

# Writeup of Ransomeware(reverse) in reversing.kr

原创

C0ss4ck 于 2018-01-22 23:43:46 发布 682 收藏

分类专栏: [Reverse of CTF](#) 文章标签: [CTF reverse wp](#)

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/cossack9989/article/details/79134684>

版权



[Reverse of CTF 专栏收录该内容](#)

7 篇文章 0 订阅

订阅专栏

emmmmm这道题就当复习了一下常规操作吧

首先下载附件, 得到 `readme.txt`, `file`, `run.exe`

打开 `readme` 理解一下题目意思, 大概就是说这个 `run.exe` 加密了某一个文件, 得到了 `file`, 现在需要将源文件逆向出来。

## 0x00 壳

DIE查壳, 发现upx壳, 脱掉

## 0x01花指令

这个程序的反汇编时间偏长, 一定有蹊跷。

果然, 一大片花指令, 而且是在 `main` 函数的开头部分以及一个无意义函数的全部

```
IDA - C:\Users\76923\Desktop\run.exe
File Edit Jump Search View Debugger Options Windows Help
Functions window  Occurrences of: call ds:printf  Hex View-1  Structures  Enums  Imports  Exports
Function name  Data  Regular function  Unexplored  Instruction  External symbol
sub_401000
main
_StartAddress
security_check_cookie(x)
_tmainCRTStartup
_start
_report_gsfailure
_CxxUnhandledExceptionFilter__EXCEPTION_PO
_amsg_exit
_onexit
_atexit
sub_44AEBA
_XcptFilter
_ValidatImageBase
FindPESection
NonwritableInCurrentImage
_pInterim
_interim_c
_SEH prolog4
_SEH epilog4
_except_handler4
_setdefaultprecision
sub_44B116
_security_init_cookie
crt.debugger_hook
terminate(void)
_unlock
_dllonexit
_lock
_except_handler4_common
_invoke_watson
_controllfp_s

00438100:    push  eax
00438101:    push  eax
00438102:    pop   eax
00438103:    push  ebx
00438104:    pop   ebx
00438105:    pusha ebx
00438106:    popa
00438107:    nop
00438108:    push  eax
00438109:    pop   eax
0043810A:    push  ebx
0043810B:    pop   ebx
0043810C:    pusha ebx
0043810D:    popa
0043810E:    nop
0043810F:    push  eax
00438110:    pop   eax
00438111:    push  ebx
00438112:    pop   ebx
00438113:    pusha ebx
00438114:    popa
00438115:    nop
00438116:    push  eax
00438117:    pop   eax
00438118:    push  ebx
00438119:    pop   ebx
0043811A:    pusha ebx
0043811B:    popa
0043811C:    nop
0043811D:    push  eax
0043811E:    pop   eax
0043811F:    push  ebx
00438120:    pop   ebx
00438121:    pusha ebx
00438122:    popa
00438123:    nop
00438124:    push  eax
00438125:    pop   eax
00438126:    push  ebx
00438127:    pop   ebx
00438128:    pusha ebx
00438129:    popa
0043812A:    nop
0043812B:    push  eax
0043812C:    popa

0043811F: _main+24B3F (Synchronized with Hex View-1)
```

但是请仔细观察, 这里的汇编代码很有意思。

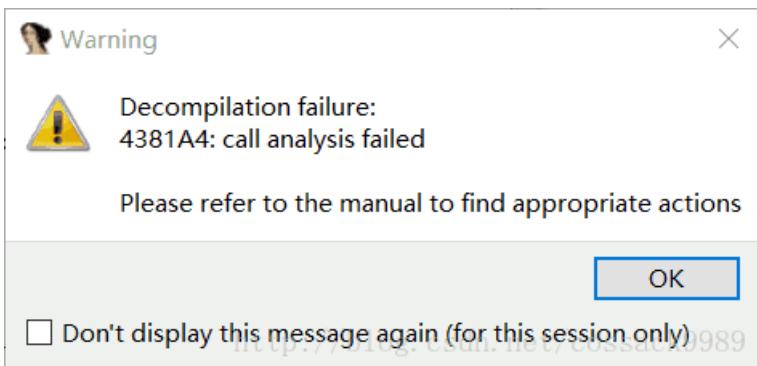
```
pusha  
popa  
nop  
push eax  
pop eax  
push ebx  
pop ebx
```

