ctf 逆向 暴力破解类的脚本

原创

cainiao78777 于 2022-03-15 19:32:26 发布 60 收藏

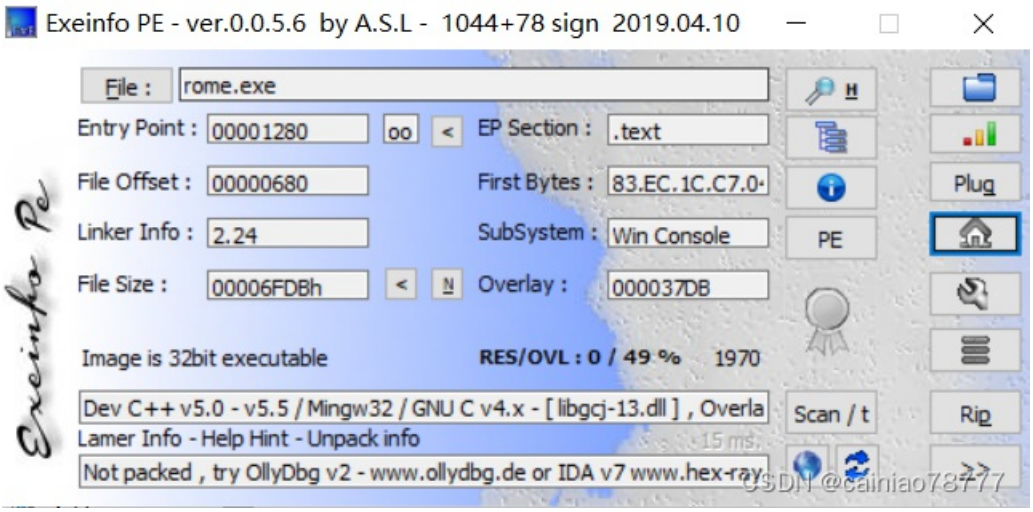
文章标签： 安全

本文链接：https://blog.csdn.net/cainiao78777/article/details/123509325

前言：在分析代码的时候常遇到不好逆向的逻辑，如求余（%）等，下面就是含有求余的两道逆向题目

例题一：buuctf [ACTF新生赛2020]rome

1.查壳



无壳32位exe.

