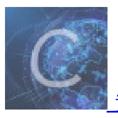
【CTF大赛】第五届XMan选拔赛 ezCM Writeup

原创

IT老涵 于 2021-08-12 15:48:31 发布 188 收藏 1

分类专栏： 安全 网络 CTF 文章标签： 网络安全 信息安全 计算机网络

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/HBohan/article/details/119645953
版权

 安全 同时被 3 个专栏收录

375 篇文章 21 订阅

订阅专栏

网络

355 篇文章 13 订阅

订阅专栏

CTF

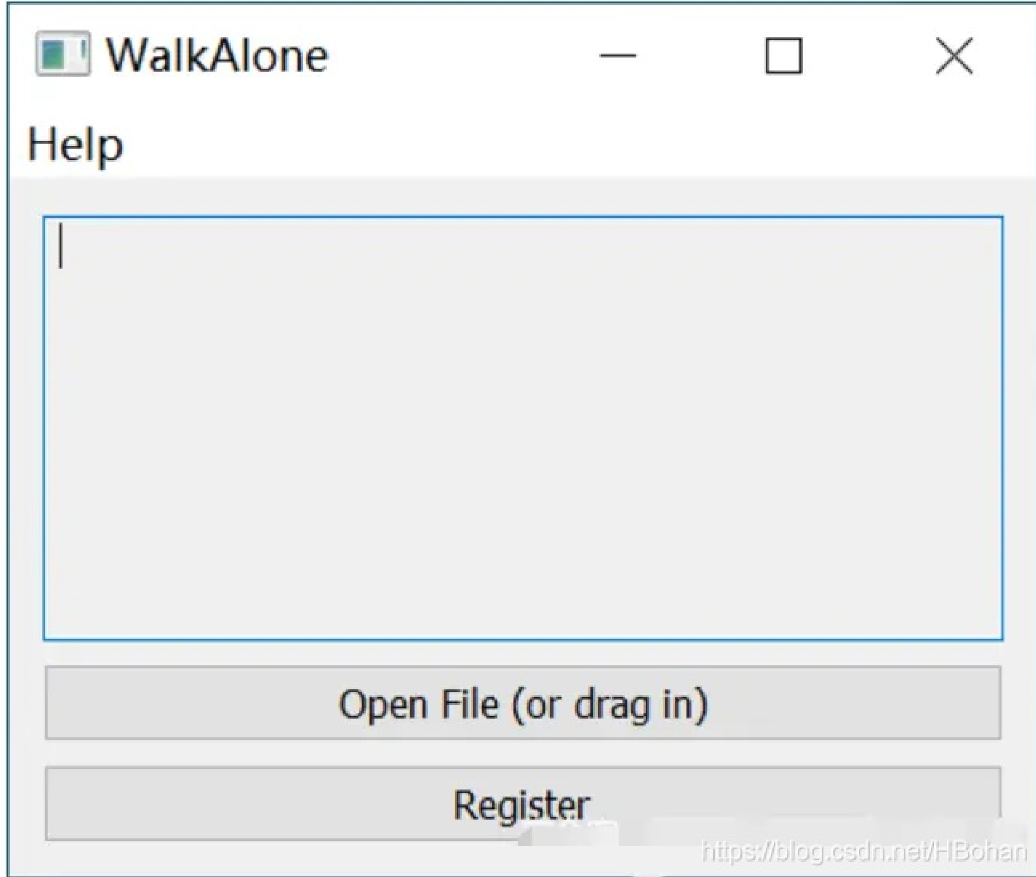
10 篇文章 3 订阅

订阅专栏

ezCM

直至比赛结束，这道题目都是 0 解题，一方面是因为比赛时间较短，另一方面还是因为这道题目较难，考察了不常见的椭圆曲线算法(ECC)，大大增加了对做题者的要求。

题目信息



题目是使用 Golang 来编写的一个 CrackMe 程序，程序内符号没有被去除，所以这篇文章就不会讲解如何恢复 Golang 程序符号，另外 IDA Pro 7.6 已经支持 Golang 程序分析，打开就可以直接恢复被去除的符号信息。

题目要求打开一个 KeyFile ，并且通过读取其文件的内容来注册程序，我们要做的就是通过分析程序验证方式来编写一个 KeyFile，使其可以通过程序注册验证，最终拿到 flag 数据。

