Buuoj-reverse-ACTF新生赛2020_usualCrypt

原创

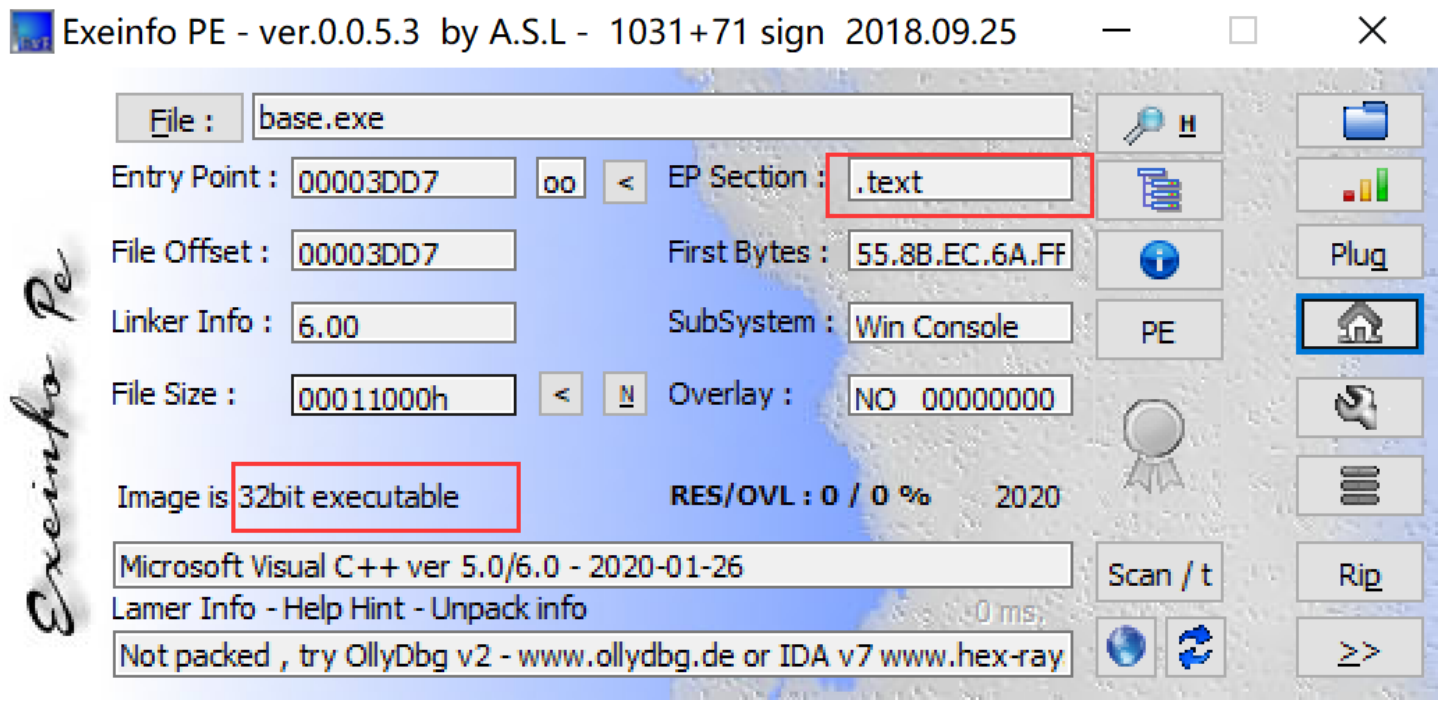
bunner_ 于 2021-02-23 22:02:23 发布 77 收藏

分类专栏： 逆向之旅

本文链接：https://blog.csdn.net/bunner_/article/details/114003786

1. 分析文件类型，是否加壳

将文件扔进exeinfope中查看，发现是无壳的32位可执行程序



2. 使用IDA对文件进行分析

- 此处的main函数代码较为简单，其逻辑为接收输入，用 `sub_401080` 函数对其进行处理，处理后的结果若与 `byte_40E0E4` 处字符串相同，则该输出即为flag。所以我们的重点应该放在 `sub_401080` 函数上。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // esi
  int result; // eax
  int output; // [esp+8h] [ebp-74h]
  int v6; // [esp+Ch] [ebp-70h]
  int v7; // [esp+10h] [ebp-6Ch]
  __int16 v8; // [esp+14h] [ebp-68h]
  char v9; // [esp+16h] [ebp-66h]
```

```
  char input; // [esp+18h] [ebp-64h]

  print(&unk_40E140);
  scanf(aS, &input);
  output = 0;
  v6 = 0;
  v7 = 0;
  v8 = 0;
  v9 = 0;
  sub_401080(&input, strlen(&input), &output);
  v3 = 0;
  while ( *((_BYTE *)&output + v3) == byte_40E0E4[v3] )
  {
    if ( ++v3 > strlen((const char *)&output) )
      goto LABEL_6;
  }
  print(aError);
LABEL_6:
  if ( v3 - 1 == strlen(byte_40E0E4) )
    result = print(aAreYouHappyYes);
  else
    result = print(aAreYouHappyNo);
  return result;
}
```

- 分析 `sub_401080` 函数的逻辑

