

0x01 (dis)con

考察点：构造与析构函数；

查看程序架构



IDA分析

函数不多

The screenshot shows the IDA Pro interface with the following details:

- FUNCTIONS:** A list of functions including:
 - _function_name
 - _init_proc
 - sub_1020
 - sub_1030
 - sub_1040
 - __cxa_finalize
 - _puts
 - _printf
 - _start
 - deregister_tm_clones
 - register_tm_clones
 - _do_global_dtors_aux
 - frame_dummy
 - main
 - start
 - stop
 - stop
 - __libc_csu_init
 - __libc_csu_fini
 - _term_proc
 - puts
 - printf
 - __libc_start_main
 - __imp___cxa_finalize
 - __gnon_start__
- Segments:** .init, .plt, .plt.got, .plt.sec, .plt.sec
- IDA View-A:** Shows the assembly code for the main function:

```
; Attributes: bp-based frame
; int __fastcall main(int argc, const char **argv, const char **envp)
public main
main proc near

var_20= qword ptr -20h
var_14= dword ptr -14h
var_8= qword ptr -8

; _ unwind {
endbr64
push rbp
mov rbp, rsp
sub rsp, 20h
mov [rbp+var_14], edi
mov [rbp+var_20], rsi
mov [rbp+var_8], 0
jmp short loc_1181
```
- Hex View-1:** Shows the hex dump of the assembly code.
- Structures:** None
- Enums:** None
- Graph overview:** Shows a graph with 4 nodes and 23 edges.
- Locals:** loc_1181: cmp [rbp+var_8], 1Ah
jbe short loc_1186
- Registers:** loc_1186: lea rsi, originalString
lea rdi, format, "%s"
mov eax, 0
call _printf
mov eax, 0
leave
ret
; } // starts at 1169
main endp

字符串查看

字符串里并没有flag字样

Address	Length	Type	String
LOAD:0000000... 0000001C	C	/lib64/ld-linux-x86-64.so.2	
LOAD:0000000... 0000000A	C	libc.so.6	
LOAD:0000000... 00000007	C	printf	
LOAD:0000000... 0000000F	C	__cxa_finalize	
LOAD:0000000... 00000012	C	__libc_start_main	
LOAD:0000000... 0000000C	C	GLIBC_2.2.5	
LOAD:0000000... 0000001C	C	_ITM_deregisterTMCloneTable	
LOAD:0000000... 0000000F	C	__gmon_start__	
LOAD:0000000... 0000001A	C	_ITM_registerTMCloneTable	
.rodata:0000... 0000000D	C	hello world!	
.rodata:0000... 0000000F	C	goodbye world!	
.eh_frame:00... 00000006	C	:*3\$\"	

main函数

查看main函数，发现有对originalString的异或操作；

```
IDA View-A Pseudocode-A Strings Hex View-1
1 int __fastcall main(int argc, const char **argv, const char **envp)
2 {
3     unsigned __int64 i; // [rsp+18h] [rbp-8h]
4
5     for ( i = 0LL; i <= 0x1A; ++i )
6         originalString[i] ^= 0xBBu;
7     printf("%s", originalString);
8     return 0;
9 }
```

双击看定义,取出字符序列

```
1 flag_0 = [0x90, 0x92, 0x9e, 0x8e, 0x9e, 0x89, 0x9b, 0xa6, 0x9a, 0xed,
0xed, 0xb9, 0x82, 0xa9, 0x95, 0xb4, 0xae, 0x82, 0xec, 0xae, 0x82, 0xa9,
0xb5, 0xee, 0x82, 0xbb, 0xb1, 0x9c, 0xba, 0xa0]
```

这里直接异或算出结果并不是flag, 而是乱码；

变量交叉引用

交叉引用看originalString；可以看到在start, main, stop三个函数都使用了变量；

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     unsigned __int64 i; // [rsp+18h] [rbp-8h]
4
5     for ( i = 0LL; i <= 0x1A; ++i )
6         originalString[i] ^= 0xBBu;
7     printf("%s", originalString);
8     return 0;
9 }
```

xrefs to originalString

Direction	Type	Address	Text
Up	o	main:loc_1186	lea rdx, originalString
Up	o	main+33	lea rdx, originalString
Do...	o	main+4F	lea rsi, originalString
Do...	o	start:loc_11F9	lea rdx, originalString
Do...	o	start+38	lea rdx, originalString
Do...	o	stop:loc_1251	lea rdx, originalString
Do...	o	stop+38	lea rdx, originalString

start函数

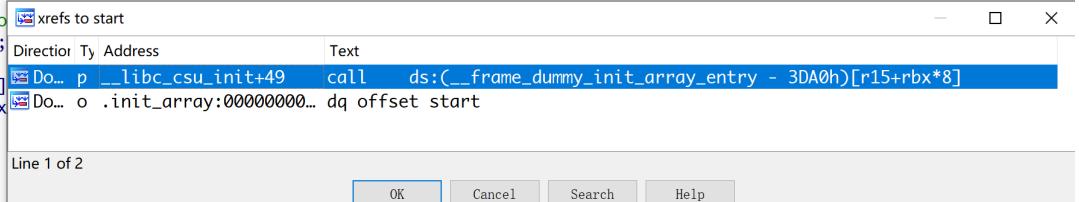
函数处理：

```
1 int start()
2{
3    char *v0; // rax
4    unsigned __int64 i; // [rsp+8h] [rbp-8h]
5
6    LODWORD(v0) = puts("hello world!");
7    for ( i = 0LL; i <= 0x1A; ++i )
8    {
9        v0 = &originalString[i];
10       originalString[i] ^= 0xAAu;
11    }
12    return (int)v0;
13}
```

怎么知道调用了函数呢；看节；参考链接；

1 | <https://b0ldfrev.gitbook.io/note/pwn/zhong-xie-.finiarrray-han-shu-zhi-zhen>

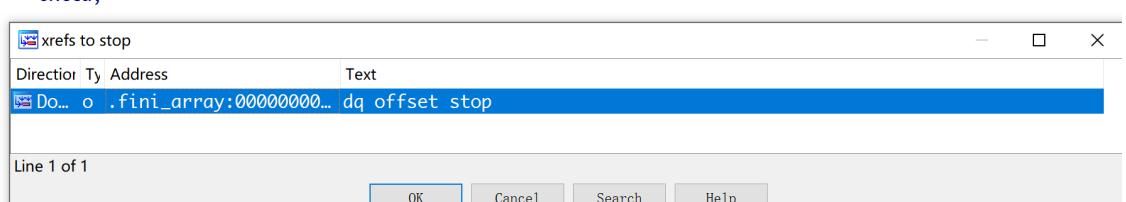
```
1 int start()
2{
3    char *v0; // rax
4    unsigned __int64 i; // [rsp+8h] [rbp-8h]
5
6    LODWORD(v0) = puts("hello") xrefs to start
7    for ( i = 0LL; i <= 0x1A; )
8    {
9        v0 = &originalString[i];
10       originalString[i] ^= 0x
11    }
12    return (int)v0;
13}
```



stop函数