前置知识

由于是 Golang 的题目，在一些数据结构和调用约定上和大多数语言都不一样，所以一定不能过于的依赖伪代码，在调试过程中最好能够多关注汇编代码，这样在逆向过程中会快速掌握到核心。这部分内容参考学习了 panda0s – Golang underlying data representaion ，本来是不想把这部分内容放在这篇文章中的，但是由于关联性过大，所以不得不拿来饱满文章内容。

函数调用

在函数调用的过程中，无论是调用参数还是返回值都是通过栈来传递。

```
mov     rax, [rsp+250h+obj]
mov     [rsp+250h+var_250], rax ; 传入参数1
mov     rcx, qword ptr [rsp+250h+var_A8]
mov     qword ptr [rsp+250h+var_248], rcx ; 传入参数2
mov     rcx, [rsp+250h+var_1F8]
mov     qword ptr [rsp+250h+var_248+8], rcx ; 传入参数3
call    main___ptr_MyMainWindow__Academy
mov     rax, [rsp+250h+var_238] ; 返回值1
```

https://blog.csdn.net/HBohan

其传参的特征是

1. 参数传递顺序是从右往左传递，而且不使用像是 push pop 这样的操作栈的指令，而是直接对栈上的内容进行修改。
2. 参数传递一般都是借助一个寄存器中转，例如
   rax、rcx，先把数据原来的储存位置的数据赋值到这个寄存器上，然后再把这个寄存器的内容赋值给栈上数据，并且如果数据是 0x10
   size 的结构体，就会借助 xmm 寄存器中转来加速。

其返回值的特征是

1. 返回值的位置紧贴着在最后一个参数的地址之后。以上图为例，最后一个参数的地址是 rsp + 0x250 – 0x248 + 0x8 = rsp + 0x10，所以这里的返回值的地址就是在 rsp + 0x250 – 0x238 = rsp +
   0x18，有多个返回值的情况也是类似。

方法调用

在上图中，严格意义上并不是一次函数调用，而是一次方法调用。他是对 MyMainWindow 这个对象下的 Academy 方法进行了调用，这个传入的参数就是这个对象的指针，像是 this 一样。这个对象的指针就相对于函数调用的第一个参数。

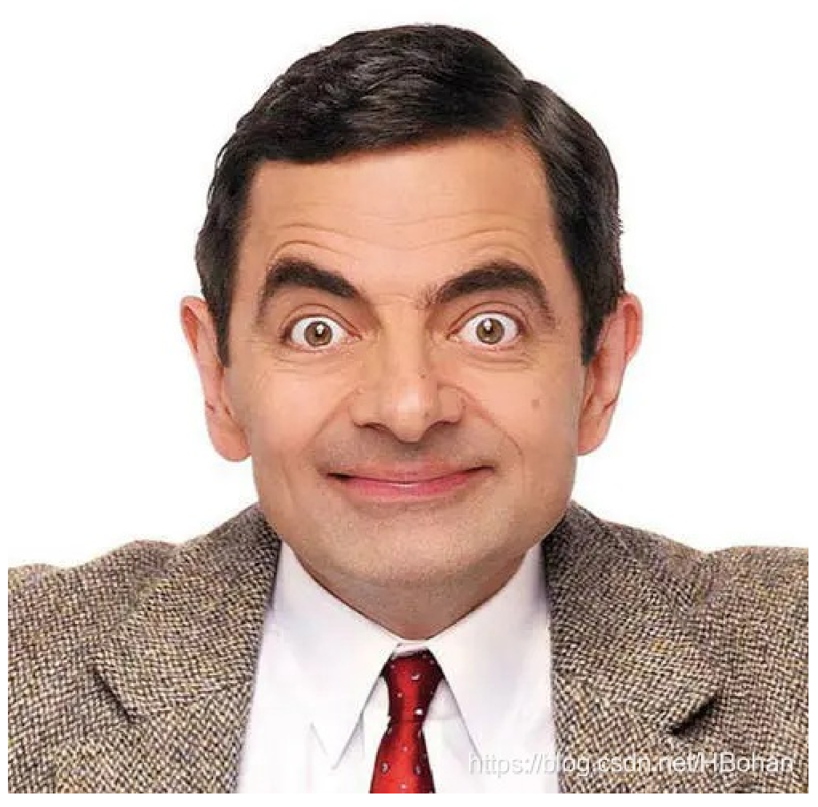
String 字符串

```
lea     rax, aCannotOpenTarg ; "Cannot open target file."
mov     qword ptr [rsp+250h+var_230], rax ; 字符串指针
mov     qword ptr [rsp+250h+var_230+8], 18h ; 字符串长度
mov     qword ptr [rsp+250h+var_230+10h], 10h
xchg    ax, ax
call    github_com_lxn_walk_MsgBox
```

