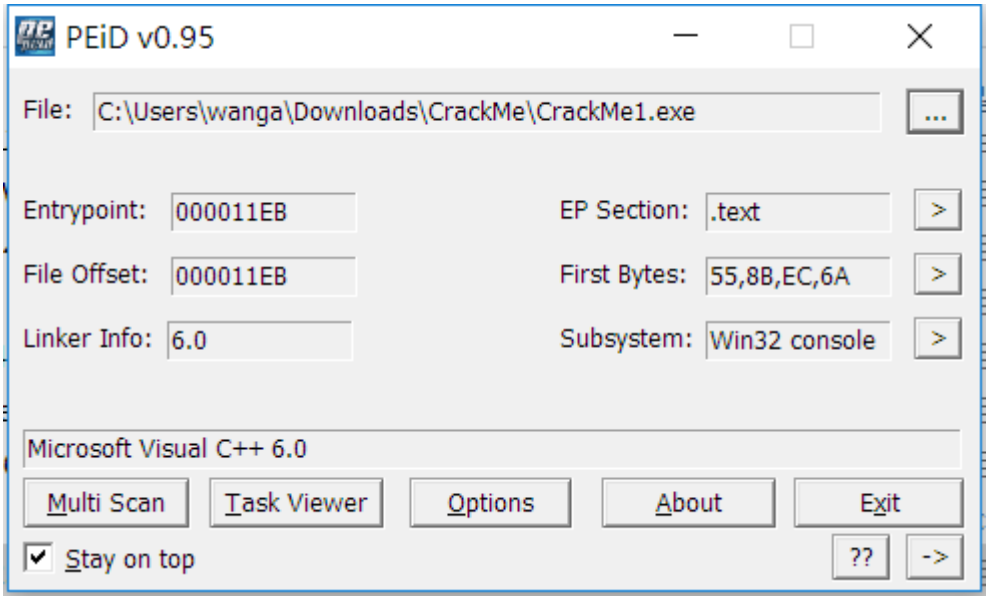
证明自己吧 WriteUp

1、首先使用 PEid 软件打开可执行文件，进行查壳：



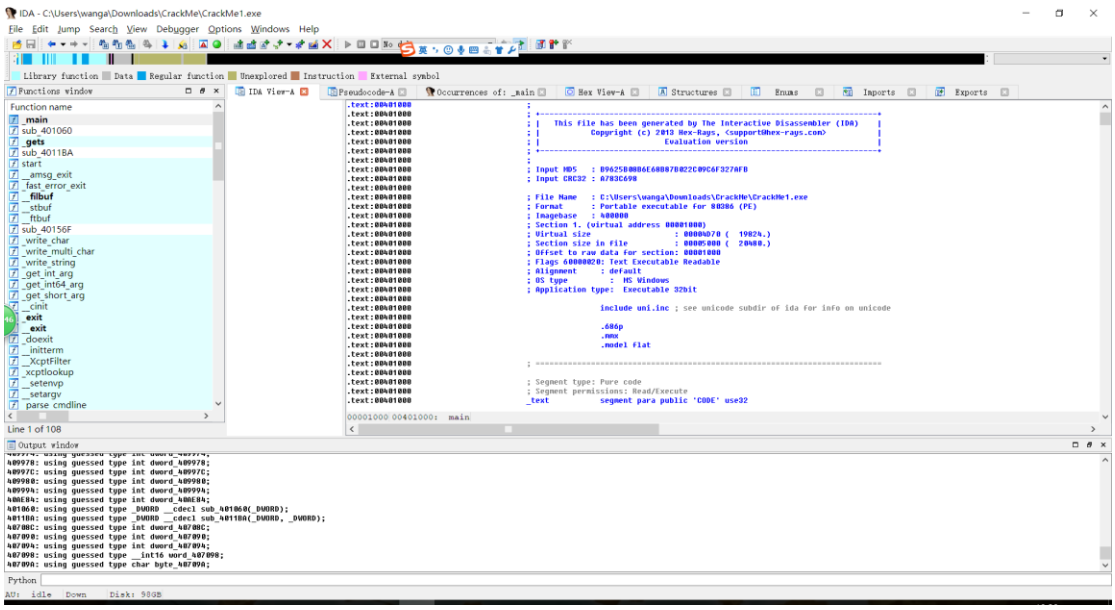
上述 not packed 说明未加壳，可以直接反编译。

PS：

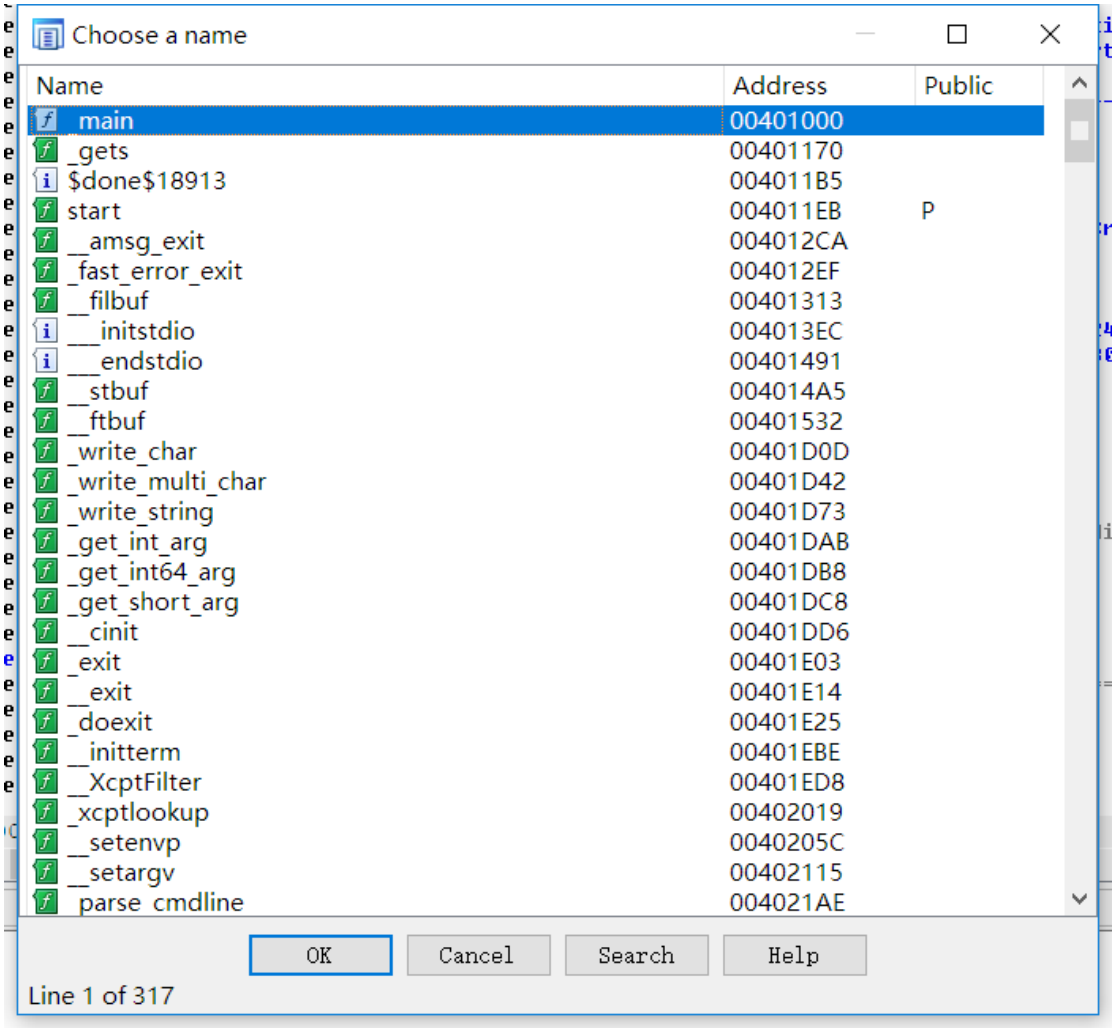
带壳程序应该是指一些加壳的应用程序。一般的病毒就是这样做免杀的。加壳的全称是可执行程序资源压缩，是保护文件的常用手段。加壳过的程序可以直接运行,但是不能查看源代码.要经过脱壳才可以查看源代码。加壳其实是利用特殊的算法，对 EXE、DLL 文件里的资源进行压缩。类似 WINZIP 的效果，只不过这个压缩之后的文件，可以独立运行，解压过程完全隐蔽，都在内存中完成。

