

【reversing.kr逆向之旅】Ransomware的writeup

原创

iqiqiya 于 2018-11-15 21:01:16 发布 372 收藏

分类专栏: [我的逆向之路](#) [我的CTF之路](#) [-----reversing.kr](#) 文章标签: [【reversing.kr逆向之旅】Ransomware的w Ransomware writeup](#) [reversing.k](#) [Reverse](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/xiangshangbashaonian/article/details/84110506>

版权



[我的逆向之路](#) 同时被 3 个专栏收录

108 篇文章 10 订阅

订阅专栏



[我的CTF之路](#)

92 篇文章 5 订阅

订阅专栏

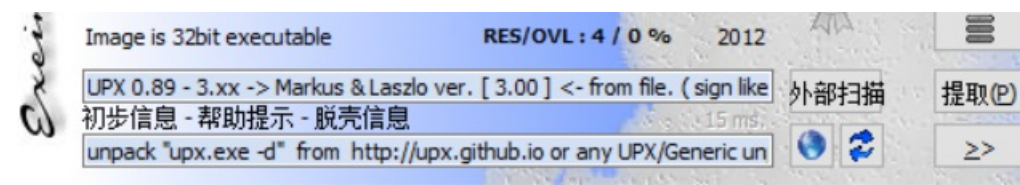


[-----reversing.kr](#)

11 篇文章 0 订阅

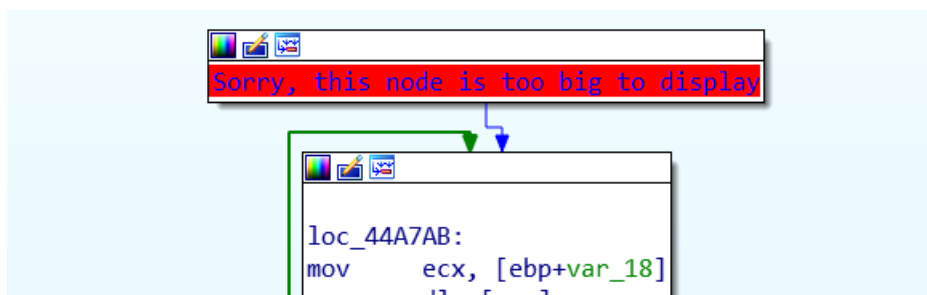
订阅专栏

Exeinfope查到有UPX壳



先使用脱壳机进行脱壳

脱壳后载入IDA 发现直接显示太大无法展示



空格转为文本视图 可以很明显知道 下面红框中的就是一段段花指令

```

.text:004135E0      push    ebp
.text:004135E1      mov     ebp, esp
.text:004135E3      sub     esp, 24h
.text:004135E6      push    ebx
.text:004135E7      push    esi
.text:004135E8      push    edi
.text:004135E9      pusha
.text:004135EA      popa
.text:004135EB      nop
.text:004135EC      push    eax
.text:004135ED      pop     eax
.text:004135EE      push    ebx
.text:004135EF      pop     ebx
.text:004135F0      pusha
.text:004135F1      popa
.text:004135F2      nop
.text:004135F3      push    eax
.text:004135F4      pop     eax
.text:004135F5      push    ebx
.text:004135F6      pop     ebx
.text:004135F7      pusha
.text:004135F8      popa
.text:004135F9      nop
.text:004135FA      push    eax
.text:004135FB      pop     eax
.text:004135FC      push    ebx
.text:004135FD      pop     ebx
.text:004135FE      pusha
.text:004135FF      popa
.text:00413600      nop
.text:00413601      push    eax
.text:00413602      pop     eax
.text:00413603      push    ebx
.text:00413604      pop     ebx

```

<https://blog.csdn.net/xiangshangbashaonian>

查看最后结束的位置 就在0x0044A775

```

.text:0044A770      nop
.text:0044A771      push    eax
.text:0044A772      pop     eax
.text:0044A773      push    ebx
.text:0044A774      pop     ebx
.text:0044A775      push    offset aKey ; "Key : "
.text:0044A77A      call    ds:printf
.text:0044A780      add     esp, 4
.text:0044A783      call    sub_401000