String 结构

```
struct String{
    char * strPtr;
    int64 size;
}
```

所以 Golang 程序在传递字符串的时候，同时也会在后续跟一个参数，这个参数指的就是字符串的长度。同时由于这样的机制，使得字符串的内容在内存中分布不需要截止符'\x00'



Slice 切片

在其他语言中（例如 python），Slice 是一种切片的操作，切片之后可以返回一个新的数据对象，但是 Golang 中的 Slice 不仅仅是一种切片的操作，更像是一种灵活的数据结构。

了解 Slice 结构后，在 IDA 中修改对应的变量类型，可以大大加快分析速度。

Slice 结构

```
struct slice {
    dq     Pointer;
    dq     Length;
    dq     Capacity;
}
```

Pointer：指向 Slice 底层数组的元素开始位置的指针

Length：Slice 的当前长度

Capacity：Slice 底层数组的最大长度，超过此长度会自动扩展

初始化 Slice

```
my_slice := make([]int, 3, 5)
```

这表示先声明一个长度为 5、数据类型为 int 的底层数组，然后从这个底层数组中从前向后取 3 个元素作为 slice 的结构（length = 3，cap = 5）

make 最底层调用 runtime_makeslice 分配空间，这个函数返回的是指向内部数组的指针

访问 Slice

```
org_len := slice1[name_size + 1]
```

```
if ( slice.len <= name_size + 1 )
  runtime_panicIndex();
org_len = slice.ptr[name_size + 1];
```

在访问 Slice 中元素时，会检测是否越界如果越界则调用 runtime_panicIndex

append / copy

当 Length 已经等于 Capacity 的时候，再使用 append 给 slice 追加元素，会调用 runtime_growslice 进行扩容。

在代码中的表现是在 append / copy 的时候会检测，slice.len + 1 与 slice1.cap 的大小关系

```
if ( name_size + 1 > slice.cap )
  runtime_panicSliceAcap();
_name_len = *slice.ptr;
copy_slice.ptr = &slice.ptr[((1 - slice.cap) >> 63) & 1];
copy_slice.len = _name_len;
str_name = runtime_slicebytetostring(0LL, copy_slice);
```

