

171220 逆向-i春秋【迎圣诞】-NoExec

原创

奈沙夜影 于 2017-12-20 23:42:12 发布 689 收藏

分类专栏: [CTF](#)

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/whklhjh/article/details/78858892>

版权



[CTF 专栏收录该内容](#)

163 篇文章 4 订阅

订阅专栏

1625-5 王子昂 总结《2017年12月20日》【连续第446天总结】

A. i春秋【迎圣诞】-NoExec

B.

后缀名写着exe, 却不能运行~

用010Editor打开分析一下

发现NT头有明显的问题

```
0100h: 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | .....  
0110h: 50 45 48 41 | 48 41 05 00 | 9A 68 CC 59 | 00 00 00 00 | PEHAHA.t.ShIY.lhjh
```

MagicNumber应该是50 45 00 00, 被改为了HA

后面一个HA很明显也是后加的, 查询一下发现这里是Machine, 表示使用的处理器

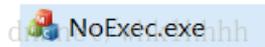
正确的值应该为4C 01, 即i386 - 332

修改以后发现还是不能运行, 继续检查

想起来DOS头里有一个重要的值: e_lfanew, 它表示了NE头的偏移

原值为00 01 00 00, 很明显是错的, 改正为10 01 00 00 (0x110)

改完发现Windows都能识别出来它是个MFC了



然而双击以后响应了一会儿还是没东西出来

拖入OD也在加载器中就跳入SEH了，估计加载的时候还有什么错误，当时没有发现

但是知道它是MFC就有地方可以操作了：

无壳，拖入IDA，发现WinMain函数里没有好辨认的东西，查找字符串也没有可读的部分\ (_) /

于是开始考虑从资源下手

用ResourceHacker查看控件ID，发现确认按钮的ID为1001

于是根据ID定位按钮函数，通过IDA的搜索立即数

(结构体自己导入)

```
.rdata:00583EE8 stru_583EE8      AFX_MSGMAP_ENTRY <0Fh, 0, 0, 0, 13h, offset sub_402B10>
.rdata:00583EE8                                     ; DATA XREF: .rdata:00583EE0↑o
.rdata:00583EE8      AFX_MSGMAP_ENTRY <37h, 0, 0, 0, 29h, offset sub_402C00>
.rdata:00583EE8      AFX_MSGMAP_ENTRY <111h, 0, 1, 1, 3Ah, offset sub_402C10>
.rdata:00583EE8      AFX_MSGMAP_ENTRY <111h, 0, 2, 2, 3Ah, offset sub_402C20>
.rdata:00583EE8      AFX_MSGMAP_ENTRY <111h, 0, 3E9h, 3E9h, 3Ah, offset sub_402C30>
```

这样就找到了函数：sub_402C30

结构很明显

```
16  if ( *(_DWORD *)(input - 12) == 37 )           // len = 37
17  {
18      if ( check(&input) == (char *)1 )
19          AfxMessageBox((int)"OK", 0, 0);
20      else
21          AfxMessageBox((int)"NO", 0, 0);
22  }
```

查看check函数，发现先是通过下标的奇偶将字符串切成两个

```
127  i = 0;
128  LOBYTE(v79) = 7;
129  len = *(_DWORD *)(*input - 12);
130  if ( len > 0 )
131  {
132      do
133      {
134          v66 = (volatile signed __int32 *)(i + 1);
135          if ( (i + 1) % 2 )
136              // 偶数
137              if ( i < 0 )
138                  goto LABEL_35;
139              if ( i > len )
140                  goto LABEL_35;
141          v69 = *(_BYTE *)(*input + i);
142          v68 = *((_DWORD *)even - 3);
143          v18 = v68 + 1;
144          v70 = v68 + 1;
145          if ( v68 + 1 < 0 )
146              goto LABEL_35;
147          if ( ((1 - *((_DWORD *)even - 1)) | ((*(_DWORD *)even - 2) - v18)) < 0 )
148              {
149                  sub_D42750(v18);
150                  even = v77;
151                  v18 = v70;
152              }
153          even[v68] = v69;
154          if ( v18 > *((_DWORD *)even - 2) )
155              goto LABEL_35;
156          *((_DWORD *)even - 3) = v18;
157          even[v18] = 0;
158      }
159      else
160      {
161          if ( i < 0 )
```

```
162 |         goto LABEL_35;
163 |         if ( i > len )
```