```
1 int stop()
2{
3    char *v0; // rax
4    unsigned __int64 i; // [rsp+8h] [rbp-8h]
5
6    LODWORD(v0) = puts("goodbye world!");
7    for ( i = 0LL; i <= 0x1A; ++i )
8    {
9        v0 = &originalString[i];
10       originalString[i] ^= 0xCCu;
11    }
12    return (int)v0;
13}
```

```
1 int stop()
2{
3    char *v0; // rax
4    unsigned __int64 i; // [rsp+8h] [rbp-8h]
5
6    LODWORD(v0) = puts("goodbye world!");
7    for ( i = 0LL; i <= 0x1A; ++i )
8    {
9        v0 = &originalString[i];
10       originalString[i] ^= 0xCCu;
11    }
12    return (int)v0;
13}
```



函数处理流程

start --> main -> stop , 这个题目没有增加难度，函数里面只进行了xor操作，如果进行多种操作，会涉及到顺序；这里不涉及顺序，直接xor三个hex数据即可；

Get flag

```
1 # Write Python 3 code in this online editor and run it.
2 flag_0 = [0x90, 0x92, 0x9e, 0x8e, 0x9e, 0x89, 0x9b, 0xa6, 0x9a, 0xed,
3           0xed, 0xb9, 0x82, 0xa9, 0x95, 0xb4, 0xae, 0x82, 0xec, 0xae, 0x82, 0xa9,
4           0xb5, 0xee, 0x82, 0xbb, 0xb1, 0x9c, 0xba, 0xa0]
5 #flag_0="MOCSTF{G00d_tHIs_1s_th3_f1Ag}"
6 flag=""
7 for i in flag_0:
8     flag += chr(i^0xaa^0xbb^0xcc)
9 print(flag)
```

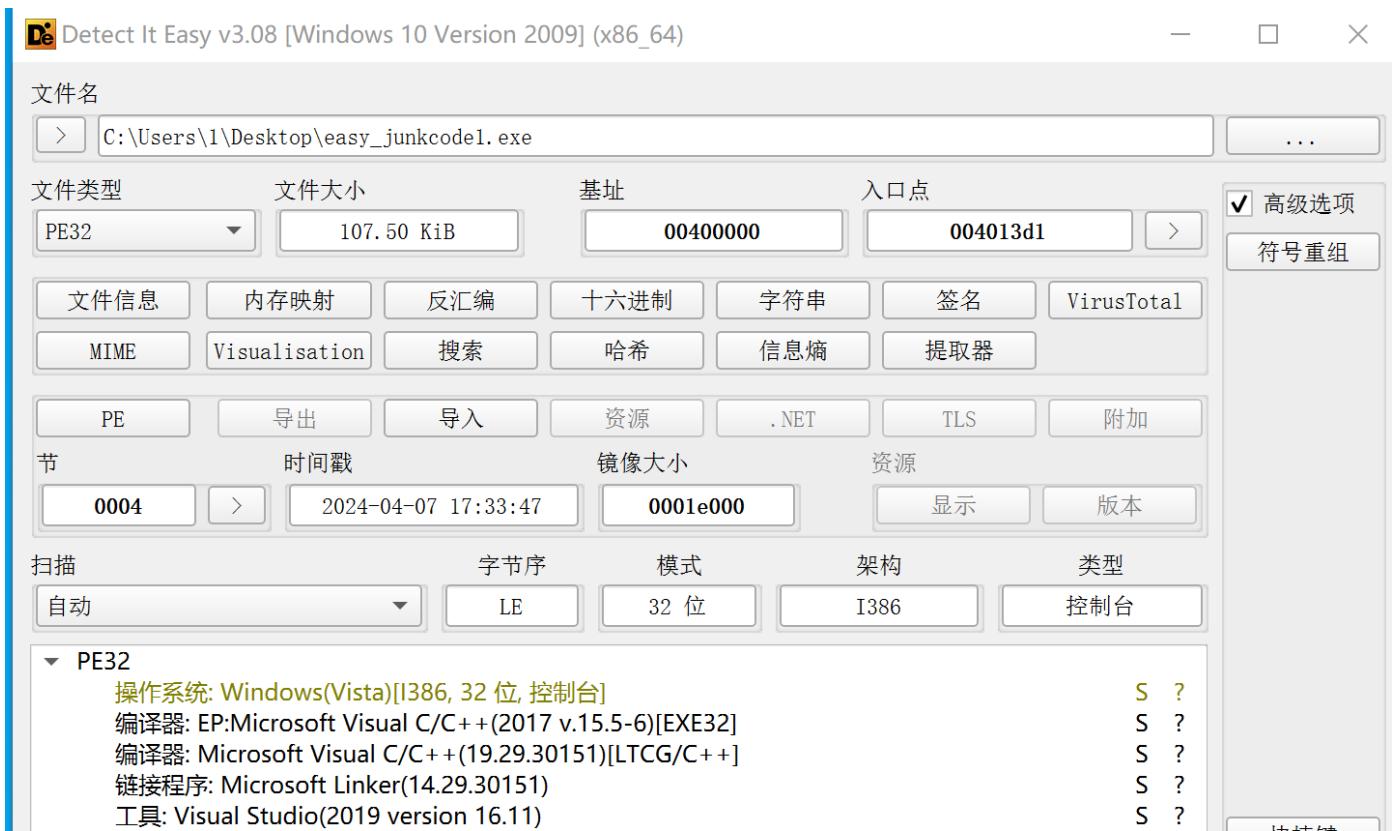
```
1 # Write Python 3 code in this online editor and run it.
2 flag_0 = [0x90, 0x92, 0x9e, 0x8e, 0x9e, 0x89, 0x9b, 0xa6, 0x9a, 0xed, 0x9d, 0xb9, 0x82,
3           0xa9, 0x95, 0xb4, 0xae, 0x82, 0xec, 0xae, 0x82, 0xa9, 0xb5, 0xee, 0x82, 0xbb, 0xb1,
4           0x9c, 0xba, 0xa0]
5 #flag_0="MOCSTF{G00d_tHIs_1s_th3_f1Ag}"
6 flag=""
7 for i in flag_0:
8     flag += chr(i^0xaa^0xbb^0xcc)
9 print(flag)
```

MOCSTF{G00d_tHIs_1s_th3_f1Ag}
==== Code Execution Successful ===

0x02 easy_junkr1.exe

考察点：主程序花指令去除；

程序架构



IDA分析

存在sp指针错误，题目提示了花指令，肯定是加了料了；