如图在把 slice 转换为字符串的过程前，由于将要 copy slice，所以会对传参 len 进行检测。

切片截取

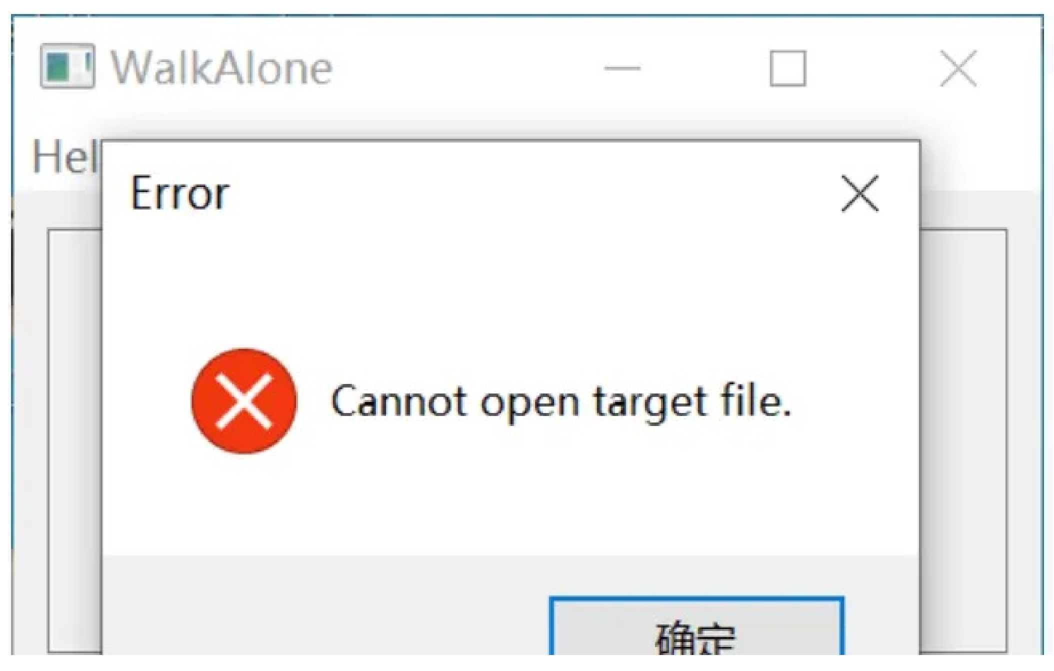
```
myvar := slice1[a:b]
```

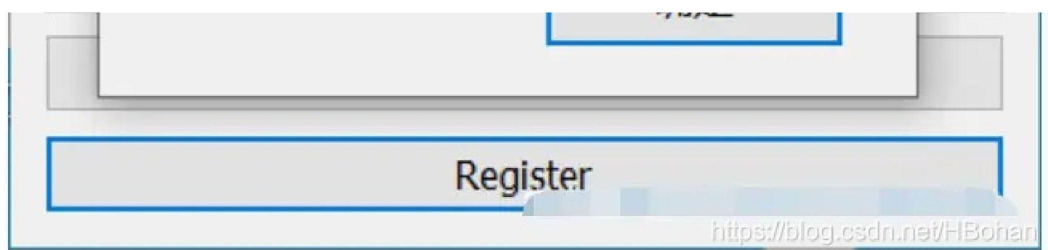
myvar 是新一个新的切片结构
dataPtr = &slice1.dataPtr[1]，相当于给了一个底层数组的指针
len = b – a
cap = slice1.cap – a

寻找关键函数

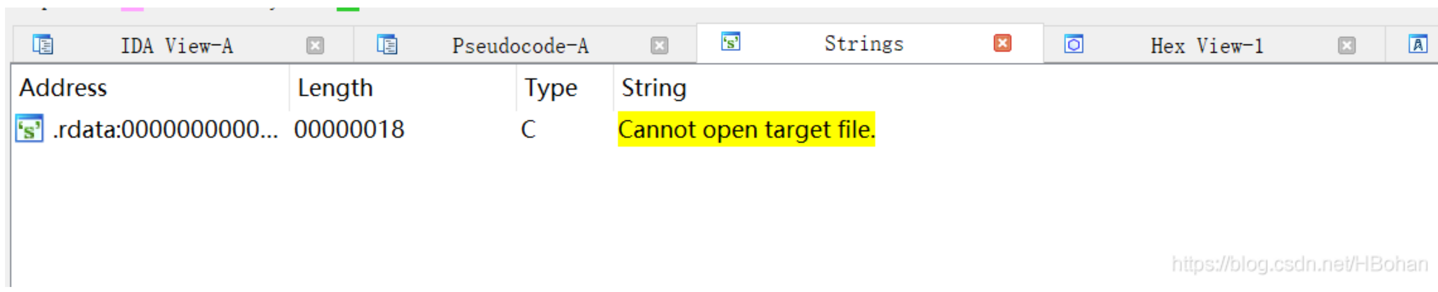
对于这种有界面的程序，和常规的只有一个控制台的题目不同的是，题目的关键信息不是直接存在于 main 函数中，所以我们首先要做的就是定位到题目的关键位置。

而在这道题里，我们的突破口就是在没有选择文件的情况下，点击“Register”就会弹出的信息框“Cannot open target file.”





