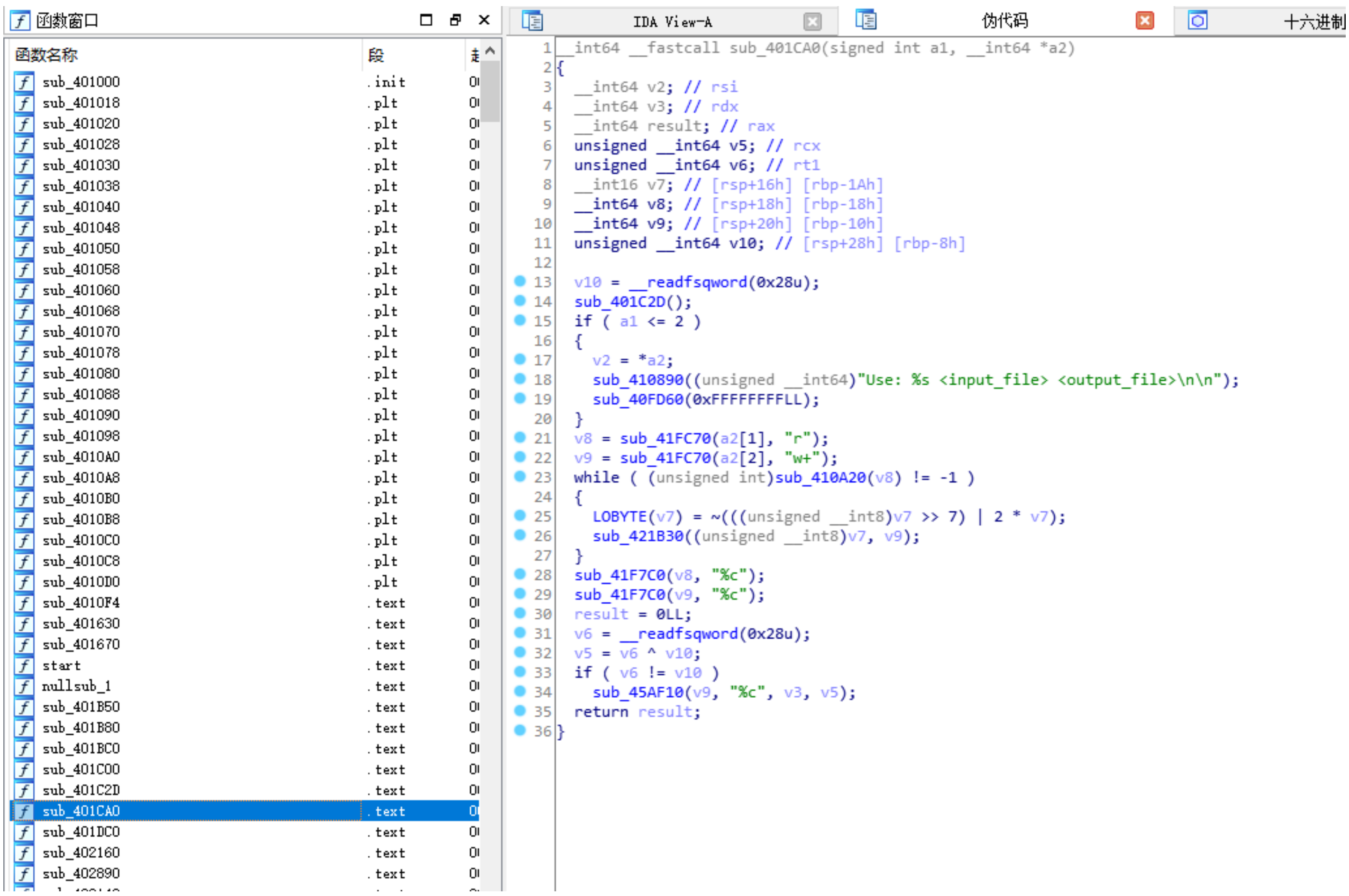
【FireShell WriteUp】Simple Encryption

原创

古月浪子 于 2020-03-22 21:24:16 发布 114 收藏

> FireShell CTF的第一道逆向题
> 做的时候还卡了一下，丢人233333~

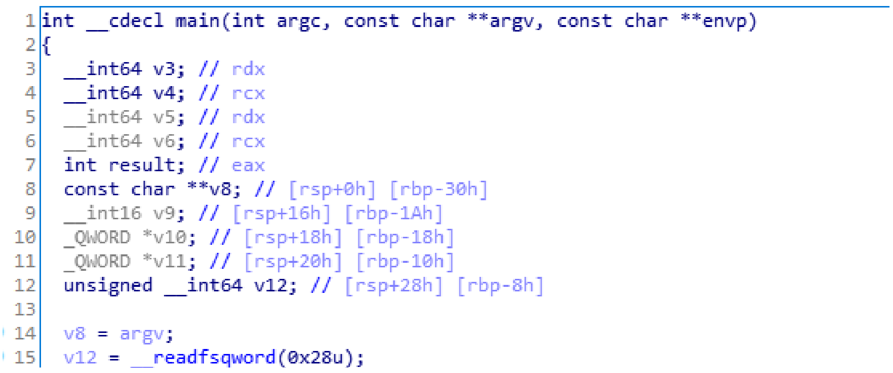
用IDA打开，找一找main函数



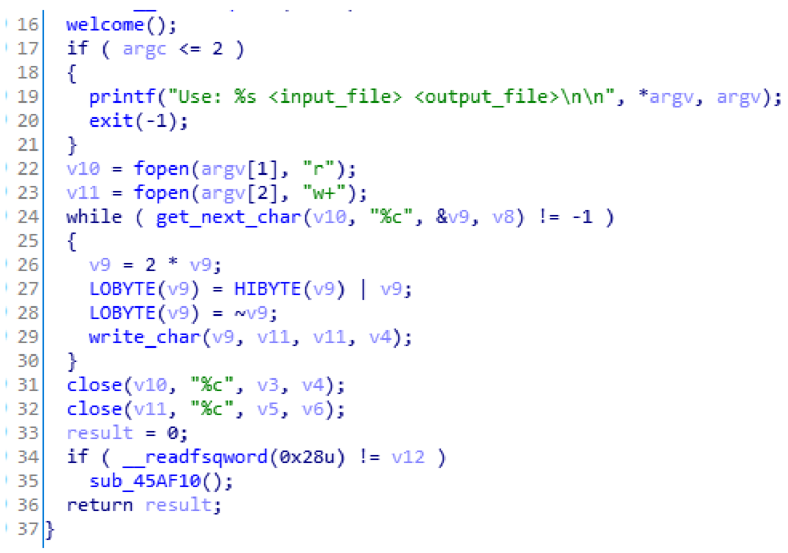
函数名称	段
sub_401000	.init
sub_401018	.plt
sub_401020	.plt
sub_401028	.plt
sub_401030	.plt
sub_401038	.plt
sub_401040	.plt
sub_401048	.plt
sub_401050	.plt
sub_401058	.plt
sub_401060	.plt
sub_401068	.plt
sub_401070	.plt
sub_401078	.plt
sub_401080	.plt
sub_401088	.plt
sub_401090	.plt
sub_401098	.plt
sub_4010A0	.plt
sub_4010A8	.plt
sub_4010B0	.plt
sub_4010B8	.plt
sub_4010C0	.plt
sub_4010C8	.plt
sub_4010D0	.plt
sub_4010F4	.text
sub_401630	.text
sub_401670	.text
start	.text
nullsub_1	.text
sub_401B50	.text
sub_401B80	.text
sub_401BC0	.text
sub_401C00	.text
sub_401C2D	.text
sub_401CA0	.text
sub_401DC0	.text
sub_402160	.text
sub_402890	.text

```
__int64 __fastcall sub_401CA0(signed int a1, __int64 *a2)
{
  __int64 v2; // rsi
  __int64 v3; // rdx
  __int64 result; // rax
  unsigned __int64 v5; // rcx
  unsigned __int64 v6; // rt1
  __int16 v7; // [rsp+16h] [rbp-1Ah]
  __int64 v8; // [rsp+18h] [rbp-18h]
  __int64 v9; // [rsp+20h] [rbp-10h]
  unsigned __int64 v10; // [rsp+28h] [rbp-8h]

  v10 = __readfsqword(0x28u);
  sub_401C2D();
  if ( a1 <= 2 )
  {
    v2 = *a2;
    sub_410890((unsigned __int64)"Use: %s <input_file> <output_file>\n\n");
    sub_40FD60(0xFFFFFFFFLL);
  }
  v8 = sub_41FC70(a2[1], "r");
  v9 = sub_41FC70(a2[2], "w+");
  while ( (unsigned int)sub_410A20(v8) != -1 )
  {