```

; Attributes: noreturn bp-based frame
_main proc near
    int __cdecl main(int argc, const char **argv, const char **envp)
    push ebp
    mov ebp, esp
    push ebx
    push esi

```

设置下栈指针，便于清除花指令；

```

.text:00401150 ; ===== S U B R O U T I N E =====
.text:00401150
.text:00401150 ; Attributes: noreturn bp-based frame
.text:00401150
.text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401150 _main proc near
.text:00401150
.text:00401150 argc = dword ptr 8
.text:00401150 argv = dword ptr 0Ch
.text:00401150 envp = dword ptr 10h
.text:00401150
.v.text:00401150 push ebp
.text:00401151 mov ebp, esp
.text:00401153 push ebx
.text:00401154 push esi
.text:00401155 loc_401155:
.text:00401155 push edi
.text:00401156 push offset aHello
.text:00401158 call sub_401040
.text:00401160 add esp, 4
.text:00401163 call sub_40116B
.text:00401168 jmp short loc_401168
.text:00401168 main endp ; sp-analysis.fce
.text:00401168
.text:00401168 ; -----
.text:0040116A db 5
.text:0040116B
.text:0040116B ; ===== S U B R O U T I N E =====
.text:0040116B
.text:0040116B
.text:0040116B sub_40116B proc near

```

IDA Options

Disassembly Analysis Cross-references Strings Browser Graph Lumina Maps

Address representation

- Function offsets
- Stack pointer
- Comments
- Repeatable comments
- Use segment names
- Empty lines
- Borders between data/code (non-graph)
- Basic block boundaries (non-graph)
- Source line numbers
- Try block lines

Display disassembly lines

Number of opcode bytes (non-graph) 0

Instruction indentation (non-graph) 16

Comments indentation (non-graph) 40

Right margin (non-graph) 70

Line prefix example: seg000:0FE4

Low suspiciousness limit 0x401000

High suspiciousness limit 0x41D000

Spaces for tabulation 4

OK Cancel Help

字符串查看

无flag相关字符串提示：

IDA View-A		Pseudocode-A		Strings		Hex View-1	
Address	Length	Type	String				
.rdata:00414...	00000009	C	__basedC				
.rdata:00414...	00000008	C	__cdecl				
.rdata:00414...	00000009	C	__pascal				
.rdata:00414...	0000000A	C	__stdcall				
.rdata:00414...	0000000B	C	__thiscall				
.rdata:00414...	0000000B	C	__fastcall				
.rdata:00414...	0000000D	C	__vectorcall				
.rdata:00414...	0000000A	C	__clrcall				
.rdata:00414...	00000007	C	__eabi				
.rdata:00414...	0000000A	C	__swift_1				
.rdata:00414...	0000000A	C	__swift_2				
.rdata:00414...	00000008	C	ntr64				

先不清除花指令看伪C

看不到flag相关函数；

```

1 int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
2 {
3     char v3; // [esp+0h] [ebp-Ch]
4
5     while ( 1 )
6     {
7         sub_401040("Hello, junkcode1!\n", v3);
8         sub_40116B();
9     }
10 }
```



```

1 void sub_40116B()
2 {
3     char *retaddr; // [esp+0h] [ebp+0h]
4
5     ++retaddr;
6 }
```

分析红框内代码，影响了伪C代码的生成；

```

.text:00401155    loc_401155:          ; CODE XREF: _main+18+j
.text:00401155  00C                 push   edi
.text:00401156  010                 push   offset aHelloJunkcode1 ; "Hello, junkcode1!\n"
.text:0040115B  014                 call   sub_401040
.text:00401160  014                 add    esp, 4
.text:00401163  010                 call   sub_40116B
.text:00401168  010                 jmp    short loc_401155
_main             endp ; sp-analysis failed
.text:00401168
.text:00401168
.text:00401168
.text:00401168
.text:00401168 db 5
.text:00401168
.text:00401168 ; ===== S U B R O U T I N E =====
.text:00401168
.text:00401168
.text:00401168
.text:00401168 ; void sub_40116B()
.text:00401168 sub_40116B proc near ; CODE XREF: _main+13+p
.text:00401168  000                 add    dword ptr [esp+0], 1
.text:0040116F  000                 retn
.text:0040116F sub_40116B endp
.text:0040116F

```

直接call了sub_40116B，中间的jump和db定义很多余；将灰色代码nop掉；

```

.text:00401160
.text:00401163 add esp, 4
.text:00401168 |call sub_40116B
.text:00401168 jmp short loc_401155
_main endp ; sp-analysis failed
.text:00401168
.text:00401168
.text:00401168 db 5
.text:00401168
.text:00401168 ; ===== S U B R O U T I N E =====
.text:00401168
.text:00401168
.text:00401168
.text:00401168 sub_40116B proc near ; CODE XREF: _main+13+p
.text:00401168  000                 add    dword ptr [esp+0], 1
.text:0040116F  000                 retn
.text:0040116F sub_40116B endp
.text:0040116F

```

undefine下

text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
text:00401150 _main proc near
text:00401150
text:00401150 argc = dword ptr 8
text:00401150 argv = dword ptr 0C
text:00401150 envp = dword ptr 10
text:00401150
text:00401150 push ebp
text:00401151 mov ebp, es
text:00401153 push ebx
text:00401154 push esi
text:00401155 push edi
text:00401156 push offset sub_401080
text:00401157 call sub_40116B
text:00401160 add esp, 4
text:00401163 nop
text:00401164 nop
text:00401165 nop
text:00401166 nop
text:00401167 nop
text:00401168 nop
text:00401169 nop
text:00401169 _main endp ; sp-analysis failed
text:00401169
text:0040116A nop
text:0040116B
text:0040116B ; ===== S U B R O U T I N E =====
text:0040116B
text:0040116B
text:0040116B sub_40116B proc near
text:0040116B nop
text:0040116C nop
text:0040116D nop
text:0040116E nop
text:0040116F nop
text:0040116F sub_40116B endp
text:0040116F
text:00401170 call sub_401080
text:00401175 xor eax, eax
text:00401177 pop edi
text:00401178 pop esi
text:00401179 pop ebx
text:0040117A pop ebp
text:0040117B ret