我们可以利用 IDA 的 Shift + F12 热键调出 Strings 窗口来查找“Cannot open target file.”字符串



搜索字符串后，我们再双击进入，在前面自动生成的名称处按下 X 热键查找交叉引用

```
.rdata:0000000000739A6F                                     ; DATA XREF: golang_org_x_sys_windows_init+623↑o
.rdata:0000000000739A87 aCannotOpenTarg db 'Cannot open target file.'
.rdata:0000000000739A87                                     ; DATA XREF: main___ptr_MyMainWindow__Aback+294↑o
.rdata:0000000000739A87
.rdata:0000000000739A9F
.rdata:0000000000739A9F
.rdata:0000000000739AB7
.rdata:0000000000739AB7
.rdata:0000000000739AB7
.rdata:0000000000739ACF
.rdata:0000000000739ACF
.rdata:0000000000739AE7
.rdata:0000000000739AE7
.rdata:0000000000739AE7
.rdata:0000000000739AFF aGdipcreatebitm db 'GdipCreateBitmapFromFile'
```

```
xrefs to aCannotOpenTarg
Directio Ty Address                        Text
Up       o  main__ptr_MyMainWindow_Aba...  lea   rax, aCannotOpenTarg; "Cannot open target file."
Up       o  main__ptr_MyMainWindow_Acc...  lea   rax, aCannotOpenTarg; "Cannot open target file."
Line 1 of 2
OK   Cancel   Search   Help
```

对于每一处引用我们都前去查看，最终找到了关键的代码位置（截图仅截取部分代码）

```
while ( (unsigned __int64)v56 <= *(_QWORD *)(*(_QWORD *)NtCurrentTeb()->NtTib.ArbitraryUserPointer + 16LL) )
  runtime_morestack_noctxt();
v69 = 0LL;
v39 = os_OpenFile(main_fn, qword_92AA18, 0LL, 0);
if ( v34 )
{
  github_com_lxn_walk_MsgBox(
    (__int64)&go_itab__main_MyMainWindow_github_com_lxn_walk_Form,
    data_a,
    (__int64)"Error",
    5LL,
    (__int64)"Cannot open target file.",
    24LL,
    16LL);
  return;
}
*((_QWORD *)&v69 + 1) = &off_742C28;
*(_QWORD *)&v69 = v28;
v1 = ((__int64 (*)(void))loc_4FCB6A)();
v68[0] = (__int64)&go_itab__os_File_io_Reader;
v68[1] = v1;
v68[2] = (__int64)&off_742320;
v68[3] = 0x10000LL;
v2 = 0;
v3 = 0LL;
for ( i = 0LL; ; i = v6 )
{
  v54 = v3;
  v66 = i;
  v52 = v2;
  v29 = bufio___ptr_Scanner__Scan((__int64)v68, v17);
  if ( !v18 )
    break;
  v35 = runtime_slicebytetostring(0LL, v68[4], v68[5]);
  v5 = v29 == 16;
  if ( v29 == 16 )
  {
    if ( *v22 == 'NIGEB---' )
    {
      if ( v22[1] == '---TREC ' )
      {
        v2 = 1;
        v3 = v54;
        v6 = v66:
```

```
        continue;
      }
      v5 = v29 == 16;
    }
    else
    {
      v5 = v29 == 16;
    }
  }
```

https://blog.csdn.net/HBohan

我们接下来对函数内容进行分步骤的解析

KeyFile 格式解析

特征格式

```
for ( i = 0LL; ; i = v6 )
{
  v53 = v3;
  v65 = i;
  v51 = v2;
  v28 = bufio___ptr_Scanner__Scan((__int64)v67, v17);
  if ( !v18 )
    break;
  v34 = runtime_slicebytetostring(0LL, v67[4], v67[5]);
  v5 = v28 == 16;
  if ( v28 == 16 )
  {
    if ( *(_QWORD *)v21 == 'NIGEB---' )
    {
      if ( *((_QWORD *)v21 + 1) == '---TREC ' )
      {
        v2 = 1;
        v3 = v53;
        v6 = v65;
        continue;
      }
      v5 = v28 == 16;
    }
    else
    {
      v5 = v28 == 16;
    }
  }
  if ( v5 && *(_QWORD *)v21 == ' DNE----' && *((_QWORD *)v21 + 1) == '----TREC' )
    break;
  v2 = v51;
  if ( v51 )
  {
    v42 = runtime_concatstring2(0LL, v65, v53, (_DWORD)v21, v28);
    v8 = v34;
    v7 = v38;
    v2 = v51;
  }
  else
  {
    v7 = v53;
    v8 = v65;
  }
  v3 = v7;
  v6 = v8;
}
```

https://blog.csdn.net/HBohan

这部分内容虽然伪代码看起来混乱，但是大概可以猜测是开头和结尾的特征字符

```
---BEGIN CERT---
xxx
----END CERT----
```