```

直接IDC脚本将他们都NOP掉

```

auto i,start = 0x004135E9,end = 0x0044A775;
for(i=start;i<end;i++){
PatchByte(i,0x90);}
Message("\n" + "OK\n");

```

接着可以发现函数sub_401000也有一段花指令

依然是找到起始和末尾地址 IDC脚本nop掉

nop之后可以发现它就是一个空函数

Function name	Segment
sub_401000	.text
main	.text
StartAddress	.text
__security_check_cookie(x)	.text
__tmainCRTStartup	.text
start	.text
__report_gsfailure	.text
__CxxUnhandledExceptionFilter(_EXCE...	.text
_amsg_exit	.text
__onexit	.text
_atexit	.text
sub_44AEBA	.text
_XcptFilter	.text
__ValidateImageBase	.text
__FindPESection	.text
__IsNonwritableInCurrentImage	.text
_initterm	.text
_initterm_e	.text
__SEH_prolog4	.text
__SEH_epilog4	.text
__except_handler4	.text
__setdefaultprecision	.text
sub_44B116	.text
__security_init_cookie	.text
_crt_debugger_hook	.text
terminate(void)	.text

.text:004135BB	nop	
.text:004135BC	push	eax
.text:004135BD	pop	eax
.text:004135BE	push	ebx
.text:004135BF	pop	ebx
.text:004135C0	pusha	
.text:004135C1	popa	
.text:004135C2	nop	
.text:004135C3	push	eax
.text:004135C4	pop	eax
.text:004135C5	push	ebx
.text:004135C6	pop	ebx
.text:004135C7	pusha	
.text:004135C8	popa	
.text:004135C9	nop	
.text:004135CA	push	eax
.text:004135CB	pop	eax
.text:004135CC	push	ebx
.text:004135CD	pop	ebx
.text:004135CE	pop	edi
.text:004135CF	pop	esi
.text:004135D0	pop	ebx
.text:004135D1	pop	ebp
.text:004135D2	ret	
.text:004135D2	sub_401000	endp
.text:004135D2		

```

auto i,start = 0x00401006,end = 0x004135CE;
for(i=start;i<end;i++){
PatchByte(i,0x90);}
Message("\n" + "OK\n");

```

之后改下main()的起始地址 改到nop指令结束的位置 再改下使堆栈平衡就可以F5了

sub_44A990	.text
sub_401000	.text
start	.text
unlock	.text
main	.text
lock	.text
_invoke_watson	.text
_initterm_e	.text
_initterm	.text
_except_handler4_common	.text
_crt_debugger_hook	.text
_controlfp_s	.text
_atexit	.text
_amsg_exit	.text
__setdefaultprecision	.text
__security_check_cookie(x)	.text
__onexit	.text
__except_handler4	.text
__dllonexit	.text
__tmainCRTStartup	.text
__security_init_cookie	.text
__report_gsfailure	.text
__ValidateImageBase	.text
__SEH_prolog4	.text
__SEH_epilog4	.text
__IsNonwritableInCurrentImage	.text
__FindPESection	.text
__CxxUnhandledExceptionFilter(_EXCE...	.text
_XcptFilter	.text

.text:004135D2	-----
.text:004135D3	align 10h
.text:004135E0	-----
.text:004135E0	----- SUBROUTINE -----
.text:004135E0	-----
.text:004135E0	Attribute
.text:004135E0	
.text:004135E0	
.text:004135E0	int __cc
.text:004135E0	_main
.text:004135E0	
.text:004135E0	var_24
.text:004135E0	var_1D
.text:004135E0	var_1C
.text:004135E0	var_18
.text:004135E0	var_14
.text:004135E0	var_10
.text:004135E0	var_C
.text:004135E0	var_8
.text:004135E0	File
.text:004135E0	argc
.text:004135E0	argv
.text:004135E0	envp
.text:004135E0	
.text:004135E1	
.text:004135E3	
.text:004135E6	
.text:004135E7	
.text:004135E7	mov ebp, esp
.text:004135E7	sub esp, 24h
.text:004135E7	push ebx
.text:004135E7	push esi