```

.align 10h
_main db 55h ; U ; CODE XREF: __scrt_common_main_seh(void)+F5:p
db 8Bh
db 0ECh
db 53h ; S
db 56h ; V
db 57h ; W
db 68h ; h
db 10h ; OFF32 SEGDEF [_rdata,419810]
db 98h
db 41h ; A
db 0
db 0E8h
db 0E0h
db 0FEh
db 0FFh
db 0FFh
db 83h
db 0C4h
db 4
db 90h
db 90h
db 90h
db 90h
.dbx 00401167 db 90h
.dbx 00401168 db 90h
.dbx 00401169 db 90h
.dbx 0040116A db 90h
.dbx 0040116B db 90h
.dbx 0040116C db 90h
.dbx 0040116D db 90h
.dbx 0040116E db 90h
.dbx 0040116F db 90h
.dbx 00401170 db 0E8h
.dbx 00401171 db 0Bh
.dbx 00401172 db 0FFh
.dbx 00401173 db 0FFh
.dbx 00401174 db 0FFh
.dbx 00401175 db 33h ; 3
.dbx 00401176 db 0C0h
.dbx 00401177 db 5Fh ; -
.dbx 00401178 db 5Eh ; ^
.dbx 00401179 db 5Bh ; [
.dbx 0040117A db 5Dh ; ]
.dbx 0040117B db 0C3h

```

转代码后创建函数

```

.text:00401150
.text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
_main:00401150     push    ebp
                   mov     ebp, esp
                   push    ebx
                   push    esi
                   push    edi
                   push    off
                   push    sub
                   call    sub_401080
                   add    esp, 4
                   nop
                   call    sub_401080
                   xor    eax, eax
                   pop    edi
                   pop    esi
                   pop    ebx
                   pop    ebp
                   ret

```

右键菜单：Create function... (P)

创建完毕

```

.text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401150 _main
    proc near                ; CODE XREF: __scrt_common_main_seh(void)+F5.p
    .text:00401150     push  ebp
    .text:00401151     mov   ebp, esp
    .text:00401153     push  ebx
    .text:00401154     push  esi
    .text:00401155     push  edi
    .text:00401156     push  offset aHelloJunkcode1 ; "Hello, junkcode1!\n"
    .text:00401158     call  sub_401040
    .text:00401160     add   esp, 4
    .text:00401163     nop
    .text:00401164     nop
    .text:00401165     nop
    .text:00401166     nop
    .text:00401167     nop
    .text:00401168     nop
    .text:00401169     nop
    .text:0040116A     nop
    .text:0040116B     nop
    .text:0040116C     nop
    .text:0040116D     nop
    .text:0040116E     nop
    .text:0040116F     nop
    .text:00401170     call  sub_401080
    .text:00401175     xor   eax, eax
    .text:00401177     pop   edi
    .text:00401178     pop   esi
    .text:00401179     pop   ebx
    .text:0040117A     pop   ebp
    .text:0040117B     retn
    .text:0040117B _main
    endp

```

看伪C

IDA View-A Pseudocode-A Hex View-1

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     sub_401040("Hello, junkcode1!\n");
4     sub_401080();
5     return 0;
6 }

```

IDA View-A Pseudocode-A Hex View-1 Structures

```

1 unsigned int sub_401080()
2 {
3     unsigned int result; // eax
4     int v1; // [esp+8h] [ebp-24h]
5     unsigned int i; // [esp+Ch] [ebp-20h]
6     char *v3; // [esp+10h] [ebp-1Ch]
7     int v5[4]; // [esp+18h] [ebp-14h]
8
9     v5[0] = 49;
10    v5[1] = 65;
11    v5[2] = 81;
12    v5[3] = 97;
13    v1 = 0;
14    result = sub_401040(&unk_41980C);
15    for ( i = 0; ; ++i )
16    {
17        v3 = byte_41B8B0;
18        do
19            LOBYTE(result) = *v3;
20            while ( *v3++ );
21            if ( i >= v3 - &byte_41B8B0[1] )
22                break;
23            byte_41B8B0[i] ^= LOBYTE(v5[v1++]);
24            if ( v1 == 4 )
25                v1 = 0;
26            result = i + 1;
27    }
28    return result;
29}

```

查看取出数据进行异或运算；

功能程序分析

byte_41B8B0 和 v5进行的异或操作

Get flag

```
1 flag_0=[0x7c, 0xe, 0x12, 0x32, 0x72, 0x15, 0x17, 0x1a, 0x72, 0x71,
2 0x3f, 0x6, 0x43, 0x0, 0x25, 0x14, 0x5d, 0x20, 0x25, 0x8, 0x1, 0x2f,
3 0x22, 0x3e, 0x68, 0x71, 0x24, 0x3e, 0x5a, 0x2f, 0x61, 0x16, 0x6e, 0x2b,
4 0x24, 0x2f, 0x5a, 0x1e, 0x32, 0x51, 0x55, 0x24, 0x2c]
5
6 flag_2=[0x31, 0x41, 0x51, 0x61]
7 flag=""
8 cnt = 0
9 for i in flag_0:
10     flag += chr(i^flag_2[cnt])
11     cnt = cnt+1
12     if cnt==4:
13         cnt = 0
14
15 print(flag)
```

