

Diy-易语言花指令引擎与编译结果打乱码

[介绍](#)

[花指令](#)

[调试](#)

[代码编写](#)

[花指令引擎](#)

[HOOK](#)

[小坑](#)

[效果](#)

[小结](#)

[编译结果打乱码](#)

[调试](#)

[代码编写](#)

[小结](#)

[结束了](#)

介绍

作者：XDbg

日期：2023年7月11日01:24:14

测试：易语言5.9.2

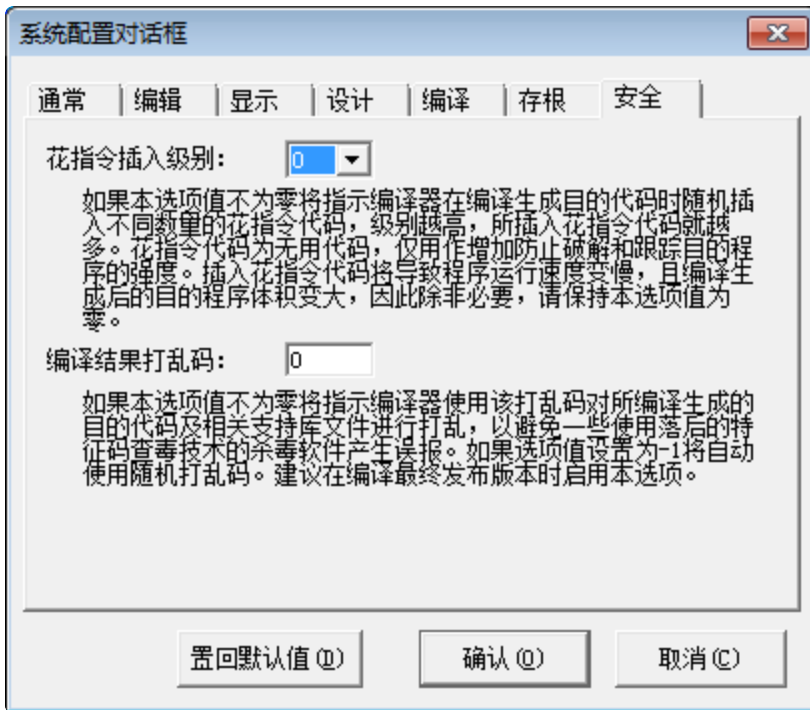
Little chick having chips on my sofa

Bearbricxs take a shit on my sofa

Smudge babies lying on my sofa

Neighborhxxds and kiks singing "so-fa"