2.ida32打开并分析main函数

发现主要函数 **func()**

```
int func()
{
  int result; // eax
  int v1; // [esp+14h] [ebp-44h]
  int v2; // [esp+18h] [ebp-40h]
  int v3; // [esp+1Ch] [ebp-3Ch]
  int v4; // [esp+20h] [ebp-38h]
  unsigned __int8 v5; // [esp+24h] [ebp-34h]
  unsigned __int8 v6; // [esp+25h] [ebp-33h]
  unsigned __int8 v7; // [esp+26h] [ebp-32h]
  unsigned __int8 v8; // [esp+27h] [ebp-31h]
  unsigned __int8 v9; // [esp+28h] [ebp-30h]
  int v10; // [esp+29h] [ebp-2Fh]
  int v11; // [esp+2Dh] [ebp-2Bh]
  int v12; // [esp+31h] [ebp-27h]
  int v13; // [esp+35h] [ebp-23h]
  unsigned __int8 v14; // [esp+39h] [ebp-1Fh]
  char v15; // [esp+3Bh] [ebp-1Dh]
  char v16; // [esp+3Ch] [ebp-1Ch]
```

```
char v17; // [esp+3Dh] [ebp-1Bh]
char v18; // [esp+3Eh] [ebp-1Ah]
char v19; // [esp+3Fh] [ebp-19h]
char v20; // [esp+40h] [ebp-18h]
char v21; // [esp+41h] [ebp-17h]
char v22; // [esp+42h] [ebp-16h]
char v23; // [esp+43h] [ebp-15h]
char v24; // [esp+44h] [ebp-14h]
char v25; // [esp+45h] [ebp-13h]
char v26; // [esp+46h] [ebp-12h]
char v27; // [esp+47h] [ebp-11h]
char v28; // [esp+48h] [ebp-10h]
char v29; // [esp+49h] [ebp-Fh]
char v30; // [esp+4Ah] [ebp-Eh]
char v31; // [esp+4Bh] [ebp-Dh]
int i; // [esp+4Ch] [ebp-Ch]

v15 = 81;
v16 = 115;
v17 = 119;
v18 = 51;
v19 = 115;
v20 = 106;
v21 = 95;
v22 = 108;
v23 = 122;
v24 = 52;
v25 = 95;
v26 = 85;
v27 = 106;
v28 = 119;
v29 = 64;
v30 = 108;
v31 = 0;
printf("Please input:");
scanf("%s", &v5);
result = v5;
if ( v5 == 65 )
{
  result = v6;
  if ( v6 == 67 )
  {
    result = v7;
    if ( v7 == 84 )
    {
      result = v8;
      if ( v8 == 70 )
      {
        result = v9;
        if ( v9 == 123 )
        {
          result = v14;
          if ( v14 == 125 )
          {
            v1 = v10;
            v2 = v11;
            v3 = v12;
            v4 = v13;
            for ( i = 0; i <= 15; ++i )
            {
```

```
            {
              if ( *((_BYTE *)&v1 + i) > 64 && *((_BYTE *)&v1 + i) <= 90 )
                *((_BYTE *)&v1 + i) = (*((char *)&v1 + i) - 51) % 26 + 65;
              if ( *((_BYTE *)&v1 + i) > 96 && *((_BYTE *)&v1 + i) <= 122 )
                *((_BYTE *)&v1 + i) = (*((char *)&v1 + i) - 79) % 26 + 97;
            }
            for ( i = 0; i <= 15; ++i )
            {
              result = (unsigned __int8)*(&v15 + i);
              if ( *((_BYTE *)&v1 + i) != (_BYTE)result )
                return result;
            }
            result = printf("You are correct!");
          }
        }
      }
    }
  }
  return result;
}
```

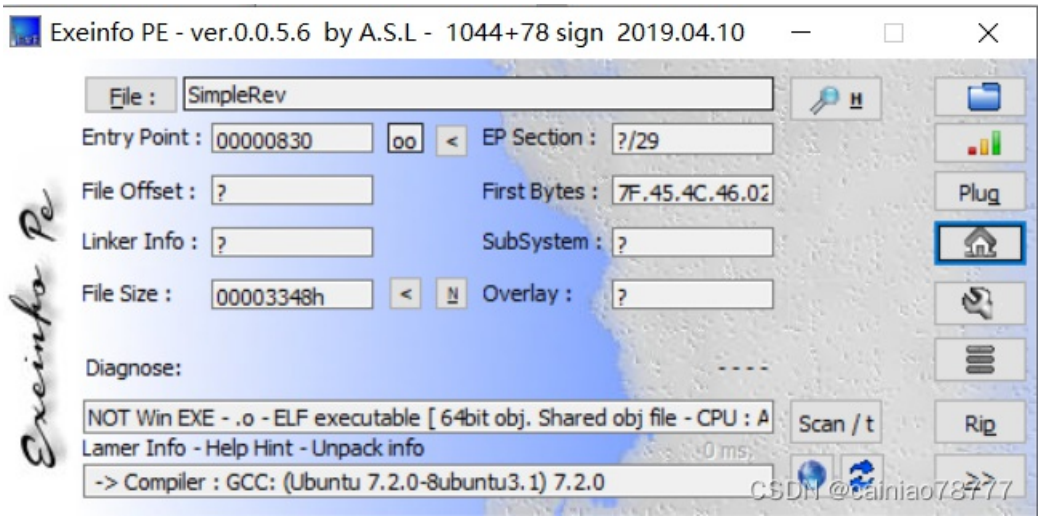
函数逻辑大概就是：一个字符组v1[ ]经过一系列变化，最后再与v15[ ]的进行比较是否相等
脚本如下：

```
v15=[81,115,119,51,115,106,95,108,122,52,95,85,106,119,64,108]
flag=''
for i in range(len(v15)):
    for j in range(0,127):      #目前ascll有127个
        a=j
        if(j>64 and j<=90):
            j=(j-51)%26+65
        if(j>96 and j<=122):
            j=(j-79)%26+97
        if(j==v15[i]):
            flag+=chr(a)
print(flag)
```