所以之前有的程序无法反编译可能是由于已经加壳，只有解壳后才能反编译。

2、用 IDA 反编译：



按住 CTRL+L 查找相关标签：



猜测_main 为 main 函数，点击_main 标签进入相关代码：

```
.text:00401000                          _main           proc near               ; CODE XREF: start+AF↓p
.text:00401000
.text:00401000                          var_7D0         = dword ptr -7D0h
.text:00401000                          argc            = dword ptr  4
.text:00401000                          argv            = dword ptr  8
.text:00401000                          envp            = dword ptr  0Ch
.text:00401000
.text:00401000 81 EC D0 07 00 00                        sub     esp, 7D0h
.text:00401006 68 70 70 40 00                           push    offset aCanYouGuessThe ; "Can you Guess the Code: "
.text:0040100B E8 AA 01 00 00                           call    sub_4011BA
.text:00401010 8D 44 24 04                              lea     eax, [esp+7D4h+var_7D0]
.text:00401014 50                                       push    eax             ; char *
.text:00401015 E8 56 01 00 00                           call    _gets
.text:0040101A 8D 4C 24 08                              lea     ecx, [esp+7D8h+var_7D0]
.text:0040101E 51                                       push    ecx
.text:0040101F E8 3C 00 00 00                           call    sub_401060
.text:00401024 83 C4 0C                                 add     esp, 0Ch
.text:00401027 85 C0                                    test    eax, eax
.text:00401029 74 16                                    jz      short loc_401041
.text:0040102B 68 48 70 40 00                           push    offset aGoodTheKeyIsYo ; "Good! The Key is your input o(^o^)o\n"
.text:00401030 E8 85 01 00 00                           call    sub_4011BA
.text:00401035 83 C4 04                                 add     esp, 4
.text:00401038 33 C0                                    xor     eax, eax
.text:0040103A 81 C4 D0 07 00 00                        add     esp, 7D0h
.text:00401040 C3                                       retn
.text:00401041                          ; ---------------------------------------------------------------------------
.text:00401041
.text:00401041                          loc_401041:                             ; CODE XREF: _main+29↑j
.text:00401041 68 30 70 40 00                           push    offset aYouDonTGuessIt ; "You Don't Guess it~\n"
.text:00401046 E8 6F 01 00 00                           call    sub_4011BA
.text:0040104B 83 C4 04                                 add     esp, 4
.text:0040104E 33 C0                                    xor     eax, eax
00001000 00401000: _main
```

F5 生成 C 代码：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax@2
  int v4; // [sp+0h] [bp-7D0h]@1

  sub_4011BA("Can you Guess the Code: ", v4);
  gets((char *)&v4);
  if ( sub_401060((const char *)&v4) )
  {
    sub_4011BA("Good! The Key is your input o(^o^)o\n", v4);
    result = 0;
  }
  else
  {
    sub_4011BA("You Don't Guess it~\n", v4);
    result = 0;
  }
  return result;
}
```

可以看出 sub_401060 为判断 code 是否正确的函数，点击 sub_401060 进入相关代码：