今日寂寞难耐，恰好最近在学习易语言，体验了它的花指令引擎和编译结果打乱码，感觉毫无用处，打算给它改改。



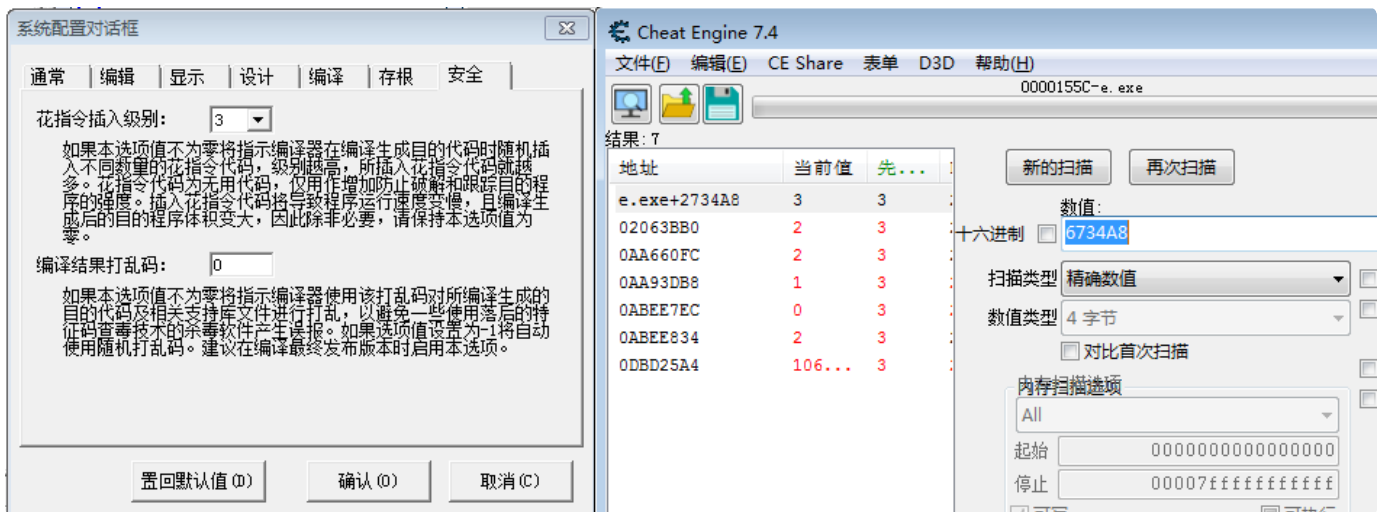
花指令

调试工具：CheatEngine、IDA

代码工具：易语言5.9.2、精易模块、超级HOOK模块

调试

经过反复的搜索，发现存储“花指令插入”的地址在 6734A8，拿到数据了，可以扔进 IDA 了。



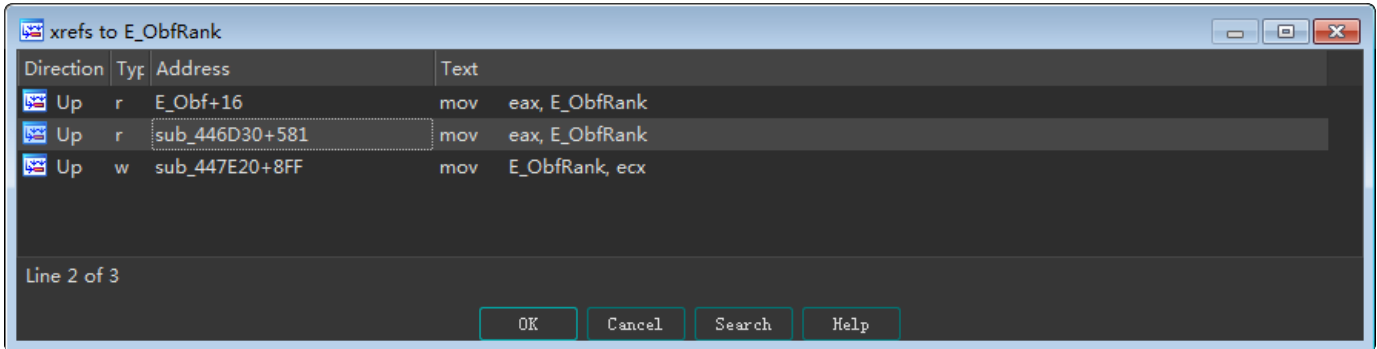
给 6734A8 打上符号，我就命名为 E_OBFRank 了。

```

.data:006734A4 dword_6734A4 dd ? ; DATA XREF: sub_4388C01D1
.data:006734A8 ; WPARAM E_ObfRank ; .text:0048FC251r ...
.data:006734A8 E_ObfRank dd ? ; DATA XREF: E_Obf+161r
.data:006734A8 ; sub_446D30+5811r ...
.data:006734AC dword_6734AC dd ? ; DATA XREF: .text:0042005C1o
.data:006734AC ; sub_446D30+50B1r ...
.data:006734B0 dd ?
.data:006734B4 dword_6734B4 dd ? ; DATA XREF: sub_4041B0:loc_4049971r
.data:006734B4 ; sub_4041B0+7FA1w ...
.data:006734B8 dword_6734B8 dd ? ; DATA XREF: sub_446D30+4B21r
.data:006734B8 ; sub_447E20+8371w ...

```

按下 X 按键，第一个引用函数的符号我已经打上了，看看第二个、第三个引用好了。



第二个引用：看一眼就知道了，是在打开设置->安全弹出花指令设置窗口时的函数，给用户看的。

```

85 sub_5BB734(String);
86 SendMessageA(this[105], 0xF1u, dword_6733A8 == 1, 0);
87 SendMessageA(this[90], 0xF1u, dword_673340 == 1, 0);
88 SendMessageA(this[401], 0xF1u, dword_6733B0 == 1, 0);
89 SendMessageA(this[416], 0xF1u, dword_6733B4 == 1, 0);
90 SendMessageA(this[431], 0xF1u, dword_6733B8 == 1, 0);
91 SendMessageA(this[446], 0xF1u, dword_6734B8 == 1, 0);
92 SendMessageA(this[461], 0xF1u, dword_673324 == 0, 0);
93 SendMessageA(this[476], 0xF1u, dword_673498 != 1, 0);
94 SendMessageA(this[491], 0xF1u, dword_6734AC == 1, 0);
95 if ( dword_673FBC < 6 )
96     CStringArray::SetSize((CStringArray *)&unk_673FB4, 6, -1);
97 sub_5BB734(*(LPCSTR *)dword_673FB8 + 5));
98 if ( dword_673FBC < 9 )
99     CStringArray::SetSize((CStringArray *)&unk_673FB4, 9, -1);
100 sub_5BB734(*(LPCSTR *)dword_673FB8 + 8));
101 SendMessageA(this[506], 0x14Eu, E_ObfRank, 0);
102 sub_5004D0(E_ShellKey, String);
103 sub_5BB734(String);
104 if ( dword_6733BC == 0x80000000 )
105 {
106     sub_448D70(this);
107 }
108 else
109 {
110     sub_448A10(this);
111     SendMessageA(this[596], 0xF1u, dword_6733BC & 1, 0);
112     SendMessageA(this[611], 0xF1u, ((unsigned int)dword_6733BC >> 1) & 1, 0);
113 }

```

第三个引用：结合 IDA 给的“w”信息，意思就是 E_OBFRank 被赋值了，可以猜测它是设置窗口关闭时调用的，即保存配置用的。

```

268 dword_6734B8 = SendMessageA(*(HWND *)this + 446), 0xF0u, 0, 0) == 1;
269 dword_673324 = SendMessageA(*(HWND *)this + 461), 0xF0u, 0, 0) != 1;
270 dword_673498 = SendMessageA(*(HWND *)this + 476), 0xF0u, 0, 0) != 1;
271 dword_6734AC = SendMessageA(*(HWND *)this + 491), 0xF0u, 0, 0) == 1;
272 dword_6734C0 = SendMessageA(*(HWND *)this + 60), 0xF0u, 0, 0) == 1;
273 dword_6733C0 = SendMessageA(*(HWND *)this + 75), 0xF0u, 0, 0) == 1;
274 v20 = SendMessageA(*(HWND *)this + 506), 0x147u, 0, 0);
275 E_ObfRank = v20 != -1 ? v20 : 0;
276 E_ShellKey = sub_4FE2B0((CDialog *)((char *)this + 2116));
277 if ( sub_4FE2B0((CDialog *)((char *)this + 2056)) >= 0 )
278     v21 = sub_4FE2B0((CDialog *)((char *)this + 2056));
279 else
280     v21 = 0;
281 dword_6734A0 = v21 - 6;
282 v22 = dword_673F04 - v21;
283 if ( v22 > 0 )
284 {
285     v23 = v22;
286     do
287     {
288         sub_4836C0(0);
289         --v23;
290     }
291     while ( v23 );

```

现在来看看第一个引用：另外两个已经被排除了，这个大概率是用来添加花指令的函数了。

1. 使用了 E_ObfRank
2. 调用 GetTickCount()
3. 后续还调用 rand()
4. 有 4 个全局变量，里面指向了 4 种花指令模板

