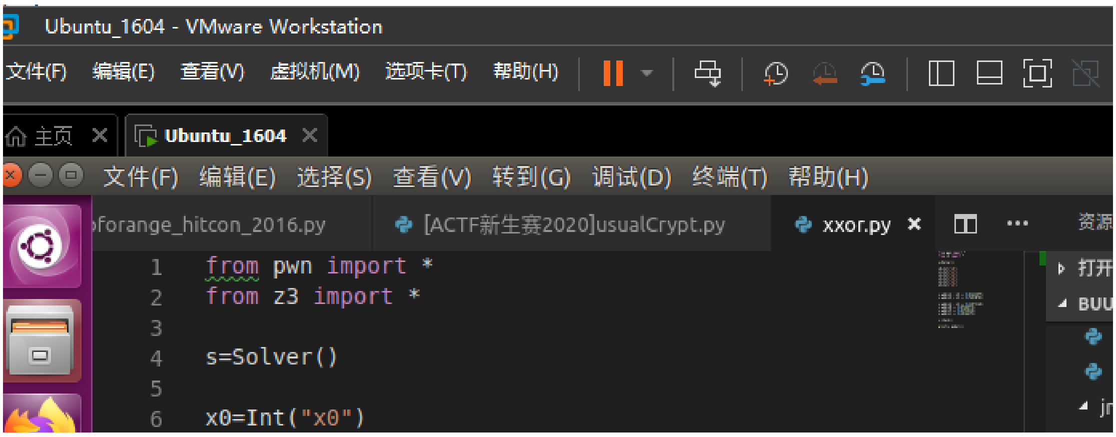
```
oldstr=[0x4D, 0x53, 0x41, 0x57, 0x42, 0x7E, 0x46, 0x58, 0x5A, 0x3A, 0x4A, 0x3A, 0x60, 0x74, 0x51, 0x4A, 0x22, 0x4E, 0x40, 0x20, 0x62, 0x70, 0x64, 0x64, 0x7D, 0x38,0x67]

ans=''
for i in range(len(oldstr)):
    ans+=chr(oldstr[i]^i)

print(ans)
```

# [GWCTF 2019]xxor

## 分析

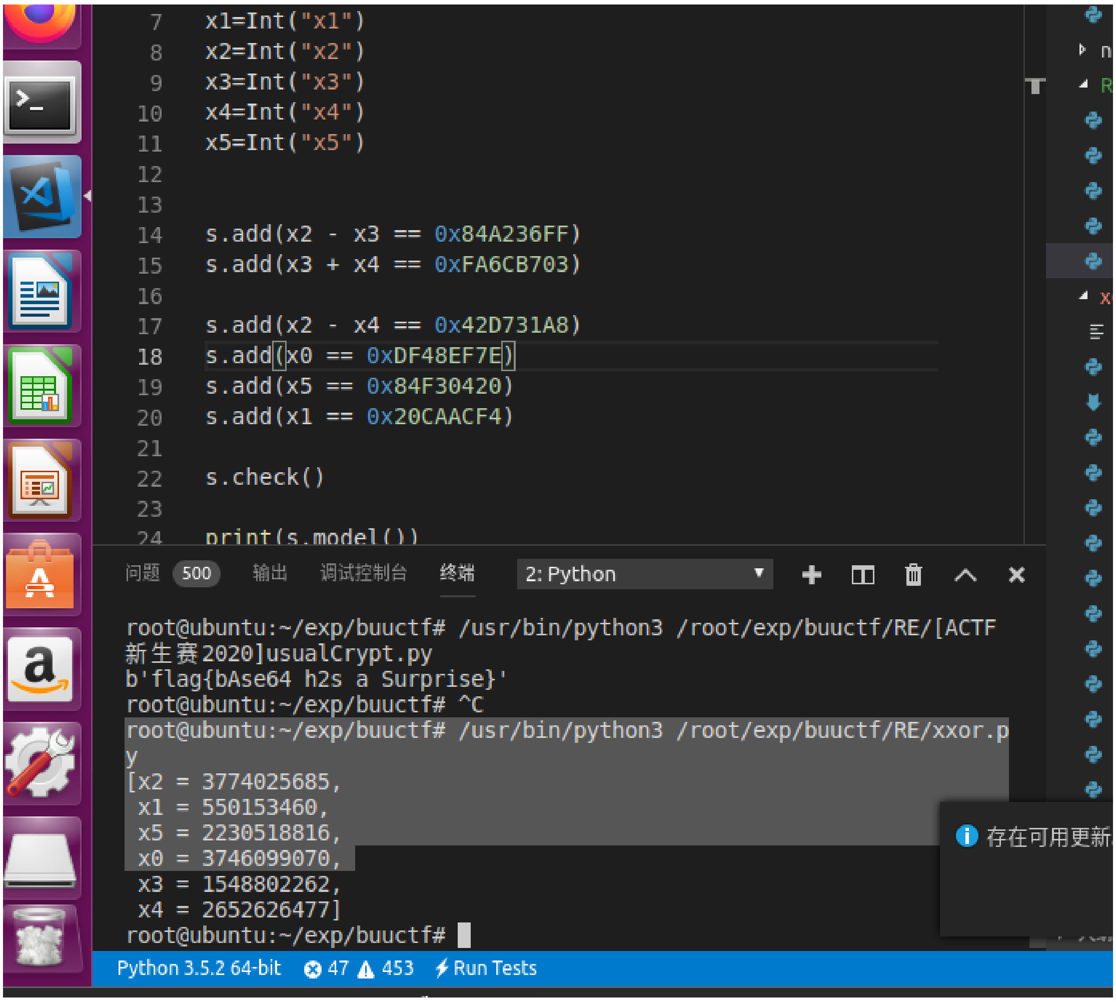
是一道算法题 （类似于TEA)