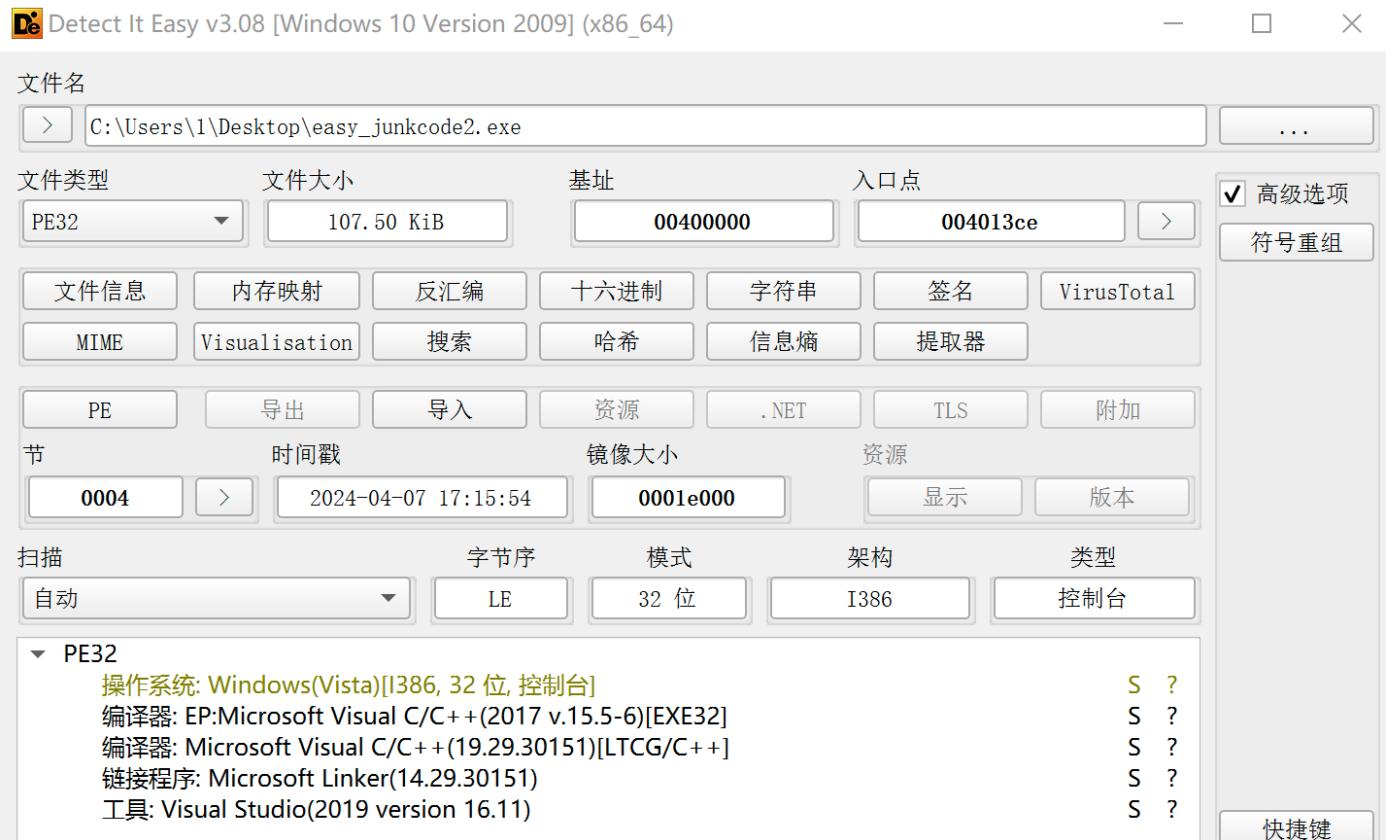
```
1 flag_0=[0x7c, 0xe, 0x12, 0x32, 0x72, 0x15, 0x17, 0x1a, 0x72, 0x71, 0x3f, 0x6, 0x43,
2 0x0, 0x25, 0x14, 0x5d, 0x20, 0x25, 0x8, 0x1, 0x2f, 0x22, 0x3e, 0x68, 0x71, 0x24,
3 0x3e, 0x5a, 0x2f, 0x61, 0x16, 0x6e, 0x2b, 0x24, 0x2f, 0x5a, 0x1e, 0x32, 0x51, 0x55
4 , 0x24, 0x2c]
5
6 flag_2=[0x31, 0x41, 0x51, 0x61]
7 flag=""
8 cnt = 0
9 for i in flag_0:
10     flag += chr(i^flag_2[cnt])
11     cnt = cnt+1
12     if cnt==4:
13         cnt = 0
14
15 print(flag)
```