Edit function

Name of function: **main**

Start address: **.text:0044A775**

End address: **.text:0044A98A**

Color: DEFAULT

Enter size of (in bytes):

Local variables area: 0x30

Saved registers: 0x4

Purged bytes: 0x0

Frame pointer delta: 0x0

Does not return

Far function

Library func

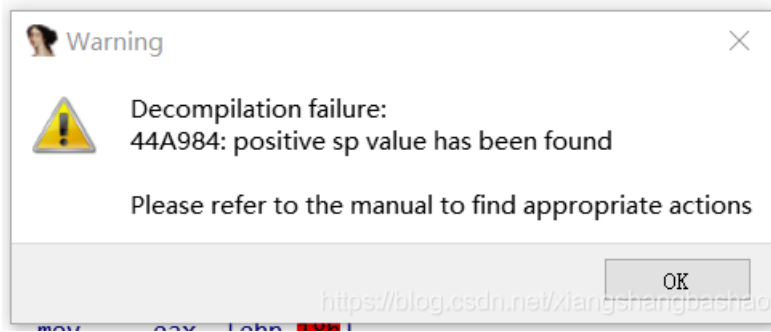
Static func

BP based frame

BP equals to SP

OK Cancel Help

F5的时候报错



提示哪里有问题 就修改它上边的数值 直到他们都是整数 就可以F5了

可以参考<https://blog.csdn.net/xiangshangbashaonian/article/details/81950110>

```
.text:0044A95A 008      call     ds:putc
.text:0044A960 008      add     esp, 8
.text:0044A963 000      call     sub_401000
.text:0044A968 000      jmp     short loc_44A93A
.text:0044A96A      ; -----
.text:0044A96A      loc_44A96A:      ; CODE XREF
.text:0044A96A 000      push    offset aEA      ; "\n颇老阑
.text:0044A96F 004      call     ds:printf
.text:0044A975 004      add     esp, 4
.text:0044A978 000      call     sub_401000
.text:0044A97D 000      call     ds:_getch
.text:0044A983 001      pop     edi
.text:0044A984 003      pop     esi
.text:0044A985 00A      pop     ebx
.text:0044A986 006      mov     esp, ebp
.text:0044A988 02A      pop     ebp
.text:0044A989 026      retn
.text:0044A989      main
```

既然堆栈已经平衡 那我们F5看下伪代码

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
    unsigned int len_key; // kr00_4
    FILE *v4; // ST2C_4
    FILE *v6; // [esp+1Ch] [ebp-14h]
    unsigned int file_len; // [esp+20h] [ebp-10h]
    int v8; // [esp+28h] [ebp-8h]
    unsigned int i; // [esp+28h] [ebp-8h]
    unsigned int j; // [esp+28h] [ebp-8h]
    FILE *file; // [esp+2Ch] [ebp-4h]

    printf("Key : ");
    sub_401000();
    scanf("%s", key);
    len_key = strlen(key);
    sub_401000();
    v4 = fopen("file", "rb");
    sub_401000();
    if ( !v4 )
    {
        sub_401000();
        printf("\n\n\n颇老阑 茫阑荐 绝促!\n");
        sub_401000();
        exit(0);
    }
}
```