先解方程，通过z3求解：



Ubuntu_1604 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Ubuntu_1604
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 调试(D) 终端(T) 帮助(H)
forange_hitcon_2016.py
[ACTF新生赛2020]usualCrypt.py
xxor.py
资源
打开
BUU

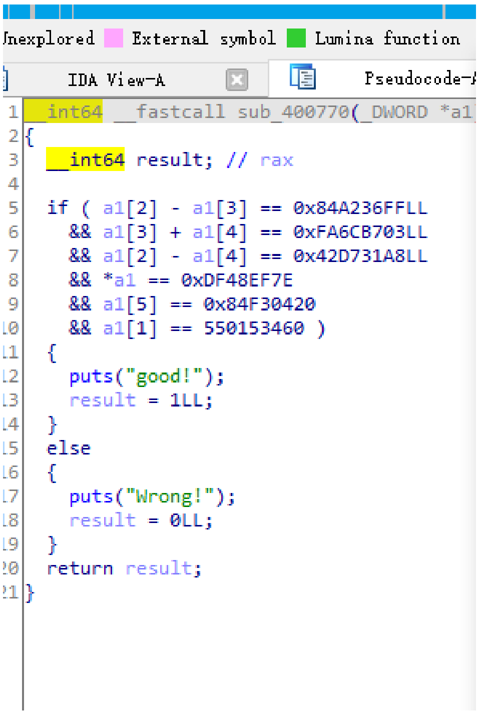
```
from pwn import *
from z3 import *

s=Solver()

x0=Int("x0")
```



```
x1=Int("x1")
x2=Int("x2")
x3=Int("x3")
x4=Int("x4")
x5=Int("x5")


s.add(x2 - x3 == 0x84A236FF)
s.add(x3 + x4 == 0xFA6CB703)

s.add(x2 - x4 == 0x42D731A8)
s.add(x0 == 0xDF48EF7E)
s.add(x5 == 0x84F30420)
s.add(x1 == 0x20CAACF4)

s.check()

print(s.model())
```