通过这两个特征读取出关键的秘钥信息 xxx，然后传入到后续函数，这样的标记格式在其他地方也很常见，所以这里不着重分析。

解密核心数据

在后续函数中的对秘钥核心数据做了一个解码

```
v19[0] = encoding_base64___ptr_Encoding__DecodeString(encoding_base64_StdEncoding, a2, *((__int64 *)&a2 + 1));// base64解码
v2 = v19[1];
data = (_BYTE *)v19[0];
if ( !v19[3] )
{
  v19[16] = v19[0];
  v19[8] = v19[1];
  i = 0LL;
  v5 = 0LL;
  v6 = 0LL;
  v7 = (unsigned __int8 *)&v19[5] + 7;
  while ( i < (__int64)v2 )
  {
    if ( i >= v2 )
      runtime_panicIndex();
    if ( i + 1 >= v2 )
      runtime_panicIndex();
    v8 = data[i] ^ 0xAA;                        // 异或加密
    if ( v8 != (data[i + 1] ^ 0x55) )          // 校验码
      return;
    v9 = v6 + 1;
    if ( v6 + 1 > v5 )
    {
      v19[7] = i;
      HIBYTE(v19[5]) = v8;
      v19[6] = v6;
      v19[5] = runtime_growslice((__int64)&unk_6CA8E0, (__int64)v7, v6, v5, v6 + 1, v19[2], v19[3], v19[4]);
      v7 = (unsigned __int8 *)v19[2];
      v5 = v19[4];
      v9 = v19[3] + 1;
      data = (_BYTE *)v19[16];
      v2 = v19[8];
      i = v19[7];
      v6 = v19[6];
      v8 = HIBYTE(v19[5]);
    }
    v7[v6] = v8;
    i += 2LL;
    v6 = v9;
  }
  main___ptr_MyMainWindow__Abiding(data_a, v7, v6, v5);
```

https://blog.csdn.net/HBohan

这里上述代码中可以看出，程序先通过一个 base64 解密对中间部分内容进行解码，然后以两个字节为一个单位进行解密，对第一字节异或 0xAA 得到数据，并且对第二字节异或 0x55，与第一字节的内容进行比对，如果不同则直接退出。

数据结构格式

解密后的数据是如何在存放的，分别又代表着那些信息？想要知道这些就要分析接下来所做的代码。

```
if ( !a.len )
  runtime_panicIndex();
name_len = (unsigned __int8)*a.data;
if ( name_len <= 15 && a.len >= (__int64)(name_len + 1) )// username长度要求小于16
{
  if ( name_len + 1 > a.cap )
    runtime_panicSliceAcap();
  __name_len = (unsigned __int8)*a.data;
  ptr_name = runtime_slicebytetostring(0LL, (__int64)&a.data[((1 - a.cap) >> 63) & 1], name_len);
  str_username = v31;
  if ( runtime_writeBarrier )
    runtime_gcWriteBarrier();
  else
    main_name = ptr_name;
  result = (__int64 *)(name_len + 1);
  if ( a.len <= name_len + 1 )
    runtime_panicIndex();
  org_len = (unsigned __int8)a.data[name_len + 1];
  if ( org_len <= 0xF )                         // org长度要求小于16
  {
    v5 = org_len + name_len + 2;
    if ( a.len >= (__int64)v5 )
    {
      if ( v5 > a.cap )
        runtime_panicSliceAcap();
      if ( v5 < name_len + 2 )
        runtime_panicSliceB(v17, v21);
      v37 = org_len + name_len + 2;
```

```
    v36 = (unsigned __int8)a.data[name_len + 1];
    ptr_org = (__int64 *)runtime_slicebytetostring(
                           0LL,
                           (__int64)&a.data[((__int64)(2 - (a.cap - name_len)) >> 63) & (name_len + 2)],
                           org_len);
    result = ptr_org;
    str_org = v31;
    v6 = name_len;
    v7 = org_len;
    if ( runtime_writeBarrier )
      result = (__int64 *)runtime_gcWriteBarrier();
    else
      main_org = (__int64)ptr_org;
```