根据这些信息，我们已经能够确定其是 添加花指令 的函数了。

```

1 int __stdcall E_Obf(int a1, signed int ObfRank)
2 {
3     int result; // eax
4     DWORD ObfKey; // eax
5     unsigned int ObfKey2; // esi
6     int ObfKey3; // esi
7     int ObfKey4; // edx
8     char *ObfIns; // eax
9     size_t ObfInsLength; // ecx
10    int ObfKey5; // eax
11
12    result = *(_DWORD*)(a1 + 8);
13    if ( *(_DWORD*)(result + 4) == 3 )
14    {
15        result = E_ObfRank;
16        if ( ObfRank <= (int)E_ObfRank )
17        {
18            ObfKey = GetTickCount();
19            dword_672D0C += (ObfKey & 7) + 1;
20            ObfKey2 = dword_672D0C + ObfKey;
21            if ( !dword_672D10 )
22            {
23                srand(ObfKey2);
24                dword_672D10 = 1;
25            }
26            ObfKey3 = rand() ^ ObfKey2;
27            ObfKey4 = ObfKey3 % 10;
28            if ( ObfKey3 % 10 < 0 )
29                ObfKey4 = -ObfKey4;
30            if ( ObfKey4 >= 5 )
31            {
32                if ( ObfKey4 >= 7 )
33                {
34                    if ( ObfKey4 >= 9 )
35

```

000358C6 E_Obf:15 (4358C6)

```

34        if ( ObfKey4 >= 9 )
35        {
36            ObfIns = (char*)&loc_65C403 + 1;
37            ObfInsLength = 10;
38        }
39        else
40        {
41            ObfIns = (char*)&loc_65C3FF + 1;
42            ObfInsLength = 3;
43        }
44    }
45    else
46    {
47        ObfIns = (char*)&loc_65C3FB + 1;
48        ObfInsLength = 3;
49    }
50 }
51 else
52 {
53     ObfIns = (char*)&loc_65C3F7 + 1;
54     ObfInsLength = 2;
55 }
56 sub_4910E0((__DWORD*)(a1 + 12), ObfIns, ObfInsLength); // 用于添加花指令
57 ObfKey5 = ObfKey3 + rand();
58 if ( ObfKey5 < 0 )
59     ObfKey5 = -ObfKey5;
60 result = sub_490B20((__DWORD*)(a1 + 12), loc_65C410[ObfKey5 % 0x42u]); // 修复花指令产生的问题，例如修复指令位置。(Jcc指令)
61 // 写的毫无用处。
62 // 替换成自己的引擎后，需要将其Nop。
63 }
64 }
65 return result;

```

上图的四种花指令模板就不看了，看看该 Hook 哪里？

看到 56 行，这个就是添加花指令的操作（动态调试验证了一下），把它的传参给 Hook 掉就好了。

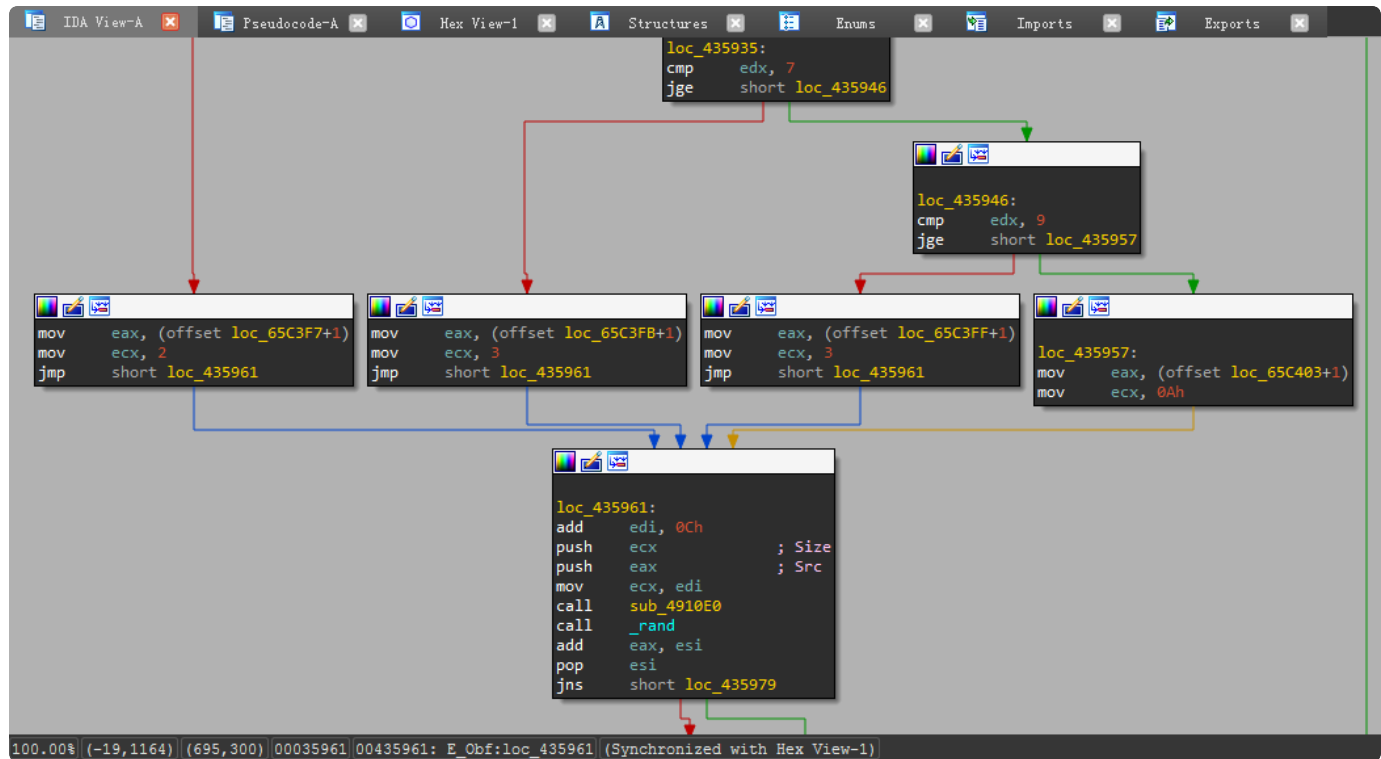
这个函数的传参方式是：__thiscall

1. ecx 是 this 指针
2. 其余的参数通过堆栈传入

打开流程图，看看上面几个 if 语句都跳到了哪里。

都跳到了 435961 这个地址，可以看到 eax 是指向花指令的指针，ecx 是花指令的长度。

```
1 add edi, 0Ch
2 push ecx ; Size
3 push eax ; Src
4 刚好 5 字节，可以在 x86 运行模式下进行 inlineHook.
```