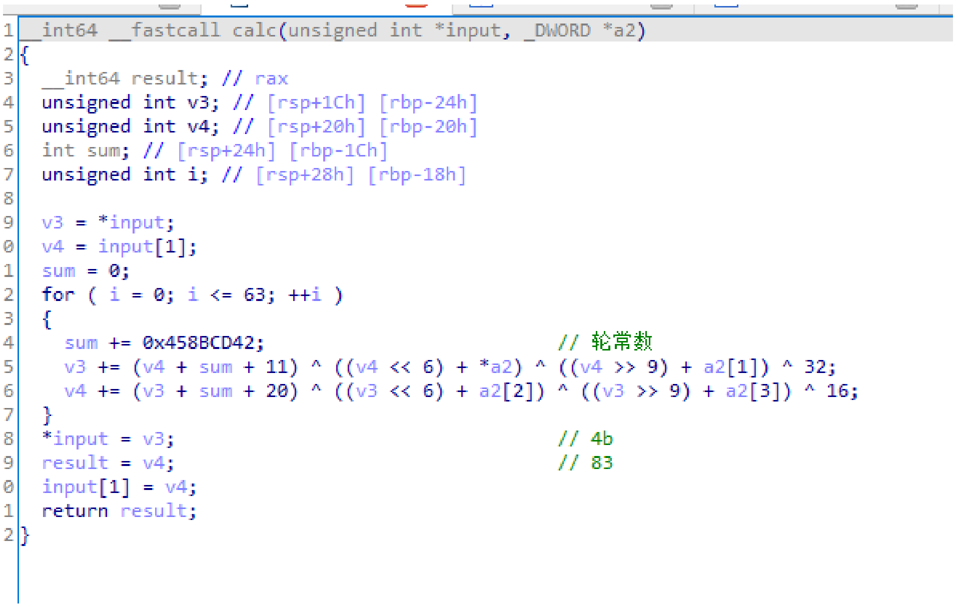
```
root@ubuntu:~/exp/buuctf# /usr/bin/python3 /root/exp/buuctf/RE/[ACTF
新生赛2020]usualCrypt.py
b'flag{bAse64 h2s a Surprise}'
root@ubuntu:~/exp/buuctf# ^C
root@ubuntu:~/exp/buuctf# /usr/bin/python3 /root/exp/buuctf/RE/xxor.p
y
[x2 = 3774025685,
 x1 = 550153460,
 x5 = 2230518816,
 x0 = 3746099070,
 x3 = 1548802262,
 x4 = 2652626477]
root@ubuntu:~/exp/buuctf#
```

主要验证代码：



```
__int64 __fastcall sub_400770(_DWORD *a1
{
  __int64 result; // rax

  if ( a1[2] - a1[3] == 0x84A236FFLL
    && a1[3] + a1[4] == 0xFA6CB703LL
    && a1[2] - a1[4] == 0x42D731A8LL
    && *a1 == 0xDF48EF7E
    && a1[5] == 0x84F30420
    && a1[1] == 550153460 )
  {
    puts("good!");
    result = 1LL;
  }
  else
  {
    puts("Wrong!");
    result = 0LL;
  }
  return result;
}
```

主要算法代码：（类似TEA轮加密）



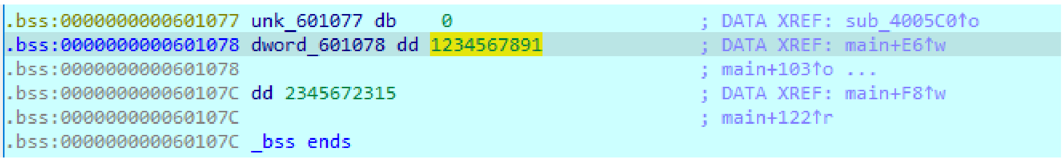
```
__int64 __fastcall calc(unsigned int *input, _DWORD *a2)
{
  __int64 result; // rax
  unsigned int v3; // [rsp+1Ch] [rbp-24h]
  unsigned int v4; // [rsp+20h] [rbp-20h]
  int sum; // [rsp+24h] [rbp-1Ch]
  unsigned int i; // [rsp+28h] [rbp-18h]

  v3 = *input;
  v4 = input[1];
  sum = 0;
  for ( i = 0; i <= 63; ++i )
  {
    sum += 0x458BCD42;                          // 轮常数
    v3 += (v4 + sum + 11) ^ ((v4 << 6) + *a2) ^ ((v4 >> 9) + a2[1]) ^ 32;
    v4 += (v3 + sum + 20) ^ ((v3 << 6) + a2[2]) ^ ((v3 >> 9) + a2[3]) ^ 16;
  }
  *input = v3;                                  // 4b
  result = v4;                                  // 83
  input[1] = v4;
  return result;
}
```