```

    }
    fseek(0, 0, 2);
    sub_401000();
    file_len = ftell(file); // 函数 ftell() 用于得到文件位置指针当前位置相对于文件首的偏移字节
    sub_401000();
    rewind(file); // 是检测流上的文件结束符，如果文件结束，则返回非0值，否则返回0
    sub_401000();
    while ( !feof(file) )
    {
        sub_401000();
        file_data[v8] = fgetc(file); // 从文件指针stream指向的文件中读取一个字符，读取一个字节后，光标位
        sub_401000();
        ++v8;
        sub_401000();
    }
    sub_401000();
    for ( i = 0; i < file_len; ++i ) // 这里是关键
    {
        file_data[i] ^= key[i % len_key]; // 先将file的每个字节与key的每个字节异或
        sub_401000();
        file_data[i] = ~file_data[i]; // ~表示按位取反 将得到的数组再做取反操作
        sub_401000();
    }
    fclose(file);
    sub_401000();
    v6 = fopen("file", "wb");
    sub_401000();
    sub_401000();
    for ( j = 0; j < file_len; ++j )
    {
        fputc(file_data[j], v6);
        sub_401000();
    }
    printf(
        "\n"
        "颇老阑 汗备沁促!\n"
        "唱绰 各矫 唱悔瘤父 距加篮 瘤虐绰 鞞唱捞促!\n"
        "蝶扼辑 呈啊 唱俊霸 捣阑 玲绊, 棵官弗 虐蔼阑 罐疽促摺 颇老篮 沥惑拳 登绢 乐阑 巴捞促!\n"
        "窍瘤父 父距 肋给等 虐甫 持菌促摺 唱绰 酒林酒林 唱悔扁 锭巩俊 呈狼 颇老篮 肚 噶啊龙 巴捞促!");
    sub_401000();
    return getch();
}

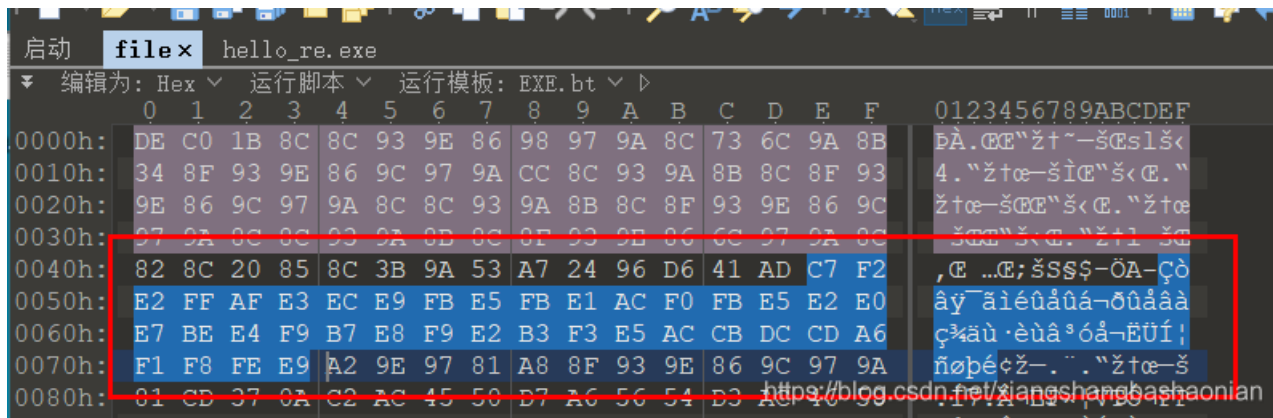
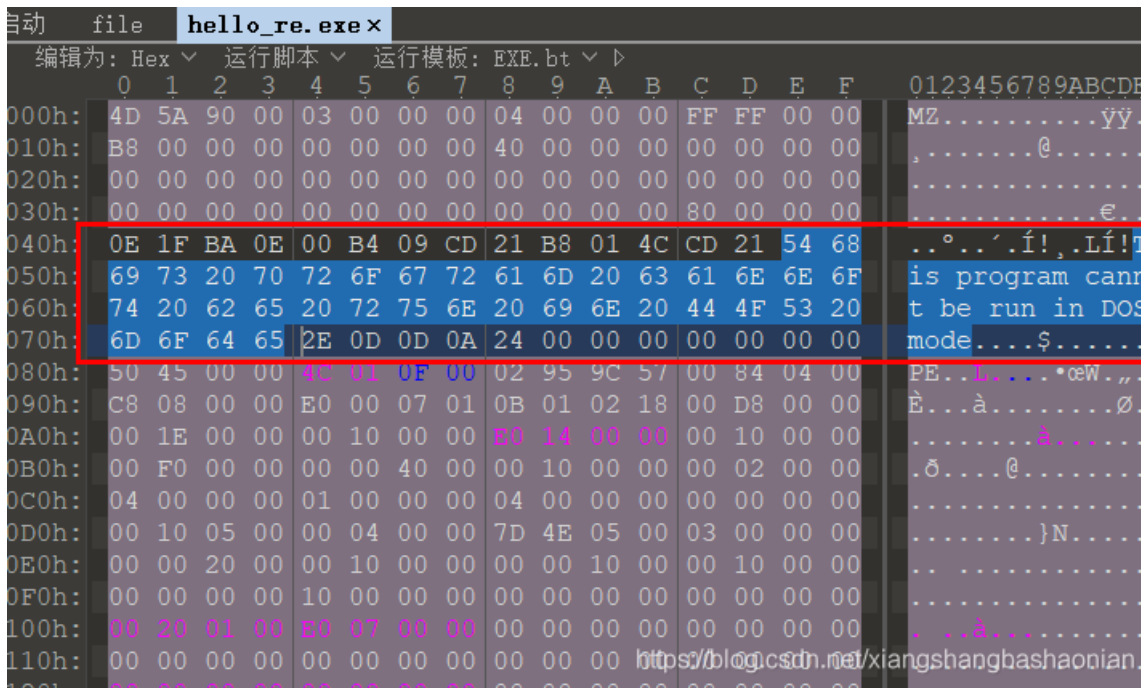
```

可以看到 多处调用了sub_401000这个空函数 那我们分析时直接忽略就好(怪不得加载那么卡)

分析过程已经写在里边了

既然我们已经知道提供的file是一个加密后的.exe文件 那我们可以先找到加密所用的Key

通过和一个正常的.exe文件进行对比 可以发现PE结构中的不同



key值可以用python脚本来得到