```
signed int __cdecl sub_401060(const char *a1)
{
  unsigned int v1; // edx@2
  unsigned int v2; // edx@4
  unsigned int v3; // edx@6
  int v5; // [sp+Ch] [bp-10h]@1
  int v6; // [sp+10h] [bp-Ch]@1
  int v7; // [sp+14h] [bp-8h]@1
  __int16 v8; // [sp+18h] [bp-4h]@1
  char v9; // [sp+1Ah] [bp-2h]@1

  v5 = dword_40708C;
  v6 = dword_407090;
  v8 = word_407098;
  v9 = byte_40709A;
  v7 = dword_407094;
  if ( strlen(a1) == strlen((const char *)&v5) )
  {
    v1 = 0;
    if ( strlen(a1) != 0 )
    {
      do
      {
        a1[v1] ^= 0x20u;
        ++v1;
      }
      while ( v1 < strlen(a1) );
    }
    v2 = 0;
    if ( strlen((const char *)&v5) != 0 )
    {
      do
        *((_BYTE *)&v5 + v2++) -= 5;
      while ( v2 < strlen((const char *)&v5) );
    }
    v3 = 0;
    if ( strlen((const char *)&v5) == 0 )
      return 1;
    while ( *((_BYTE *)&v5 + v3 + a1 - (const char *)&v5) == *((_BYTE *)&v5 + v3) )
    {
      ++v3;
      if ( v3 >= strlen((const char *)&v5) )
        return 1;
    }
  }
  return 0;
```

第一段代码：

```
v5 = dword_40708C;
v6 = dword_407090;
v8 = word_407098;
v9 = byte_40709A;
v7 = dword_407094;
if ( strlen(a1) == strlen((const char *)&v5) )
{
  v1 = 0;
  if ( strlen(a1) != 0 )
  {
    do
    {
      a1[v1] ^= 0x20u;
      ++v1;
    }
    while ( v1 < strlen(a1) );
  }
```

看到我们传进来的字符串叫做 a1，假如 a1 的长度等于 v5 的长度，则开始逐个元素做异或运算，和 20 异或。

点击 dword_40708C 进入定义区，发现后续有一个 db 伪指令定义的 0。

```
5E 06                   word_407098     dw 65Eh                 ; |
00                      byte_40709A     db 0                    ; |
```

猜测这是一个结尾是'\0'的字符串，并可以看出字符串为：68571948 506e5878 546a1958 5e06H。

```
89 00 00 00
8C 68 57 19 48
90 50 6E 58 78
94 54 6A 19 58
98 5E 06
9A 00
9B 00
```

第二段代码：

```
v2 = 0;
if ( strlen((const char *)&v5) != 0 )
{
  do
    *((_BYTE *)&v5 + v2++) -= 5;
  while ( v2 < strlen((const char *)&v5) );
}
```

v5 中的每个字节中存储的值-5。

第三段代码：

```
v3 = 0;
if ( strlen((const char *)&v5) == 0 )
  return 1;
while ( *((_BYTE *)&v5 + v3 + a1 - (const char *)&v5) == *((_BYTE *)&v5 + v3) )
{
  ++v3;
  if ( v3 >= strlen((const char *)&v5) )
    return 1;
}
```

`*((_BYTE *)&v5 + v3 + a1 - (const char *)&v5)` 故意混淆代码，实际作用仍然为：*（v3+a1）。

此段代码是比较如果当前的 v5 字符串和 a1 相同，说明 a1 为正确值。

3、仔细分析：

a) 输入字符串，对每个字节异或 0x20u；

b) 待比较字符串每个字节值减 5；

c) 此时如果输入串和待比较字符串相同，则输入串正确。

贴出解密代码：

```
code=(0x68,0x57,0x19,0x48,0x50,0x6e,0x58,0x78,0x54,0x6a,0x19,0x58,0x5e,0x06)
for i in code:
    i=(i-5)^0x20
    print (chr(i), end='')
print()
```

```
PS D:\My CTF\实实验验吧吧\逆逆向向工工程程python.exe .\main.py
Cr4ckIsSoE4sy!
```