在数组中，伪代码

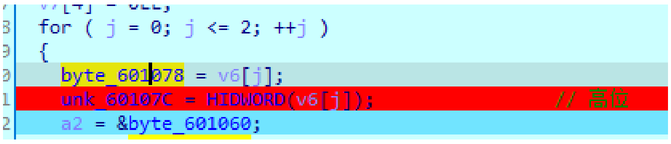
```
*input = input[0]
```

看了WP提示

```
a2[1]
a2[2]取值，要看类型
_DWORD 占4位

*a2=a2+0*4=a2
a2[1]= a2 +1*4
```



```
.bss:0000000000601077 unk_601077 db    0                      ; DATA XREF: sub_4005C0↑o
.bss:0000000000601078 dword_601078 dd 1234567891              ; DATA XREF: main+E6↑w
.bss:0000000000601078                                         ; main+103↑o ...
.bss:000000000060107C dd 2345672315                           ; DATA XREF: main+F8↑w
.bss:000000000060107C                                         ; main+122↑r
.bss:000000000060107C _bss ends
```



```
  v7[4] = 0LL;
  for ( j = 0; j <= 2; ++j )
  {
    byte_601078 = v6[j];
    unk_60107C = HIDWORD(v6[j]);                // 高位
    a2 = &byte_601060;
```