找到了 hook 地址、要 hook 的参数，就可以开始写代码了。

代码编写

程序流程：

1. 初始化花指令引擎
2. 寻找 "e.exe" 进程 id 并获取其进程句柄
3. 利用超级 hook 模块进行 hook "e.exe"

子程序名	返回值类型	公开	易包	备注	
初始化花指令引擎2					
参数名	类型	参考	可空	数组	备注
_strL	文本型			✓	
_obfBin	字节集	✓		✓	

变量名	类型	静态	数组	备注
i	整数型			
x	整数型			
x2	整数型			
x3	整数型			
obfStr	文本型			
table	文本型	✓		

```

table = "1234567890ABCDEF"
--> 变量循环首 (1, 取数组成员数 (_strL), 1, i)
    x2 = 取随机数 (1, 取文本长度 (table))
    x3 = 取随机数 (1, 取文本长度 (table))
    obfStr = _strL [i]
    obfStr = 文本_删除空行 (obfStr)
    obfStr = 文本_替换 (obfStr, , , , "??", 取文本中间 (table, x2, 1) + 取文本中间 (table, x3, 1), , , , , , , , , )
    加入成员 (_obfBin, 字节集_十六进制到字节集 (obfStr))
-- 变量循环尾 ()

```

强化一下花指令：读取随机数量的花指令，然后将其拼接就好了。

子程序名	返回值类型	公开	易包	备注	
随机生成一个花指令_BinEx	字节集				
参数名	类型	参考	可空	数组	备注
Length	整数型	✓			

变量名	类型	静态	数组	备注
i	整数型			
result	字节集			

```

--> 变量循环首 (1, 取随机数 (8, 23), 1, i)
    result = result + 随机生成一个花指令_Bin (Length)
-- 变量循环尾 ()

```

返回 (result)

HOOK

1. 开始 HOOK
2. 等待用户编译代码

开始 HOOK 代码。

```

写入 obf 到易语言 ()
Fuck 易语言没用的 jcc 修复 ()
flagHook = eHook_开始 (processHandle, "00435961", @HOOK 回调, eHookData, 字节集_取长度 ({ 131, 199, 12, 81, 80 }), 2, , 真)

```

HOOK 的回调：把易语言生成的花指令替换成我生成的就好了。

子程序名	返回值类型	公开	易包	备注
HOOK回调				

变量名	类型	静态	数组	备注
n	整数型			

置随机数种子 0

n = 取随机数 (1, 取数组成员数 (eObfBin))

eHook. 写值 (eHookData. 返回地址, "eax", eObfBinRemoteAddr [n])

eHook. 写值 (eHookData. 返回地址, "ecx", 取字节集长度 (eObfBin [n]))

▶ 调试输出 ("远程生成的花指令地址", 进制_十到十六 (eObfBinRemoteAddr [n]))

▶ 调试输出 ("生成的花指令", 字节集_字节集到十六进制 (eObfBin [n]))

▶ 调试输出 ("生成的花指令大小", 取字节集长度 (eObfBin [n]))

小坑

1. 易语言自带的 jcc 修复 (不 nop 掉 与我们的花指令引擎不兼容)
2. 算法有问题, 有些花指令需要修复

易语言有个修复 jcc 指令的函数, 将其 Nop 掉。

子程序名	返回值类型	公开	易包	备注
Fuck易语言没用的jcc修复				

内存. 写字节集 (processId, 进制_十六到十 ("0043598A"), { 144, 144, 144, 144, 144 })

子程序名	返回值类型	公开	易包	备注
UnFuck易语言没用的Jcc修复				

内存. 写字节集 (processId, 进制_十六到十 ("0043598A"), { 82, 232, 144, 177, 5, 0 })

似乎算法写的有问题, 有些花指令需要替换一下。(obf替换)

子程序名	返回值类型	公开	易包	备注
写入obf到易语言				

变量名	类型	静态	数组	备注
i	整数型			

▶ 变量循环首 (1, 取数组成员数 (eObfBin), 1, i)

obf替换 (i, { 122, 1 }, { 235, 1 })

obf替换 (i, { 121, 1 }, { 235, 1 })

加入成员 (eObfBinRemoteAddr, 内存_远程创建内存_字节集 (processHandle, eObfBin [i]))