    LOBYTE(v7) = ~(((unsigned __int8)v7 >> 7) | 2 * v7);
    sub_421B30((unsigned __int8)v7, v9);
  }
  sub_41F7C0(v8, "%c");
  sub_41F7C0(v9, "%c");
  result = 0LL;
  v6 = __readfsqword(0x28u);
  v5 = v6 ^ v10;
  if ( v6 != v10 )
    sub_45AF10(v9, "%c", v3, v5);
  return result;
}
```

因为没有符号表，而且有很多无关的代码，所以尝试着判断一下各个函数的功能，改一下名



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rdx
  __int64 v4; // rcx
  __int64 v5; // rdx
  __int64 v6; // rcx
  int result; // eax
  const char **v8; // [rsp+0h] [rbp-30h]
  __int16 v9; // [rsp+16h] [rbp-1Ah]
  _QWORD *v10; // [rsp+18h] [rbp-18h]
  _QWORD *v11; // [rsp+20h] [rbp-10h]
  unsigned __int64 v12; // [rsp+28h] [rbp-8h]

  v8 = argv;
  v12 = __readfsqword(0x28u);
```



```
welcome();
if ( argc <= 2 )
{
  printf("Use: %s <input_file> <output_file>\n\n", *argv, argv);
  exit(-1);
}
v10 = fopen(argv[1], "r");
v11 = fopen(argv[2], "w+");
while ( get_next_char(v10, "%c", &v9, v8) != -1 )
{
  v9 = 2 * v9;
  LOBYTE(v9) = HIBYTE(v9) | v9;
  LOBYTE(v9) = ~v9;
  write_char(v9, v11, v11, v4);
}
close(v10, "%c", v3, v4);
close(v11, "%c", v5, v6);
result = 0;
if ( __readfsqword(0x28u) != v12 )
  sub_45AF10();
return result;
}
```

现在可以比较清楚的看到程序的逻辑了，打开input_file，将字符串加密以后写入output_file
具体的加密方法就在while循环里，我们写一个逆向算法把加密后的flag解出来即可

```
int main()
{
 char flag[140] = { 0 };
 FILE* fflag = fopen("flag.enc", "rb");
 fgets(flag, 140, fflag);
 for (size_t i = 0; flag[i]; i++)
 {
  unsigned char tmp = ~flag[i];
  for (short j = 0; j < 65536; j++)
  {
   if ((unsigned char)j >> 7 | 2 * j == tmp)
   {
    cout << (char)j;
    break;
   }
  }
 }
}
```

比赛地址：FireShell