M0CSCTF{C0ngrAtulati0ns_Y0u_kn0w_juNk_c0de}
==== Code Execution Successful ===

0x03 easy_junkr2.exe

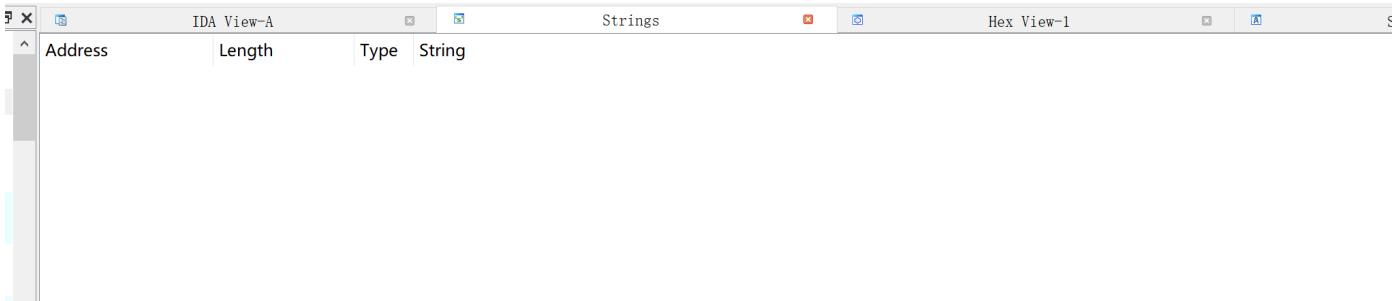
考察点：子程序花指令去除；

程序架构



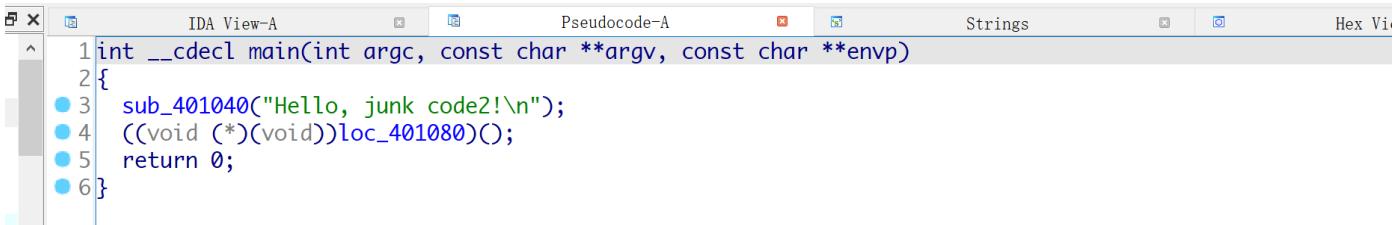
IDA分析

字符串查看



看主程序查看伪C

发现有loc标号；进去看看；



似曾相识的一幕

```

.text:00401080 loc_401080:          ; CODE XREF: _main+10:p
.text:00401080     push    ebp
.text:00401081     mov     ebp, esp
.text:00401083     sub     esp, 2Ch
.text:00401086     mov     eax, __security_cookie
.text:00401088     xor     eax, ebp
.text:0040108D     mov     [ebp-4], eax
.text:00401090     push    ebx
.text:00401091     push    esi
.text:00401092     push    edi
.text:00401093     mov     dword ptr [ebp-14h], 31h ; '1'
.text:0040109A     mov     dword ptr [ebp-10h], 41h ; 'A'
.text:004010A1     mov     dword ptr [ebp-0Ch], 51h ; 'Q'
.text:004010A8     mov     dword ptr [ebp-8], 61h ; 'a'
.text:004010AF
.text:004010AF loc_4010AF:          ; CODE XREF: .text:004010C8;j
.text:004010AF     mov     dword ptr [ebp-24h], 0
.text:00401086     push    offset unk_41980C
.text:004010BB     call    sub_401040
.text:004010C0     add     esp, 4
.text:004010C3     call    sub_4010CB
.text:004010C8     jmp    short near ptr loc_4010AF+6
.text:004010C8 ; -----
.text:004010CA     db 5
.text:004010CB
.text:004010CB ; ===== S U B R O U T I N E =====
.text:004010CB
.text:004010CB
.text:004010CB
.text:004010CB sub_4010CB    proc near             ; CODE XREF: .text:004010C3:p
.text:004010CB     add     dword ptr [esp+0], 1
.text:004010CF     retn
.text:004010CF sub_4010CB    endp
.text:004010CF
.text:004010D0 ; -----
.text:004010D0     mov     dword ptr [ebp-20h], 0
.text:004010D7     jmp    short loc_4010E2
.text:004010D9 ; -----

```

去除花指令

nop掉

```

.text:004010C0     add    esp, 4
.text:004010C3     call   sub_4010CB
.text:004010C8     jmp   short near ptr loc_4010AF+6
.text:004010C8 ; -----
.text:004010CA     db 5
.text:004010CB
.text:004010CB ; ===== S U B R O U T I N E =====
.text:004010CB
.text:004010CB
.text:004010CB sub_4010CB    proc near             ; CODE XREF: .text:004010C3↑p
.text:004010CB     add     dword ptr [esp+0], 1
.text:004010CF     retn
.text:004010CF sub_4010CB    endp
.text:004010CF

```

```

. text:00401080
    push    ebp
    mov     ebp, esp
    sub    esp, 2Ch
    mov    eax, __security_cookie
    xor    eax, ebp
    mov    [ebp+var_4], eax
    push    ebx
    push    esi
    push    edi
    mov    [ebp+var_14], 31h ; '1'
    mov    [ebp+var_10], 41h ; 'A'
    mov    [ebp+var_C], 51h ; 'Q'
    mov    [ebp+var_8], 61h ; 'q'
    mov    [ebp+var_24], 0
    push    offset unk_41980C
    call    sub_401040
    add    esp, 4
    nop
    mov    [ebp+var_20], 0
    jmp    short loc_4010E2
.loc_4010D9; ; CODE XREF: sub_401080:loc_401145+j
.loc_4010D9    mov    eax, [ebp+var_20]
.loc_4010DC    add    eax, 1

```

再看伪C

相同的算法和操作；

```

1 unsigned int sub_401080()
2 {
3     unsigned int result; // eax
4     int v1; // [esp+14h] [ebp-24h]
5     unsigned int i; // [esp+18h] [ebp-20h]
6     char *v3; // [esp+1ch] [ebp-1Ch]
7     int v5[4]; // [esp+24h] [ebp-14h]
8
9     v5[0] = 49;
10    v5[1] = 65;
11    v5[2] = 81;
12    v5[3] = 97;
13    v1 = 0;
14    result = sub_401040(&unk_41980C);
15    for ( i = 0; ; ++i )
16    {
17        v3 = byte_41B8B0;
18        do
19            LOBYTE(result) = *v3;
20            while ( *v3++ );
21            if ( i >= v3 - &byte_41B8B0[1] )
22                break;
23            byte_41B8B0[i] ^= LOBYTE(v5[v1++]);
24            if ( v1 == 4 )
25                v1 = 0;
26            result = i + 1;
27    }
28    return result;
29}

```

Get flag

取出数据异或运算；

```

1 flag_0=[0xe6, 0x81, 0xac, 0x42, 0xe8, 0x9a, 0xa9, 0x6a, 0xe3, 0x8f,
2   0x87, 0x70, 0xf4, 0xa4, 0x9a, 0x5f, 0xc0, 0xad, 0xdf, 0x75, 0xce, 0x91,
3   0xde, 0x7f, 0xf4, 0xbd, 0x9a, 0x27, 0xf4, 0xad, 0xdf, 0x75, 0xce, 0xb3]
4
5 flag_2=[0xab, 0xce, 0xef, 0x11]
6 flag=""
7 cnt = 0
8 for i in flag_0:
9     flag += chr(i^flag_2[cnt])
10    cnt = cnt+1
11    if cnt==4:
12        cnt =0
13
14 print(flag)

```

```

1 flag_0=[0xe6, 0x81, 0xac, 0x42, 0xe8, 0x9a, 0xa9, 0x6a, 0xe3, 0x8f, 0x87, 0x70, 0xf4,
2   0xa4, 0x9a, 0x5f, 0xc0, 0xad, 0xdf, 0x75, 0xce, 0x91, 0xde, 0x7f, 0xf4, 0xbd, 0x9a
3   , 0x27, 0xf4, 0xad, 0xdf, 0x75, 0xce, 0xb3]
4
5 flag_2=[0xab, 0xce, 0xef, 0x11]
6 flag=""
7 cnt = 0
8 for i in flag_0:
9     flag += chr(i^flag_2[cnt])
10    cnt = cnt+1
11    if cnt==4:
12        cnt =0
13
14 print(flag)

```

MOCSCTF{HAha_juNkc0de_1n_su6_c0de}
==== Code Execution Successful ===

0x04 easy_rev

考察点：自定义函数、花指令去除

与上类似，加了不同花指令；取出后算法相同；

Get flag

```

1 flag_0=[0x7f, 0xc, 0x17, 0x36, 0x71, 0x17, 0x12, 0x1e, 0x61, 0x36,
2 0x37, 0x6, 0x1, 0x30, 0x27, 0x3a, 0x41, 0x2e, 0x37, 0x3a, 0x55, 0x73,
3 0x20, 0x3a, 0x3, 0x17, 0x3b, 0x18]
4
5 flag=""
6 cnt = 0
7 for i in flag_0:
8     flag += chr(i^flag_2[cnt])
9     cnt = cnt+1
10    if cnt==4:
11        cnt =0
12
13
14 print(flag)

```

```

1 # Online Python compiler (interpreter) to run Python online.
2 # Write Python 3 code in this online editor and run it.
3 flag_0=[0x7f, 0xc, 0x17, 0x36, 0x71, 0x17, 0x12, 0x1e, 0x61, 0x36, 0x37, 0x6, 0x1,
4 0x30, 0x27, 0x3a, 0x41, 0x2e, 0x37, 0x3a, 0x55, 0x73, 0x20, 0x3a, 0x3, 0x17, 0x3b,
5 0x18]
6
7 flag_2=[0x32, 0x43, 0x54, 0x65]
8 flag=""
9 cnt = 0
10 for i in flag_0:
11     flag += chr(i^flag_2[cnt])
12     cnt = cnt+1
13     if cnt==4:
14         cnt =0
15
16 print(flag)