在动态调试中观察，可以很明显发现

其实是取

```
    byte_601078 = v6[j];
    unk_60107C = v6[j+1];
```

注意的点：

1. unsigned 符号类型 变量类型很重要
2. 加密顺，解密倒着完事

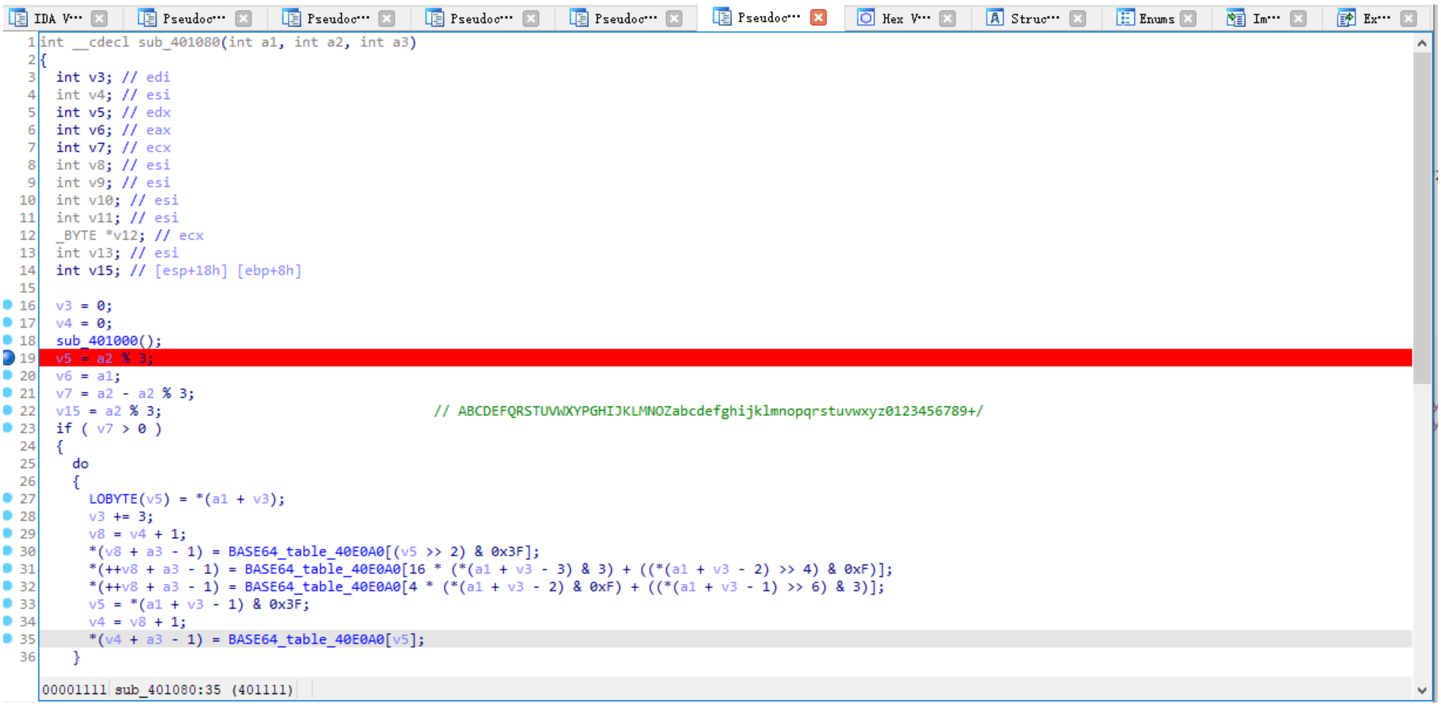
## EXP

```
void __fastcall decalc()
{
  __int64 xorm[6];
 xorm[0] = 3746099070;
 xorm[1] = 550153460;
 xorm[2] = 3774025685;
 xorm[3] = 1548802262;
 xorm[4] = 2652626477;
 xorm[5] = 2230518816;
 unsigned int  i = 0, j = 0, sum;
 unsigned int  temp[2] = { 0 };
 unsigned int  data[4] = { 2,2,3,4 };//unk哪个数字
 for (i = 0; i < 5; i += 2)
 {
  temp[0] = xorm[i];
  temp[1] = xorm[i + 1];

  sum = 0x458BCD42 * 64;//类似于tea 逆向
  for (j = 0; j < 64; j++)
  {
   temp[1] -= (temp[0] + sum + 20) ^ ((temp[0] << 6) + 3) ^ ((temp[0] >> 9) + 4) ^ 0x10;
   temp[0] -= (temp[1] + sum + 11) ^ ((temp[1] << 6) + 2) ^ ((temp[1] >> 9) + 2) ^ 0x20;
   sum -= 0x458BCD42;
  }
  xorm[i] = temp[0];
  xorm[i + 1] = temp[1];
 }
 for (i = 0; i < 6; i++)
  printf("%c%c%c", *((char*)&xorm[i] + 2), *((char*)&xorm[i] + 1), *(char*)&xorm[i]);

}
int main()
{
 decalc();


}
```