--- 变量循环尾 0

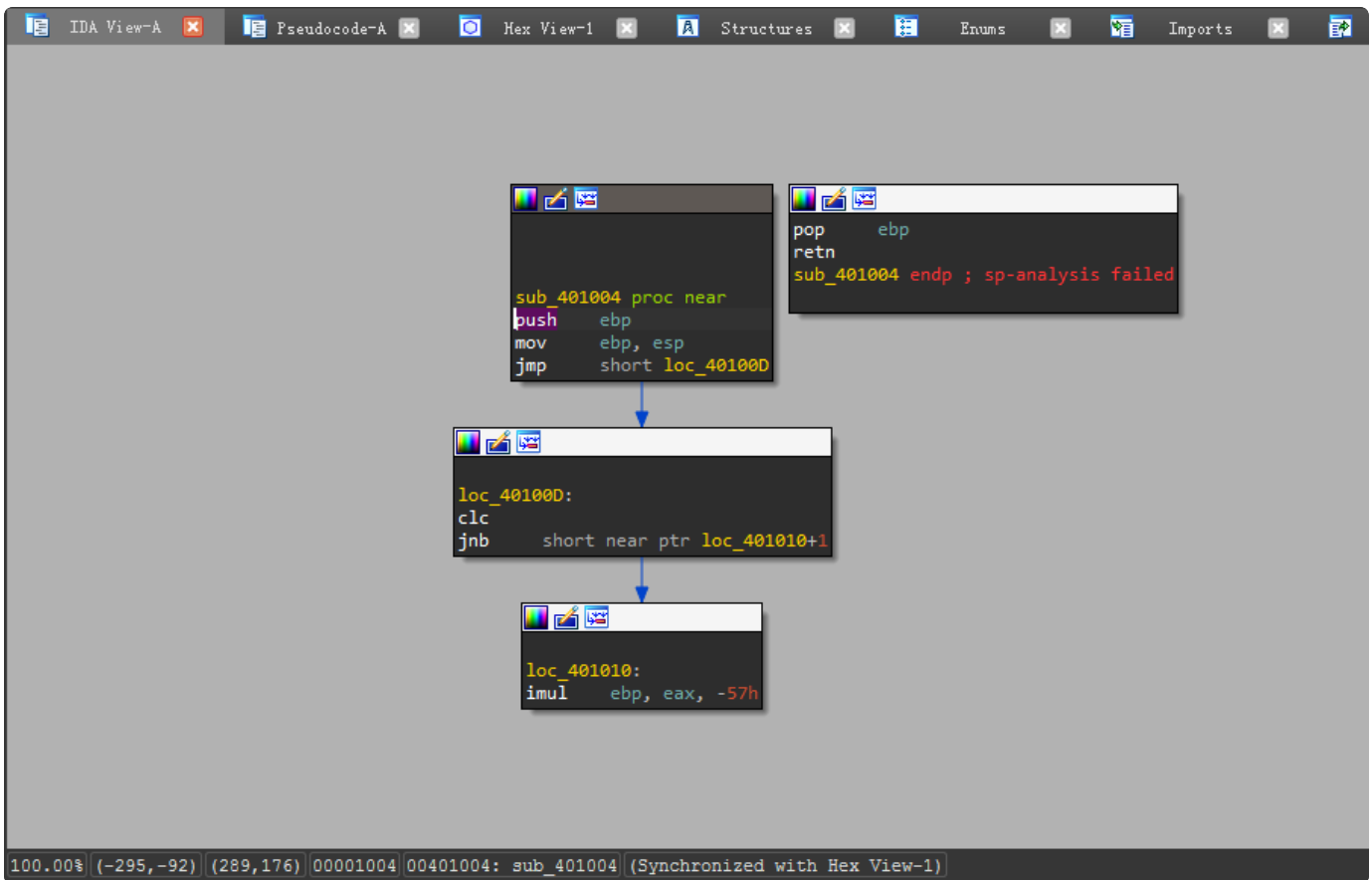
效果

1. IDA 反编译结果
2. IDA Graph
3. 使用 Dejunk 清理花指令（我们的花指令模板来源于它）
4. 使用 E junk code 清理花指令

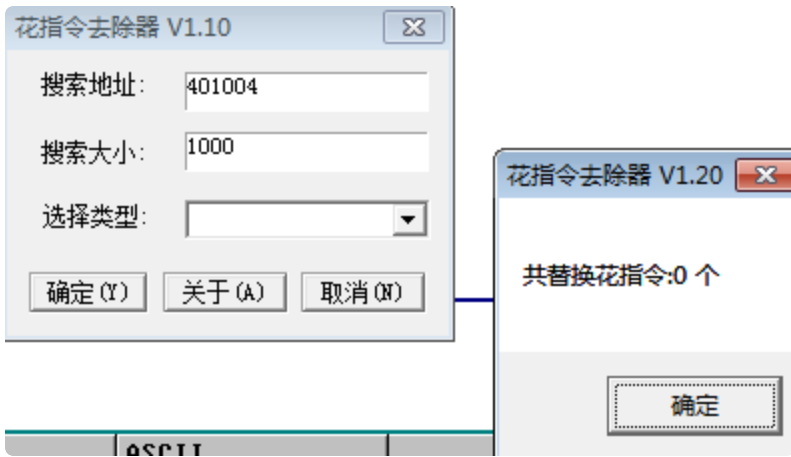
IDA 反编译结果，已经不能看了，需要清除花指令。

```
1 // positive sp value has been detected, the output may be wrong!  
2 void sub_401004()  
3 {  
4   JUMPOUT(0x401011);  
5 }
```

IDA Graph 结果，指令识别失败。



Dejunk 效果如图，配置失败，下一个。（下载下来就这样，尴尬了）



E junk code 效果如图，清理了 107 个花指令，不错不错~

004010BF	55	push ebp	
004010C0	8BEC	mov ebp,esp	
004010C2	81EC 08000000	sub esp,0x8	
004010C8	6A FF	push -0x1	
004010CA	6A 08	push 0x8	
004010CC	68 46000116	push 0x16010046	
004010D1	68 01000152	push 0x52010001	
004010D6	E8 42070100	call 修改易语.0041181D	jmp 到 <修改易语.读取组件属性>
004010DB	83C4 10	add esp,0x10	
004010DE	8945 FC	mov dword ptr ss:[ebp-0x4],eax	kernel32.BaseThreadInitThunk
004010E1	EB 05	jmp short 修改易语.004010E8	
004010E3	49	dec ecx	
004010E4	49	dec ecx	
004010E5	49	dec ecx	
004010E6	49	dec ecx	
004010E7	49	dec ecx	
004010E8	E8 01000000	call 修改易语.004010EE	
004010ED	1083 042406C3	adc byte ptr ds:[ebx-0x3CF9DBFC],al	
004010F3	E8 03000000	call 修改易语.004010FB	
004010F8	25 252583C4	and eax,0xC4832525	
004010FD	04 E9	add al,0xE9	
004010FF	04 00	add al,0x0	
00401101	0000	add byte ptr ds:[eax],al	
00401103	8181 81817C03	add dword ptr ds:[ecx+0x37C8181],0x74DA	
0040110D	FB	sti	
0040110E	EB 03	jmp short 修改易语.00401113	
00401110	BD BDBEB05	mov ebp,0x5EBBDB05	
00401115	05 05050505	add eax,0x05050505	