<http://blog.csdn.net/whklhxxx>

然后分别将两个字符串base64，再连接起来

```
197 |         v19 = v19,
198 |     }
199 |     v20 = base64(v19, even, &v70); // 偶数转base64放入str中
200 |     snprintf((int)&Str, "%s", v20);
201 |     strcat(&v76, Str, *((_DWORD *)Str - 3)); // 连接到字符串v76中
202 |     v21 = *((_DWORD *)odd - 3);
203 |     v70 = v21;
204 |     if ( v21 < 0 )
205 |         sub_D42800(0x80070057);
206 |     if ( ((1 - *((_DWORD *)odd - 1)) | (*((_DWORD *)odd - 2) - v21)) < 0 )
207 |     {
208 |         sub_D42750(v21);
209 |         odd = v74;
210 |         v21 = v70;
211 |     }
212 |     v22 = base64(v21, odd, &v70);
213 |     snprintf((int)&Str, "%s", v22); // 奇数转base64放入str中
214 |     strcat(&v76, Str, *((_DWORD *)Str - 3));
215 |     v23 = v76;
216 |     if ( *((_DWORD *)v76 - 12) % 8 ) // 在字符串后补上\xCC, 使得长度为8的倍数
217 |     {
218 |         snprintf((int)&v71, "%c", 204);
219 |         v24 = *((_DWORD *)v23 - 12) % 8;
220 |         v70 = 8 - v24;
221 |         if ( 8 - v24 > 0 )
222 |         {
223 |             v25 = v71;
224 |             v26 = 8 - v24;
225 |             do
226 |             {
```

<http://blog.csdn.net/whklhxxx>

之后大段的代码太长不想看（。

反正没有对v76操作的地方

最后可以看出来是和一个字符串比较

```
9 |     if ( *((_DWORD *)Str - 3) <= 0 )
10 |     {
11 | LABEL_94:
12 |         ReturnValue = 1;
13 |     }
14 |     else
15 |     {
16 |         v50 = Str - "Gcdk0SvnNA1tsmp5FCK1FpSDfUXZbhHBSPheZaixuMyzqyysOAPCPB/p7sMpmK1KZo+1PfhMZxw="; // 要求v50=0
17 |         v65 = Str - "Gcdk0SvnNA1tsmp5FCK1FpSDfUXZbhHBSPheZaixuMyzqyysOAPCPB/p7sMpmK1KZo+1PfhMZxw=";
18 |         while ( 1 )
19 |         {
20 |             if ( i2 < 0 )
21 |                 goto LABEL_35;
22 |             v51 = *((_DWORD *)Str - 3);
23 |             if ( i2 > v51 )
24 |                 goto LABEL_35;
25 |             if ( aGcdk0svnna1tsm[i2 + v50] != aGcdk0svnna1tsm[i2] )
26 |                 break;
27 |             v50 = v65;
28 |             if ( ++i2 >= v51 )
29 |                 goto LABEL_94;
30 |         }
31 |         ReturnValue = -1;
32 |     }
33 |     LOBYTE(v70) = 6.
```

<http://blog.csdn.net/whklhxxx>

直接把它拖下来解b64发现不可见，说明中间还有操作

到这里就陷入了僵局

回头重新研究

发现当用IDA调试的时候会报start函数执行失败

这个提示让我茅塞顿开，节区属性的可执行属性被关闭了！

原来题目名“NoExec”就给出提示了，(´_´) 奈何一直没get到

于是将代码所在节区.text的属性加上可执行（DWORD Executable：1）

The image shows a screenshot of IDA Pro. The top part is a memory dump from 0220h to 0290h. The bottom part is the 'Inspector - Templatel.bt' window showing the section table. The selected entry is 'struct SECTION_FLAGS Characteristics' with 'Code Readable' set to 1.