```
  idx = 0;
  v4 = 0;
  sub_401000();                                 一个处理函数，其对base编码表进行了一个预处理
  len_3 = len % 3;
  v6 = input;
  len_3_remain = len - len % 3;
  lena = len % 3;                               看看其长度除以3后的余数
  if ( len_3_remain > 0 )
  {
    do
    {
      LOBYTE(len_3) = *(_BYTE *)(input + idx);                  大循环，每3个字符处理一次，且处理成四个字符
      idx += 3;
      v8 = v4 + 1;
      *(_BYTE *)(v8++ + output - 1) = byte_40E0A0[(len_3 >> 2) & 0x3F];
      *(_BYTE *)(v8++ + output - 1) = byte_40E0A0[16 * (*(_BYTE *)(input + idx - 3) & 3)
                                                + (((signed int)*(unsigned __int8 *)(input + idx - 2) >> 4) & 0xF)];
      *(_BYTE *)(v8 + output - 1) = byte_40E0A0[4 * (*(_BYTE *)(input + idx - 2) & 0xF)
                                              + (((signed int)*(unsigned __int8 *)(input + idx - 1) >> 6) & 3)];
      len_3 = *(_BYTE *)(input + idx - 1) & 0x3F;// input[idx-1]
      v4 = v8 + 1;
      *(_BYTE *)(v4 + output - 1) = byte_40E0A0[len_3];
    }
    while ( idx < len_3_remain );
    len_3 = lena;
  }
  if ( len_3 == 1 )
  {
    LOBYTE(len_3_remain) = *(_BYTE *)(idx + input);          余数为1时的处理方式
    v9 = v4 + 1;
    *(_BYTE *)(v9 + output - 1) = byte_40E0A0[(len_3_remain >> 2) & 0x3F];
    v10 = v9 + 1;
    *(_BYTE *)(v10 + output - 1) = byte_40E0A0[16 * (*(_BYTE *)(idx + input) & 3)];
    *(_BYTE *)(v10 + output) = 61;
LABEL_8:
    v13 = v10 + 1;
    *(_BYTE *)(v13 + output) = 61;                61为 '='
    v4 = v13 + 1;
    goto LABEL_9;
  }
```

```
  if ( len_3 == 2 )
  {
    v11 = v4 + 1;                                                    余数为2时的处理方式
    *(_BYTE *)(v11 + output - 1) = byte_40E0A0[((signed int)*(unsigned __int8 *)(idx + input) >> 2) & 0x3F];
    v12 = (_BYTE *)(idx + input + 1);
    LOBYTE(v6) = *v12;
    v10 = v11 + 1;
```

```
63    *(_BYTE *)(v10 + output - 1) = byte_40E0A0[16 * (*(_BYTE *)(idx + input) & 3) + ((v6 >> 4) & 0xF)];
64    *(_BYTE *)(v10 + output) = byte_40E0A0[4 * (*v12 & 0xF)];
65    goto LABEL_8;
66  }
67LABEL_9:
68  *(_BYTE *)(v4 + output) = 0;
69  return sub_401030((const char *)output);      // 大小写相互转化    这里是对处理后的结果进行大小写互换
70}
```

从上面两幅图的分析中可以得到，函数`sub_401080`的作用就是进行**base64编码**，**只是它还将原本的编码表做了一个简单的处理。**所以根据以上分析，我们可以写出解码代码。

解题

```python
# 大小写转化
def convert_little_big(s):
    ans = ''
    for c in s:
        if c >= 'a' and c <= 'z':
            ans += chr(ord(c) - ord('a') + ord('A'))
        elif c >= 'A' and c <= 'Z':
            ans += chr(ord(c) - ord('A') + ord('a'))
        else:
            ans += c
    return ans

# 处理base编码表
def shift(s):
    idx = 6
    s = list(s)
    for idx in range(6, 15):
        s[idx], s[idx + 10] = s[idx + 10], s[idx]
    return ''.join(s)

'''
base64解码
因为编码后无 '='
所以可以知道原字符串的长度必然是3的倍数
所以我们这里不需要去处理余数的情况
'''
def solve(s, base):
    flag = [0] * (len(s) * 3 // 4)
    idx = 0
    for i in range(0, len(s), 4):
        a, b, c, d = base.index(s[i]), base.index(s[i+1]), base.index(s[i+2]), base.index(s[i+3])
        flag[idx] = (a << 2) | ((b >> 4) & 0x3)
        flag[idx + 1] = ((b & 0xf) << 4) | ((c >> 2) & 0xf)
        flag[idx + 2] = ((c & 0x3) << 6) | (d & 0x3f)
        idx += 3
    print(flag)
    print(''.join(chr(c) for c in flag))

if __name__ == '__main__':
    base = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
    base = shift(base)
    print(base)
    res = 'zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9'
    res = convert_little_big(res)
    print(len(res))
    solve(res, base)
```

`flag{bAse64_h2s_a_Surprise}`