```
#加密后的
cipher_data = [
    0xC7, 0xF2, 0xE2, 0xFF, 0xAF, 0xE3, 0xEC, 0xE9, 0xFB, 0xE5, 0xFB, 0xE1,
    0xAC, 0xF0, 0xFB, 0xE5, 0xE2, 0xE0, 0xE7, 0xBE, 0xE4, 0xF9, 0xB7, 0xE8,
    0xF9, 0xE2, 0xB3, 0xF3, 0xE5, 0xAC, 0xCB, 0xDC, 0xCD, 0xA6, 0xF1, 0xF8,
    0xFE, 0xE9
]
#原始的
plain_data = [
    0x54, 0x68, 0x69, 0x73, 0x20, 0x70, 0x72, 0x6F, 0x67, 0x72, 0x61, 0x6D,
    0x20, 0x63, 0x61, 0x6E, 0x6E, 0x6F, 0x74, 0x20, 0x62, 0x65, 0x20, 0x72,
    0x75, 0x6E, 0x20, 0x69, 0x6E, 0x20, 0x44, 0x4F, 0x53, 0x20, 0x6D, 0x6F,
    0x64, 0x65
]

key = ''
#与0xff异或可以用来取反
for i in xrange(len(cipher_data)):
    key += chr(cipher_data[i] ^ 0xFF ^ plain_data[i])

print key
#letsplaychessletsplaychessletsplayches
```

因为数据比较多 所以多个key连在一起

可以很明显知道Key就是**letsplaychess**

那么我们既然已经拿到了key值 就可以解密啦

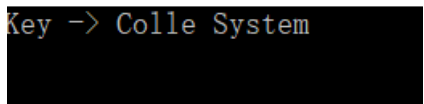
```
#加密后的
enc_data = open('file', 'rb').read()
#原始的
origin = open('origin.exe', 'wb')
#我们的key
key = 'letsplaychess'

for i in xrange(len(enc_data)):
    origin.write(chr((ord(enc_data[i]) ^ 0xFF) ^ ord(key[i % len(key)])))

origin.close()
```

解密后得到origin.exe

运行即可(win10可能会提示缺少.dll文件 下载好放在同级目录即可)

A terminal window with a black background and white text. The text reads "Key -> Colle System".

Key -> Colle System

参考链接:

<http://www.mottoin.com/article/reverse/88447.html>