ecx=00000000

地址	HEX 数据	ASCII
004A8000	E3 13 02 75 CD 45 02 75 3B 48 02 75 7A CD 01 75	? 0 蚱 0 ;H 0 z?u
004A8010	99 13 02 75 00 00 00 00 89 64 86 73 51 15 87 73	? 0 ... 培 唾 Q 啤
004A8020	00 00 00 00 BA D1 D6 74 7C DA D6 74 CC E0 D6 74	... 貉 谦] 溢 t 梯 谦
004A8030	34 8B D6 74 48 63 D9 74 D8 81 D6 74 11 17 D7 74	坡 壤 t Hc 臂 捍 谦 谦
004A8040	AF 87 D6 74 D8 52 D6 74 A1 A4 D6 74 38 72 D6 74	瘰 谦 谦 谦 · 谦 8r 谦
004A8050	53 8A D6 74 FB E4 D6 74 26 89 D6 74 89 56 D6 74	效 埃 t 谦 & 壹 t 嫁 谦
004A8060	CF 72 D6 74 95 BC D6 74 70 79 D6 74 8D BD D6 74	螳 谦 0 啤 谦 · 谦 0 啤 谦
004A8070	85 5A D6 74 A8 AD D6 74 00 60 D6 74 4F 82 D6 74	匍 谦 0 谦 · 谦 0 啤 t
004A8080	69 86 D6 74 05 E4 D6 74 B5 58 D9 74 BE 59 D9 74	i 隐 t 分 t 墨 臂 总 臂
004A8090	C3 5A D9 74 BA 3B D9 74 F1 3B D9 74 2A B8 D6 74	胖 臂 ? 臂 ? 臂 * 钢 t
004A80A0	2A 90 D6 74 19 B7 D6 74 FA 8B D6 74 29 C2 D7 74	* 倍 t 分 t 谦) 伦 t
004A80B0	5C A7 D7 74 48 5F D6 74 E4 7B D6 74 52 5D D6 74	\ e t H 谦 程 谦 B] 谦
004A80C0	70 4F D6 74 E4 85 D6 74 47 BF D6 74 C4 AF D6 74	p 0 谦 铍 谦 C 恐 t 莫 谦
004A80D0	40 A6 D6 74 E5 77 D6 74 66 DD D6 74 17 4F D6 74	@ x t 金 谦 t 黎 t 0 谦

M1 M2 M3 M4 M5 Command:

Remove junk code 107

But。。。

004010E8	E8 01000000	call 修改易语.004010EE
004010ED	1083 042406C3	adc byte ptr ds:[ebx-0x3CF9DBFC],al
004010F3	E8 03000000	call 修改易语.004010FB
004010F8	25 252583C4	and eax,0xC4832525
004010FD	04 E9	add al,0xE9
004010FF	04 00	add al,0x0
00401101	0000	add byte ptr ds:[eax],al
00401103	8181 81817C03	add dword ptr ds:[ecx+0x37C8181],0x74DA
0040110D	FB	sti
0040110E	EB 03	jmp short 修改易语.00401113
00401110	BD BDBDEB05	mov ebp,0x5E8BD05
00401115	25 25252525	and eax,0x25252525
0040111A	EB 08	jmp short 修改易语.00401124
0040111C	3C 3C	cmp al,0x3C
0040111E	3C 3C	cmp al,0x3C
00401120	3C 3C	cmp al,0x3C
00401122	3C 3C	cmp al,0x3C
00401124	EB 06	jmp short 修改易语.0040112C
00401126	7A 7A	jpe short 修改易语.004011A2
00401128	7A 7A	jpe short 修改易语.004011A4
0040112A	7A 7A	jpe short 修改易语.004011A6
0040112C	7C 03	jl short 修改易语.00401131
0040112E	EB 03	jmp short 修改易语.00401133
00401130	3174FB E8	xor dword ptr ds:[ebx+edi*8-0x18],esi

小结

易语言的花指令指令研究到此结束了。

如果想继续写的话，就准备实现如下功能了

1. 自动花指令生成，要求生成最基本的花指令模板超过 1000 个
2. 指令变形
3. 指令膨胀

编译结果打乱码

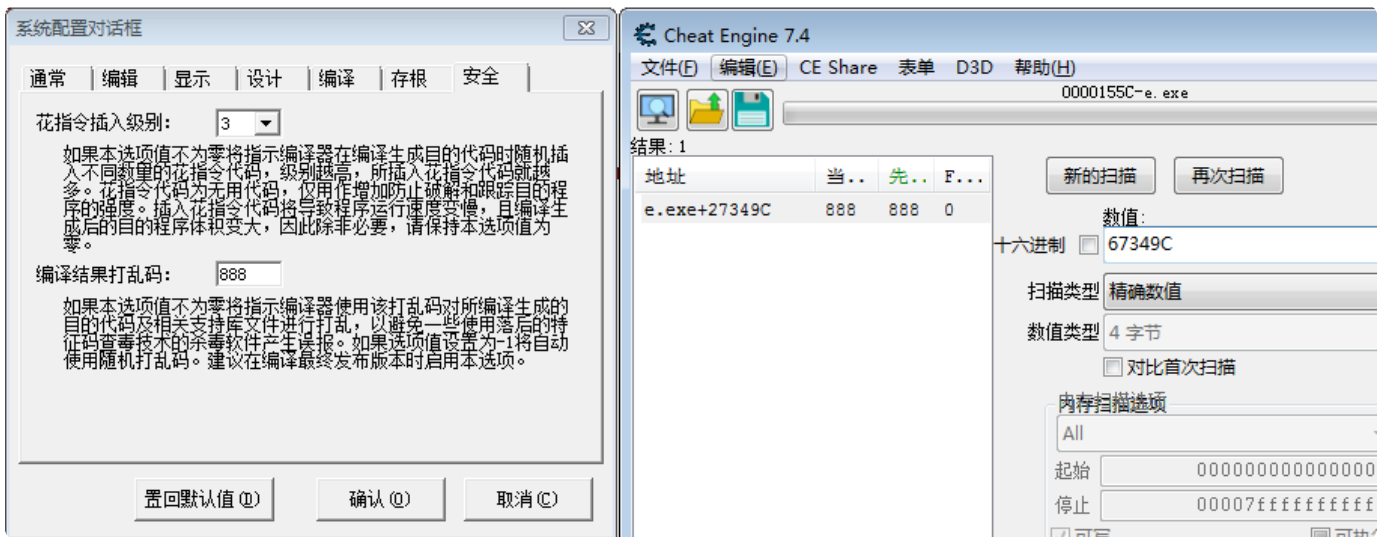
调试工具：CheatEngine、IDA 、X32Dbg（迟早换成CpuDbg->XDbg）