相当于没有进行任何操作，不如全部patch成nop

shift+F2祭出IDC脚本

```
static main()  
{  
    auto i,j,from,size;  
    from=0x4135E9;  
    size=0xA7EB;  
    for(i=0;i<size;i++)  
    {  
        if((Byte(from)==0x60)  
        &&(Byte(from+1)==0x61)  
        &&(Byte(from+2)==0x90)  
        &&(Byte(from+3)==0x50)  
        &&(Byte(from+4)==0x58)  
        &&(Byte(from+5)==0x53)  
        &&(Byte(from+6)==0x5B))  
        {  
            for(j=0;j<7;j++)  
            {  
                PatchByte(from,0x90);  
                from++;  
            }  
            continue;  
        }  
        from++;  
    }  
    Message("\n"+OJBK\n");  
}
```

批量Patch之后一共修改了4万多行代码，然后进行反编译，发现失败。



这就很尴尬了，然后找到0x4381A4，发现是一个没什么作用的printf，直接nop掉。

然后再次反编译，成功。

## 0x02 伪代码

观察反编译得到的代码。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    unsigned int len; // kr00_4@1
    FILE *fp; // [sp+1Ch] [bp-14h]@10
    unsigned __int32 filelen; // [sp+20h] [bp-10h]@4
    int k; // [sp+28h] [bp-8h]@1
    unsigned int i; // [sp+28h] [bp-8h]@7
    unsigned __int32 j; // [sp+28h] [bp-8h]@10
    FILE *File; // [sp+2Ch] [bp-4h]@1

    *(_BYTE *)(*(_DWORD *)(__readfsdword(24) + 48) + 2) = 68;
    CreateThread(0, 0, StartAddress, 0, 0, 0);
    printf("Key : ");
    nop();
    scanf("%s", key);
    len = strlen(key);
    nop();
    k = 0;
    File = fopen("file", "rb");
    nop();
    if ( !File )
    {
        nop();
        printf("\n\n\n颇老阑 茫阑荐 绝促!\n");
        nop();
        exit(0);
    }
    fseek(File, 0, 2);
    nop();
    filelen = ftell(File);
    nop();
    rewind(File);
    nop();
    while ( !feof(File) ) // read file
    {
        nop();
        s1[k] = fgetc(File);
        nop();
        ++k;
        nop();
    }
    nop();
    for ( i = 0; i < filelen; ++i ) // encrypt
    {
        s1[i] ^= key[i % len];
        nop();
        s1[i] = ~s1[i];
        nop();
    }
    fclose(File);
    nop();
    fp = fopen("file", "wb");
    nop();
    nop();
    for ( j = 0; j < filelen; ++j ) // store encrypted data
```

```
{  
    fputc(s1[j], fp);  
    nop();  
}  
printf("\n颇老阑 汗备沁促!\n唱绰 各矫 唱悔瘤父 距加篮 瘤虐绰 莢唱捞促!\n蝶扼辑 呈啊 唱俊霸 捣阑 玲绊， 棵官弗 虐蔼阑  
nop();  
return getch();  
}
```

(无视那个nop函数和printf的乱七八糟的内容就好)

不难发现，加密过程是流异或与取反，问题的关键在于key。

### 0x03 解密

不管那么多，先取反试试。

祭出python小脚本

```
f1=open('file','rb')  
f2=open('mid','wb')  
s=f1.read()  
for i in s:  
    i=~ord(i)  
    f2.write(chr(i%256))  
f1.close()  
f2.close()
```

然后用16进制编辑器打开mid文件，发现字符串'letsplaychess'重复了很多次，于是猜测key就是这个字符串。

下一步脚本

```
key='letsplaychess'  
f3=open('mid','rb')  
f4=open('test','wb')  
s1=f3.read()  
for i in range(len(s1)):  
    k=ord(s1[i])^ord(key[i%len(key)])  
    f4.write(chr(k))  
f3.close()  
f4.close()
```

再次使用16进制编辑器打开test文件，发现是一个upx加壳的.exe文件。再次脱壳，扔到IDA里面，发现

```
push    offset Format    ; "Key -> Colle System"  
call    ds:printf
```

嗯，Colle System就是我们需要提交的字符串。