https://blog.csdn.net/HBohan

代码中由于对切片做了很多索引操作，所以有各种各样的越界检测，我们抛开这部分代码来看，就可以看出 Username 和 Organization 的储存结构 —— 第一个字节存放字符串长度，后续跟字符数据。

```
    main_org = (__int64)ptr_org;
  if ( a.len >= (__int64)(org_len + name_len + 20) )
  {
    v8 = v7 + v6 + 10;
    if ( v8 > a.cap )
      runtime_panicSliceAcap();
    if ( v37 > v8 )
      runtime_panicSliceB(v18, v22);
    v35 = org_len + __name_len + 20;
    v34 = v7 + v6 + 10;
    big_a = (_QWORD *)runtime_newobject((__int64)&unk_6ECE00);
    other_len = a.cap - (org_len + __name_len);
    v9 = big_a;
    _name_len = v36 + __name_len + 18;
    if ( runtime_writeBarrier )
    {
      runtime_gcWriteBarrierR9();
      v9 = v16;
    }
    else
    {
      main_a = (__int64)big_a;
    }
    slice_a.data = &a.data[((2 - other_len) >> 63) & v37];
    slice_a.len = 8LL;
    slice_a.cap = other_len - 2;
    v10 = math_big_nat_setBytes(v9[1], v9[2], v9[3], slice_a);// main_a = toBig(a[org_len + name_len + 2:org_len + name_len + 2 + 8])
    v9[2] = v32;
    v9[3] = v33;
    if ( runtime_writeBarrier )
      runtime_gcWriteBarrier();
    else
      v9[1] = v10;
    *(_BYTE *)v9 = 0;
    if ( _name_len > a.cap )
      runtime_panicSliceAcap();
    if ( v34 > _name_len )
      runtime_panicSliceB(v19, v24);
    big_b = (__int64 *)runtime_newobject((__int64)&unk_6ECE00);
    v11 = other_len - 10;
    v12 = &a.data[v34 & ((10 - other_len) >> 63)];
    v13 = big_b;
    if ( runtime_writeBarrier )
      runtime_gcWriteBarrierSI();
    else
      main_b = (__int64)big_b;
    slice_b.data = v12;
    slice_b.len = 8LL;
    slice_b.cap = v11;
    v14 = math_big_nat_setBytes(big_b[1], big_b[2], big_b[3], slice_b);// main_b = toBig(a[org_len + name_len + 10:org_len + name_len + 10 + 8])
    v13[2] = v32;
    v13[3] = v33;
    if ( runtime_writeBarrier )
      runtime_gcWriteBarrier();
    else
      v13[1] = v14;
    *(_BYTE *)v13 = 0;
    if ( v35 > a.cap )
      runtime_panicSliceAcap();
    if ( v35 < _name_len )
      runtime_panicSliceB(v20, v26);
    v15 = _name_len & ((18 - other_len) >> 63);
    result = (__int64 *)*(unsigned __int16 *)&a.data[v15];
    main_expire = *(_WORD *)&a.data[v15]; // 读取2字节的expire信息
  }
```

https://blog.csdn.net/HBohan

根据前置知识中切片的相关知识，这里调用 math_big_nat_setBytes 的切片内容我们可以大致还原，主要就是根据切片的 len 和 cap 来确定，

切片左边的值：由来源切片的 cap 减去的内容

切片右边的值：由来源切片左边的值 + 新切片的 len

所以 main_a 和 main_b 的内容来源于新的切片内容分别是

```
a[org_len + name_len + 2:org_len + name_len + 2 + 8]
a[org_len + name_len + 10:org_len + name_len + 10 + 8]
```

这部分内容可以结合上面的伪代码结合得出，也就是 Username 和 Organization 后的 8 字节是 main_a 的内容，再后 8 字节是 main_b 的内容，最后 4 字节是 main_expire 的内容。

这里需要注意的是，main_a 和 main_b 的内容都是以字节的形式直接转换为大数类型，而 main_expire 是以 WORD 的形式读取（使用 2 字节），这两种读取方式的字节序不同。

数据表

综合上面所说的，可以得出以下表格来记录 KeyFile 文件内加密数据格式

偏移	内容	变量名	长度
0	Username 长度	name_len	1
1	Username 内容	str_name	name_len
1 + name_len	Organization 长度	org_len	1
2 + name_len	Organization 内容	str_org	org_len
2 + name_len + org_len	内容 A	main_a	8
10 + name_len + org_len	内容 B	main_b	8
18 + name_len + org_len	过期时间	main_expire	2