代码工具：易语言5.9.2

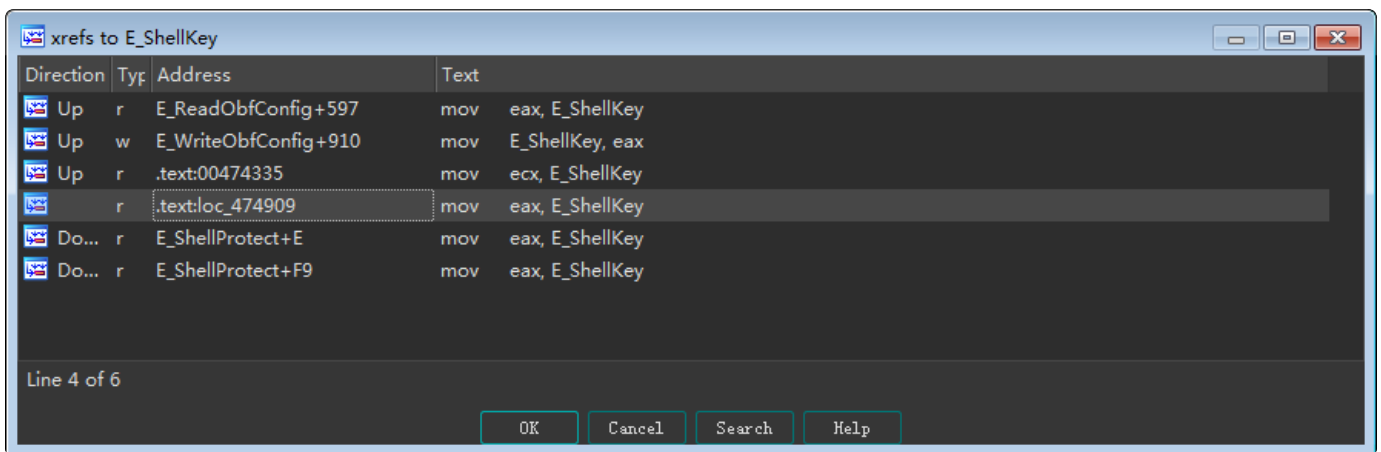
信息收集：FireFox

调试

跟之前一样的操作就好了，拿到编译结果打乱码的地址 67349C。到 IDA 里去看看。



分析方法重复一下就好了，到 E_ShellProtect



发现获取了一个 dll 的函数地址，赶紧扔到 X32Dbg 里调试。

在第 35 行，0047D374 这个地址里下断点，

```

26 v8 = (LPCSTR)sub_5BA720((CString)v12, v3, aCnvpe);
27 this[274] = LoadLibraryA(*v6);
28 sub_5BA491(v12);
29 v16 = -1;
30 sub_5BA491(&v13);
31 v4 = (HMODULE)this[274];
32 if ( v4 )
33 {
34 LABEL_19:
35   cnvpe = GetProcAddress(v4, aCnvpe);
36   this[584] = cnvpe;
37   if ( cnvpe )
38   {
39 LABEL_7:
40     sub_490820(v14);
41     v16 = 1;
42     sub_490840(*(_DWORD*)(a2 + 16) + 10240);
43     ShellKey = E_ShellKey;
44     if ( E_ShellKey == -1 )
45       ShellKey = GetTickCount();
46     v9 = *(_DWORD*)(a2 + 16);
47     if ( v9 )
48       v10 = *(_DWORD*)(a2 + 8);
49     else
50       v10 = 0;
51     v11 = ((int (__stdcall*)(int, int, DWORD, int, int))this[584])(v10, v9, ShellKey, v15 != 0 ? v14[2] : 0, v15);
52     if ( v11 >= 0 && v15 >= v11 )
53     {
54       sub_490910((int)v14, v11);
55       sub_490CC0(a2);
56       v16 = -1;

```

发现信息，cnvpe.cnvpe，随后拿到 FireFox 里搜索。

```

EAX 05602610 <cnvpe.cnvpe>
EBX 0000000F
ECX 05600000 cnvpe.05600000
EDX 05600000 cnvpe.05600000
EBP 0018F7C8
ESP 0018E688 &"@棧"
ESI 006734C8 e.006734C8
EDI 00000000

EIP 0047D374 e.0047D374

EFLAGS 00000300
ZF 0 PF 0 AF 0
QE 0 SE 0 DF 0
CF 0 TF 1 IF 1

LastError 00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)

GS 002B FS 0053

```

搜索到的信息如下，发现这位坛友已经研究过易语言了，并知道 cnvpe.cnvpe 是易语言的支持库里的。提供的接口叫做: 可执行文件数据转换支持

<https://zhuanlan.zhihu.com/p/569792913>

[cnvpe.fne 是病毒吗?我在用mcafee查毒时,找到了这个...](#)

2个回答 · 回答时间: 2009年3月8日

最佳答案: 这个其实应该是易语言的运行库文件,运行用易语言写的程序时,就会自动释放出这个文件。不知道易语言用了什么技术,好多杀毒软件都误报病毒。如果你程序的来源可靠...

[更多关于cnvpe的问题>>](#)

百度知道

[易语言程序分析笔记 - 知乎](#)