flag{Cae3ar_th4_Gre@t}
目的就是让数据经过上面的处理得到新数据，如果新数据和v15【】相等，那么就取它的值

buuctf SimpleRev

1.查壳



elf64位文件，

2.用ida64打开，分析main函数

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  int v3; // eax
  char v4; // [rsp+Fh] [rbp-1h]

  while ( 1 )
  {
    while ( 1 )
    {
      printf("Welcome to CTF game!\nPlease input d/D to start or input q/Q to quit this program: ", argv, envp);
      v4 = getchar();
      if ( v4 != 100 && v4 != 68 )
        break;
      Decry();
    }
    if ( v4 == 113 || v4 == 81 )
      Exit();
    puts("Input fault format!");
    v3 = getchar();
    putchar(v3);
  }
}
```


跟进**Decry()** 函数

```
unsigned __int64 Decry()
{
  char v1; // [rsp+Fh] [rbp-51h]
  int v2; // [rsp+10h] [rbp-50h]
  int v3; // [rsp+14h] [rbp-4Ch]
  int i; // [rsp+18h] [rbp-48h]
  int v5; // [rsp+1Ch] [rbp-44h]
  char src[8]; // [rsp+20h] [rbp-40h]
  __int64 v7; // [rsp+28h] [rbp-38h]
  int v8; // [rsp+30h] [rbp-30h]
  __int64 v9; // [rsp+40h] [rbp-20h]
  __int64 v10; // [rsp+48h] [rbp-18h]
  int v11; // [rsp+50h] [rbp-10h]
  unsigned __int64 v12; // [rsp+58h] [rbp-8h]

  v12 = __readfsqword(0x28u);
  *(_QWORD *)src = 'SLCDN';    #记得大端序和小端序，下同。
  v7 = 0LL;
```

```
  v8 = 0;
  v9 = 'wodah';
  v10 = 0LL;
  v11 = 0;
  text = (char *)join(key3, &v9);
  strcpy(key, key1);
  strcat(key, src);
  v2 = 0;
  v3 = 0;
  getchar();
  v5 = strlen(key);
  for ( i = 0; i < v5; ++i )              #转换大小写
  {
    if ( key[v3 % v5] > 64 && key[v3 % v5] <= 90 )
      key[i] = key[v3 % v5] + 32;
    ++v3;
  }
  printf("Please input your flag:", src);
  while ( 1 )
  {
    v1 = getchar();
    if ( v1 == 10 )
      break;
    if ( v1 == 32 )
    {
      ++v2;
    }
    else
    {
      if ( v1 <= 96 || v1 > 122 )
      {
        if ( v1 > 64 && v1 <= 90 )
          str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;
      }
      else
      {
        str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;
      }
      if ( !(v3 % v5) )
        putchar(32);
      ++v2;
    }
  }
  if ( !strcmp(text, str2) )
    puts("Congratulation!\n");
  else
    puts("Try again!\n");
  return __readfsqword(0x28u) ^ v12;
}
```

大致逻辑是：将输入的v1经过处理和text【】进行比较是否相等。
发现处理过程有 % 操作，所以考虑用暴力破解。
脚本如下：

```
text = 'killshadow'
key = 'adsfkndcls'
v3=0
v5=len(key)
dict1="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" #字典，从处理过程可以看出，v1只含英语字母
flag = ""
for i in range(0,10):
    n=0
    for char in dict1:
        x = (ord(char) - 39 - ord(key[v3%v5])+97)%26+97
        if(chr(x)==text[i]):
            n = n+1
            if(n==1):
                print(char,end="")
    v3=v3+1
```

flag{KLDQCUDFZO}

总结：

1.在有结果的时候，并且逆向逻辑比较困难是可以考虑用暴力破解

2.字符串不是很长时，也可以考虑暴力破解