```

M0CSCTF{Succ3ss_sm0t_1To}
==== Code Execution Successful ===

0x05 Little hard

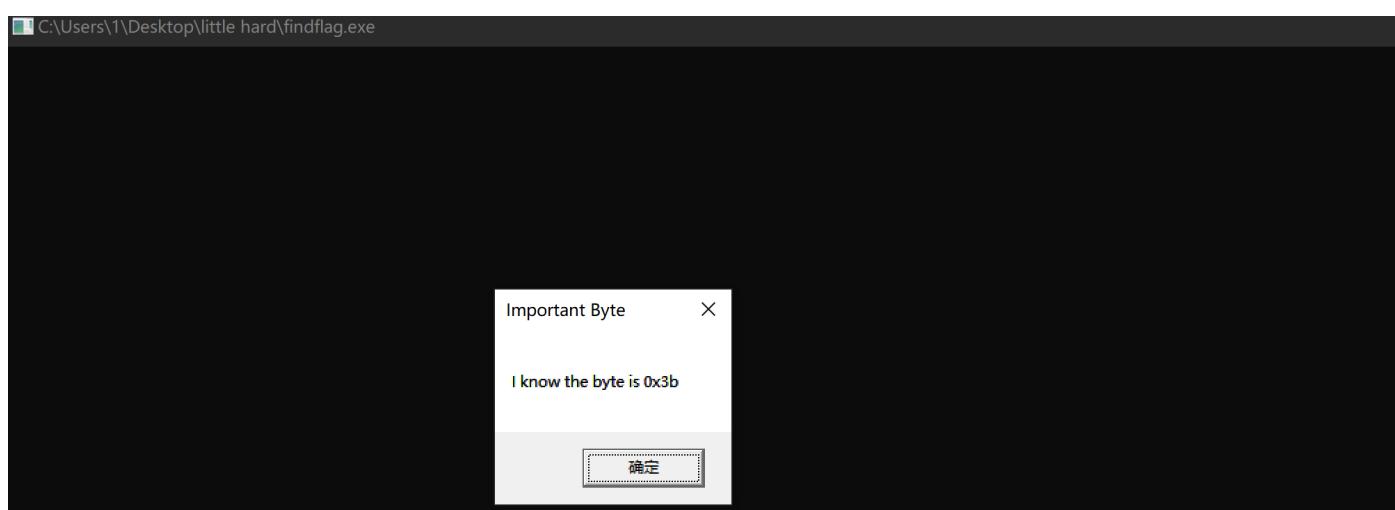
原题思路来自xx木马，实际findflag.exe是加载器，findflag.exe通过内存加载find.inc（exe文件，可以直接改后缀运行它）；而find.inc通过手工加载dll文件执行其导出函数；findflag.exe做了VMP加壳，其实这个是误导；关键点在dll文件中的resource节；这也是恶意木马常用的隐藏payload的手段；通过运行findflag.exe拿到加密key: 0x3b；通过单字节（除0x00外）解密resource数据，直接命令行运行获取flag；

详细分析过程

查看文件加了壳；

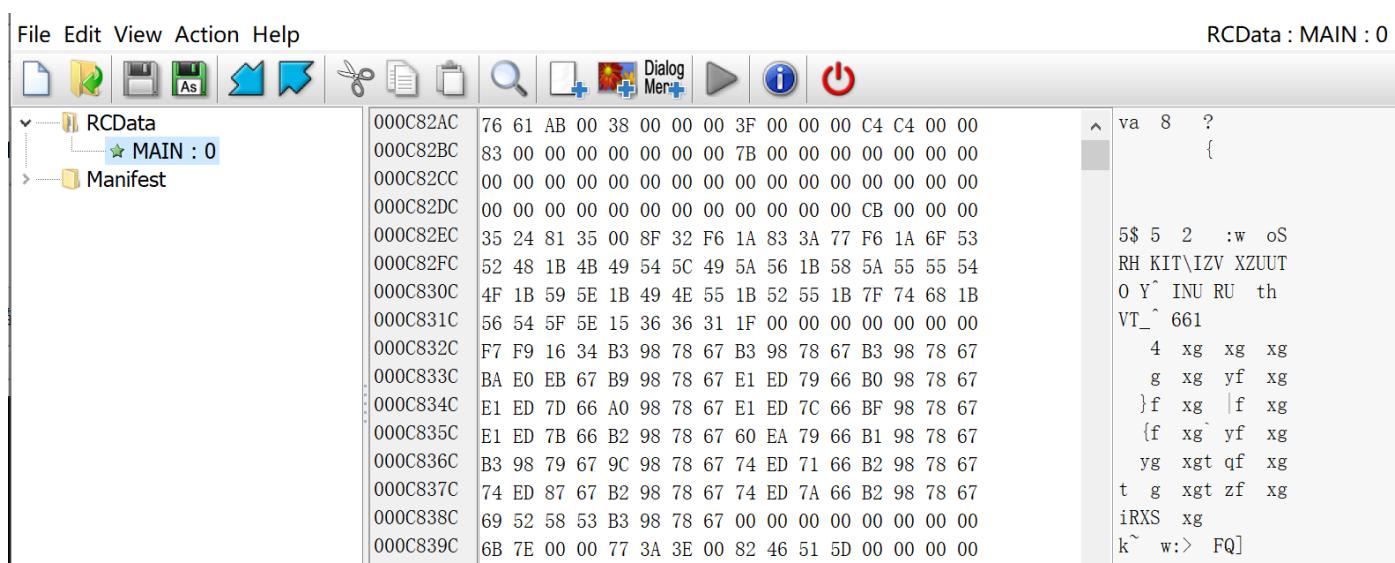


运行提示重要字节是 0x3b；



IDA没发现有用线索；

查看dll文件； resourceheader看下；发现疑似PE文件头被加密；



dump下来；运行脚本解密；

Get flag

```
1 def modify_exe(input_file, output_file):
2     with open(input_file, 'rb') as f:
3         content = bytearray(f.read())
4
5     for i in range(len(content)):
6         if content[i] != 0x3b and content[i] != 0x00:
7             content[i] ^= 0x3b
8
9     with open(output_file, 'wb') as f:
10        f.write(content)
11
12 if __name__ == "__main__":
13     input_file = "MAIN.exe"
14     output_file = "output.exe"
15     modify_exe(input_file, output_file)
16     print("Modification complete.")
17
```