可执行文件数据转换支持



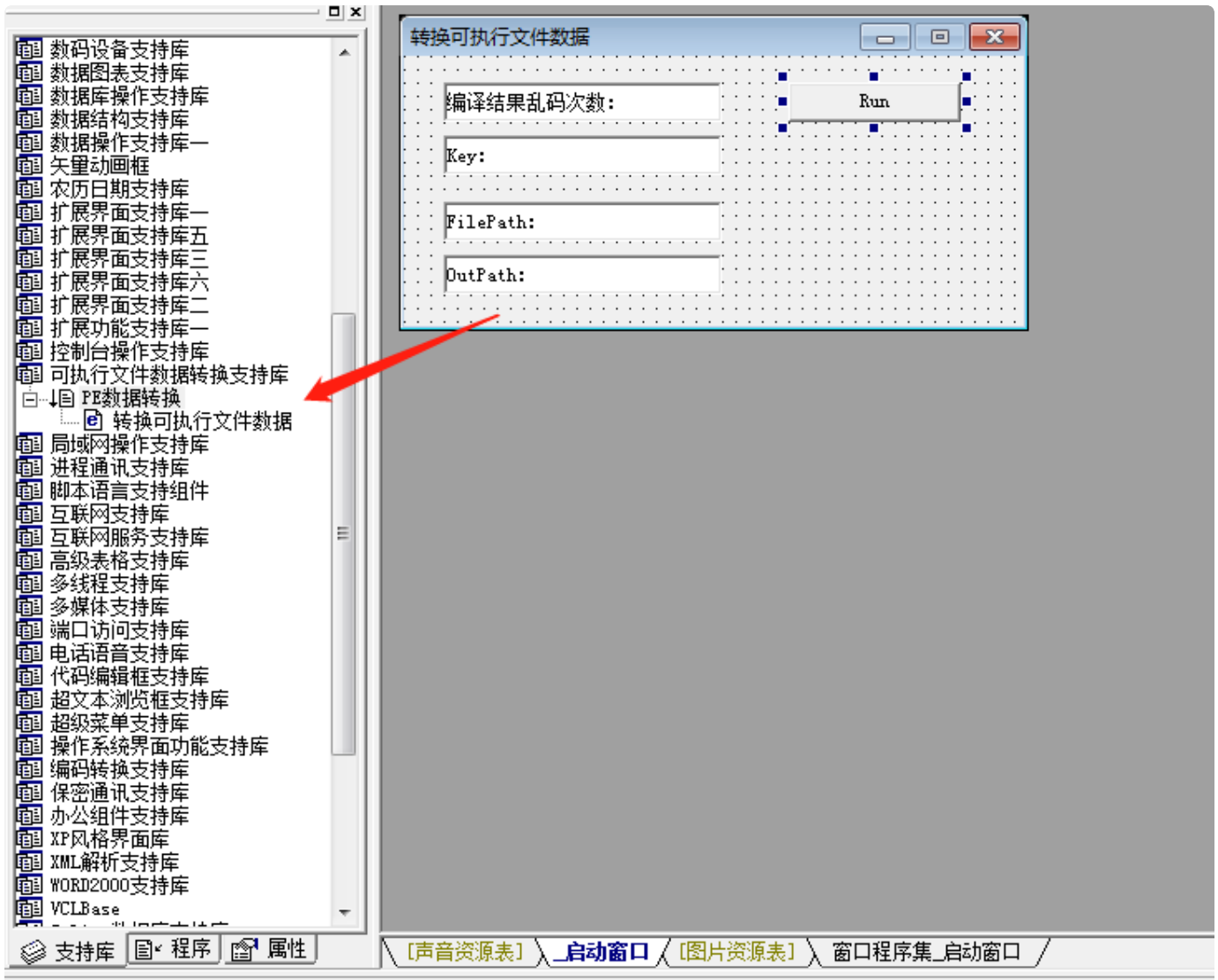
2022年9月30日 0x1C:延迟 0x24:申请内存 [可执行文件数据转换支持](#)
库——cnvpe 0x0:转换可执行文件数据 数据操作支持库——dp1 0x0:压缩
数据 0x4:解压数据 0x8:取数据摘要 0xC:加密数据 0x10:解密数...

知乎

代码编写

调用接口就行了

程序图如下



代码如下

窗口程序集名	保留	保留	备注
窗口程序集_启动窗口			

子程序名	返回值类型	公开	易包	备注
_启动窗口_创建完毕				

子程序名	返回值类型	公开	易包	备注
_按钮1_被单击				

变量名	类型	静态	数组	备注
result	整数型			
mBin	字节集			
i	整数型			

```

mBin = 读入文件 (编辑框3.内容)

--> 变量循环首 (1, 到整数 (编辑框1.内容), 1, i)
    mBin = 转换可执行文件数据 (mBin, 到整数 (编辑框2.内容), )
-- 变量循环尾 0
-> 写到文件 (编辑框4.内容, mBin)

```

看看其功能介绍吧，大概就是早期易语言程序误杀严重，然后作者提供了这个函数给大家使用。

- 1 调用格式： (字节集) 转换可执行文件数据 (字节集 可执行文件数据, [整数型 转换码], [整数型变量 转换结果]) - 可执行文件数据转换支持库->PE数据转换
- 2 英文名称: cnvpe
- 3 转换所提供的使用易语言编译的可执行文件或易语言支持库内容数据, 使其内容完全被改变, 但不影响其正常使用。返回转换后的结果字节集。如果转换失败, 将返回空字节集, 同时如果参数中提供了转换结果获取变量, 其中将返回具体的错误代码值。本命令主要用作应付一些使用落后的特征码杀毒技术的杀毒软件, 使其不再或很难对易语言编译出来的程序和易语言本身支持库误报, 建议大家在发布自己的软件前使用本命令转换下编译后的可执行文件和所需要携带的支持库。注意: 1、多次重复转换可能会导致转换后的可执行文件或易语言支持库无法使用, 请确保只转换一次; 2、对于非易语言编译的可执行文件或DLL, 不保证转换后能够正常使用。本命令为初级命令。
- 4 参数<1>的名称为“可执行文件数据”, 类型为“字节集 (bin) ”。提供需要转换的PE格式可执行文件数据, 可以是EXE或DLL文件内容。
- 5 参数<2>的名称为“转换码”, 类型为“整数型 (int) ”, 可以被省略。可以是非零的任何数值, 使用不同的数值将得到完全不同的转换结果, 使用相同的数值将得到完全相同的转换结果。如果提供数值0, 将自动使用随机数值。如果本参数被省略, 默认值为0。
- 6 参数<3>的名称为“转换结果”, 类型为“整数型 (int) ”, 可以被省略, 提供参数数据时只能提供变量。通过为本参数提供一个变量即可获取命令运行后具体的转换是否成功状态, 所返回值有以下几种: 0:转换成功; -1:所提供数据无效; -2:数据中没有需要转换的部分; -3:无法找到转换代码插入空间。如果本参数被省略, 将不返回转换结果。
- 7
- 8 操作系统需求: Windows

小结

编译结果打乱码的意思就是: 加壳

下次再发易语言 Diy , 就把这个 cnvpe 给改了, 直接加 VMP、SafeEngine、TMD。

结束了

还算不错,

当我不在你会不会难过,

你够不够我这样洒脱,

说不上爱别说谎~

2023年7月11日02:43:56