## [ACTF新生赛2020]usualCrypt

### 分析

经典的BASE64



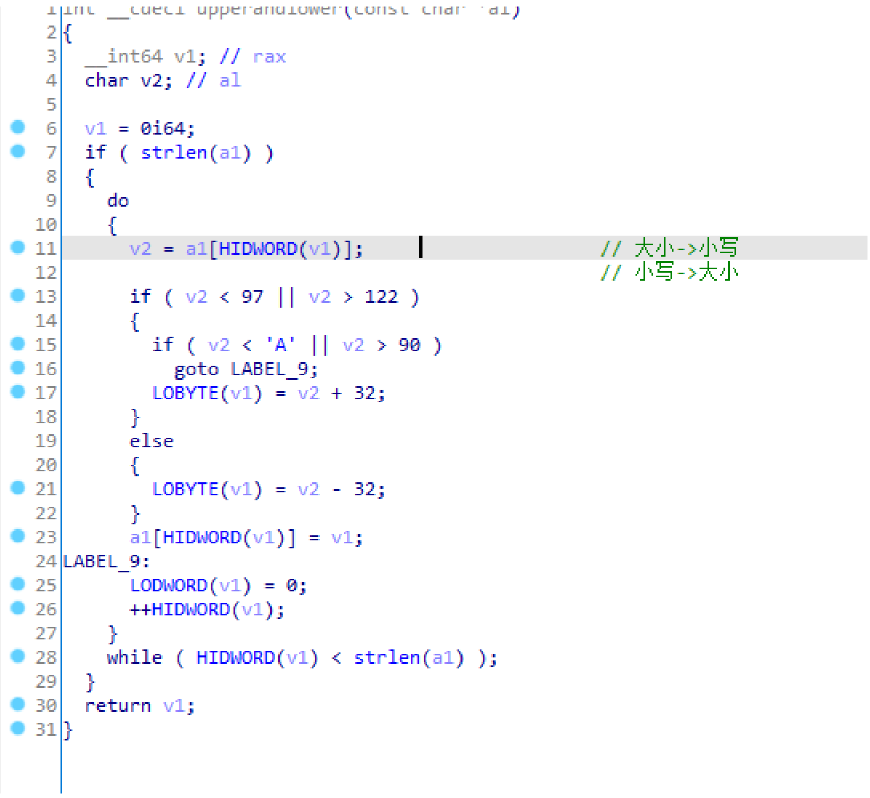
```
int __cdecl sub_401080(int a1, int a2, int a3)
{
  int v3; // edi
  int v4; // esi
  int v5; // edx
  int v6; // eax
  int v7; // ecx
  int v8; // esi
  int v9; // esi
  int v10; // esi
  int v11; // esi
  _BYTE *v12; // ecx
  int v13; // esi
  int v15; // [esp+18h] [ebp+8h]

  v3 = 0;
  v4 = 0;
  sub_401000();
  v5 = a2 % 3;
  v6 = a1;
  v7 = a2 - a2 % 3;
  v15 = a2 % 3;                                 // ABCDEFQRSTUVWXYPGHIJKLMNOZabcdefghijklmnopqrstuvwxyz0123456789+/
  if ( v7 > 0 )
  {
    do
    {
      LOBYTE(v5) = *(a1 + v3);
      v3 += 3;
      v8 = v4 + 1;
      *(v8 + a3 - 1) = BASE64_table_40E0A0[(v5 >> 2) & 0x3F];
      *(++v8 + a3 - 1) = BASE64_table_40E0A0[16 * (*(a1 + v3 - 3) & 3) + ((*(a1 + v3 - 2) >> 4) & 0xF)];
      *(++v8 + a3 - 1) = BASE64_table_40E0A0[4 * (*(a1 + v3 - 2) & 0xF) + ((*(a1 + v3 - 1) >> 6) & 3)];
      v5 = *(a1 + v3 - 1) & 0x3F;
      v4 = v8 + 1;
      *(v4 + a3 - 1) = BASE64_table_40E0A0[v5];
    }
```

```
00001111 sub_401080:35 (401111)
```

动态调试后换表：

ABCDEFQRSTUVWXYPGHIJKLMNOZabcdefghijklmnopqrstuvwxyz0123456789+/

继续分析，发现base64后还有一个大小写颠倒的算法函数：



```
int __cdecl upperandlower(const char *a1)
{
  __int64 v1; // rax
  char v2; // al

  v1 = 0i64;
  if ( strlen(a1) )
  {
    do
    {
      v2 = a1[HIDWORD(v1)];                     // 大小->小写
                                                // 小写->大小
      if ( v2 < 97 || v2 > 122 )
      {
        if ( v2 < 'A' || v2 > 90 )
          goto LABEL_9;
        LOBYTE(v1) = v2 + 32;
      }
      else
      {
        LOBYTE(v1) = v2 - 32;
      }
      a1[HIDWORD(v1)] = v1;
LABEL_9:
      LODWORD(v1) = 0;
      ++HIDWORD(v1);
    }
    while ( HIDWORD(v1) < strlen(a1) );
  }
  return v1;
}
```

## EXP

```
import string
from pwn import *
oldtable="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
newtable="ABCDEFQRSTUVWXYPGHIJKLMNOZabcdefghijklmnopqrstuvwxyz0123456789+/"

dictA={}
for i in range(len(oldtable)):
    dictA[newtable[i]]=oldtable[i]

dictA["="]="="

convert_data="zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9"

ans=""
for i in range(len(convert_data)):
    now=ord(convert_data[i])
    if(now<97 or now>122):# no ||  is or
        if((now<65 or now>90)==False):
            now=now+32
    else:
        now=now-32


    ans=ans+dictA[chr(now)]

print(b64d(ans))
```