前面提示重要字节0x3b；显然是单字节xor，这里直接异或得不到结果，观察异或后数据，发现需要剔除0x00字节；

```
C:\Users\1\Desktop\little hard>python a.py
Modification complete.

C:\Users\1\Desktop\little hard>"C:\Users\1\Desktop\little hard\output.exe"
MOCSTF{Haha_Y0u_f1nd_th3_f1Ag}
```

或拖入IDA看下；

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char ArgList[31]; // [esp+0h] [ebp-24h] BYREF
4
5     qmemcpy(ArgList, "MOCSTF{Haha_Y0u_f1nd_th3_f1Ag}", sizeof(ArgList));
6     sub_401010("%s", (char)ArgList);
7     return 0;
8 }
```

```
1 MOCSTF{Haha_Y0u_f1nd_th3_f1Ag}
```

0x06 XOR

考察PE识别，题目提示了XOR，但是没提示XOR哪个字节；给的是一堆HEX数据，想到XOR后可能是个二进制文件；

将文本文件复制，粘贴进010editor中；

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	61	76	BC	2C	2F	2C	2C	2C	28	2C	2C	2C	D3	D3	2C	2C	av%/,,,,C,,,Ó,,,
0010h:	94	2C	6C	2C	",,,,,,l,,,,,,												
0020h:	2C	,,,,,,,,,,,,,															
0030h:	2C	2D	2C	2C	,,,,,,,,,,-,,												
0040h:	22	33	96	22	2C	98	25	E1	0D	94	2D	60	E1	0D	78	44	"3-",~%á."-`á.xD
0050h:	45	5F	0C	5C	5E	43	4B	5E	4D	41	0C	4F	4D	42	42	43	E_.^\^CK^\MA.OMBBC
0060h:	58	0C	4F	49	0C	5F	59	42	0C	45	42	0C	68	63	7F	0C	X NT AYR FR hc

```
1 import subprocess
2
3 def xor_file_and_extract_strings(filename, output_dir,
4     strings_output_file):
5     all_strings = []
6
7     with open(filename, 'rb') as f:
8         content = f.read()
9
10    for xor_value in range(1, 256): # 0x01 to 0xff
11        xored_content = bytes([byte ^ xor_value for byte in content])
12        output_file = f"output_file_{xor_value:02X}.bin" # Save with
13        the XOR value in the filename
14        with open(output_file, 'wb') as f_out:
15            f_out.write(xored_content)
16
17    # Example usage:
18    input_filename = "main.bin"
19    output_dir = ""
20    strings_output_file = "all_strings.txt"
21    xor_file_and_extract_strings(input_filename, output_dir,
22        strings_output_file)
```

执行下生成255个文件

■ output_file_FF.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_FE.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_FD.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_FC.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_FB.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_FA.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F9.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F8.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F7.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F6.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F5.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F4.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F3.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F2.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F1.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_F0.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_EF.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_EE.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_ED.bin	2024/6/14 17:38	BIN 文件	18 KB
■ output_file_EC.bin	2024/6/14 17:38	BIN 文件	18 KB

简单方法打包上传威胁情报看下；

找到恶意文件；

或者在linux下直接 file * 命令看下类型；找到目标文件分析；

```
output_file_2A.bin: data
output_file_2B.bin: data
output_file_2C.bin: PE32 executable (console) Intel 80386, for MS Windows
output_file_2D.bin: data
output_file_2E.bin: data
```

此题目实际为PE加载执行shellcode，还原exe后可以直接执行（不可信文件在vm中运行）；也可以还原下shellcode，拖入IDA；

```
1 int sub_0()
2 {
3     _int64 v0; // rax
4     int v1; // eax
5     int result; // eax
6     int __stdcall *v4)(__DWORD, char *, __DWORD, __DWORD); // [esp+10h] [ebp-BCh]
7     char *v5; // [esp+1Ch] [ebp-80h]
8     char *v6; // [esp+20h] [ebp-ACh]
9     int v7; // [esp+24h] [ebp-A8h]
10    char v8[40]; // [esp+28h] [ebp-A4h] BYREF
11    char v9[16]; // [esp+50h] [ebp-7Ch] BYREF
12    char v10[16]; // [esp+60h] [ebp-6Ch] BYREF
13    char v11[12]; // [esp+70h] [ebp-5Ch] BYREF
14    char v12[12]; // [esp+7Ch] [ebp-50h] BYREF
15    __int64 v13; // [esp+88h] [ebp-44h]
16    __int64 i; // [esp+90h] [ebp-3Ch]
17    int __stdcall *v15)(int); // [esp+98h] [ebp-34h]
18    int __stdcall *v16)(char *, char *); // [esp+9Ch] [ebp-30h]
19    unsigned __int16 *v17; // [esp+A0h] [ebp-2Ch]
20    __DWORD *v18; // [esp+A4h] [ebp-28h]
21    __int64 v19; // [esp+A8h] [ebp-24h]
22    __DWORD *v20; // [esp+B0h] [ebp-1Ch]
23    int v21; // [esp+B4h] [ebp-18h]
24    __int16 v22; // [esp+B8h] [ebp-14h]
25    __DWORD *v23; // [esp+BCh] [ebp-10h]
26    __int64 v24; // [esp+C0h] [ebp-Ch]
27    _BYTE *v25; // [esp+C8h] [ebp-4h]
28
29    v24 = 0i64;
30    for ( i = *(int *)(*(_DWORD *)(_readfsdword(0x30u) + 0xC) + 0x14); i; i = *(unsigned int *)i )
31    {
32        v13 = *(int *)(i + 0x28);
33        v22 = *(_WORD *)i + 0x24;
34        v19 = 0i64;
35        do
36        {
37            v19 = (unsigned int)_ROR4_(v19, 0x0);
38            if ( *(unsigned __int8 *)v13 < 0x61u )
39                v0 = *(unsigned __int8 *)v13;
40            else
41
42        }
43        if ( !v16 || !v15 )
44            _debugbreak();
45        strcpy(v12, "USER32.dll");
46        strcpy(v11, "MessageBoxA");
47        strcpy(v8, "MOCSTF{Brut3Forc3_th3_f1Ag_U$3_X0r}");
48        v1 = v16(v12, v11);
49        v4 = (int __stdcall *(_DWORD, char *, __DWORD, __DWORD))v15(v1);
50        if ( !v4 )
51            _debugbreak();
52        result = v4(0, v8, 0, 0);
53        _debugbreak();
54        return result;
55    }
56}
```

```
1 MOCSTF{Brut3Forc3_th3_f1Ag_U$3_X0r}
```

更为简洁的方法，看到文件存在大量2C，直接异或2C即可；这就是为什么一般的恶意软件使用异或算法时会将自身和0x00避过的原因；