Address	Hex	ASCII
0220h	00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 40@
0230h	2E 72 64 61 74 61 00 00 FA D1 04 00 00 40 15 00	.rdata..úÑ...@..
0240h	00 D2 04 00 00 28 15 00 00 00 00 00 00 00 00 00	.Ò... (.....
0250h	00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00@..@.data...
0260h	40 DA 00 00 00 20 1A 00 00 64 00 00 00 FA 19 00	@Ú... ..d...ú..
0270h	00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0@...À
0280h	2E 72 73 72 63 00 00 00 98 0F 16 00 00 00 1B 00	.rsrc...~.....
0290h	00 10 16 00 00 5E 1A 00 00 00 00 00 00 00 00 00^.....

Name	Value	Start	Size
> struct IMAGE_DATA_DIRECTORIES DataDirectory		188h	80h
▼ struct IMAGE_SECTION_HEADER sections_table[5]		208h	C8h
▼ struct IMAGE_SECTION_HEADER sections_table[0]	.text	208h	28h
> BYTE Name[8]	.text	208h	8h
DWORD VirtualSize	1385439	210h	4h
DWORD VirtualAddress	1000h	214h	4h
DWORD SizeOfRawData	1385472	218h	4h
DWORD PointerToRawData	400h	21Ch	4h
DWORD NonUsedPointerToRelocations	0	220h	4h
DWORD NonUsedPointerToLinenumbers	0	224h	4h
WORD NonUsedNumberOfRelocations	0	228h	2h
WORD NonUsedNumberOfLinenumbers	0	22Ah	2h
> struct SECTION_FLAGS Characteristics	Code Readable	22Ch	4h

就可以运行了！

在OD中对刚才分析的结果b64后的字符串下断，断到sub_401EC0->sub_401CE0中进行了操作

在IDA中大概看了一下，很复杂(´_´) 奈何

掏出密码学分析插件看一眼，哼，果然在这个函数里有对DES_box的调用

那这个sub_401EC0就是DES没跑了

OD中对这个函数下断，发现它是每次截取8个字符进行DES，密钥通过对参数的观察发现是

```
6E 06 15 51 93 5B 07 EA
```

IDA中大概能看到的上面一堆复杂操作出来的....看不懂（望天

跑了一下发现这个密钥是不变的

解密出来由于是二进制值，所以还有一次base64转为可见字符

于是按照思路写出解密脚本：

```

from base64 import b64encode, b64decode
from Cryptodome.Cipher import DES

k2 = [0x6E, 0x06, 0x15, 0x51, 0x93, 0x5B, 0x07, 0xEA]
key = bytes(k2)
x = DES.new(key, DES.MODE_ECB)
s = b"GcDk0SvnNA1tsmp5FCK1FpSDfUXZbhHBSPhZaiXuMyzqyysOAPCPB/p7sMpmK1KZo+1PfhMZxw="
c = b64decode(s)

# 解密
p = x.decrypt(c)

# 还原成俩字符
x = b64decode(p)
y = b64decode(p[28:-4])
for i in range(37):
    if(i%2==0):
        print(chr(x[i//2]), end='')
    else:
        print(chr(y[i//2]), end='')

```

活动页面上写着难度为中下，不过这个题目着实难了我一会儿(:3/ <)
 虽然复盘想想好像的确没有太复杂的东西，但是各种小细节还是挺麻烦的
 学到和巩固了很多知识~感谢出题人和春秋提供的题目(°▽°)/

C. 明日计划

加密与解密/看雪题目