https://blog.csdn.net/HBohan

验证逻辑

约束条件

了解了程序如何解析 KeyFile 后，接下来才是本文最关键的地方，也就是程序的验证方法。

```
if ( LOBYTE(v19[1]) )
{
  v34[0] = 0;
  v35 = 0LL;
  v36 = 0LL;
  math_big___ptr_Int__Add((__int64 *)v34, (char *)main_a, main_b);
  if ( v19[0] )
  {
    v19[2] = math_big_nat_itoa(
               *(_QWORD *)(v19[0] + 8),
               *(_QWORD *)(v19[0] + 16),
               *(_QWORD *)(v19[0] + 24),
               *(_BYTE *)v19[0],
               10LL);
    v19[0] = runtime_slicebytetostring((__int64)&v19[9], v19[2], v19[3]);
    v11 = (const char *)v19[0];
    sum_len = v19[1];
  }
  else
  {
    sum_len = 5LL;
    v11 = "<nil>";
  }
  if ( sum_len == 0x14 )                        // 检查a + b后的长度是否为20
  {
    v19[1] = runtime_memequal((__int64)v11, (__int64)"13417336609348053335", 0x14LL, v19[0]);// a + b == 0xba33f48ee008e957
```

https://blog.csdn.net/HBohan

首先会把 main_a 和 main_b 的内容相加，然后与 13417336609348053335（0xba33f48ee008e957）进行比对，如果相同则进入后续的判定，这就是对 a 和 b 之间关系的一个约束。

初始化大数

```
math_big___ptr_Int__SetString(          // gy
  (__int64)v31,
  (__int64)"74975941149641473430322663241721131468224546544767090640896879163372606900274",
  77LL,
  10LL);
v19[14] = v19[1];
v14 = runtime_newobject((__int64)&unk_6ECE00);
math_big___ptr_Int__SetString(          // p
  v14,
  (__int64)"115792089237316195423570985008687907853269984665640564039457584007908834671663",
  78LL,
  10LL);
if ( runtime_writeBarrier )
  runtime_gcWriteBarrier();
else
  main_p = v19[1];
v28[0] = 0;
v29 = 0LL;
v30 = 0LL;
math_big___ptr_Int__SetString(          // gx
  (__int64)v28,
  (__int64)"109469123007241702985147254990014649146957563280263710168135835850904453044386",
  78LL,
  10LL);
v19[15] = v19[1];
```

https://blog.csdn.net/HBohan

接下来又对三个大数进行了初始化，我把这几个常量去 Google 搜索了一下，发现 main_p 的值是在 GF§ 上的椭圆曲线中常用的一种取值，这对于我们了解接下来的代码的大致内容有所帮助。

这样根据常量来猜测程序意义的方法也是常用的，这里就是借助了椭圆曲线中常见的 p。

验证代码

```
v19[15] = v19[1];
main___ptr_MyMainWindow__Acacia();
v19[21] = v19[15];
v19[22] = v19[14];
main___ptr_MyMainWindow__get_np(data_a, *(__int128 *)&v19[21], v15);// public_key = scalar_mult(private_key, curve.g)
                                      //
*(_OWORD *)&v19[19] = *(_OWORD *)&v19[1];
main___ptr_MyMainWindow__point_add(data_a, v19[21], v19[22], v19[1], v19[2]);
*(_OWORD *)&v19[17] = *(_OWORD *)&v19[2];
v19[13] = v19[2];
v25[0] = 0;
v26 = 0LL;
v27 = 0LL;                            // on_curve
math_big___ptr_Int__Mul((__int64)v25, v19[3], v19[3]);// y ^ 2
math_big___ptr_Int__Mod((__int64)v25, (__int64)v25, main_p);// (y ^ 2) % p
k[0] = 0;
v23 = 0LL;
v24 = 0LL;
math_big___ptr_Int__Mul((__int64)k, v19[13], v19[13]);// x^2
math_big___ptr_Int__Mul((__int64)k, (__int64)k, v19[13]);// x^3
LOBYTE(v19[23]) = 0;
v20 = 0LL;
v21 = 0LL;
math_big___ptr_Int__Mul((__int64)&v19[23], main_a, v19[13]);// a * x
math_big___ptr_Int__Add((__int64 *)k, k, v19[0]);// x^3 + ax
math_big___ptr_Int__Add((__int64 *)k, k, main_b);// x^3 + ax + b
math_big___ptr_Int__Mod((__int64)k, (__int64)k, main_p);// (x^3 + ax + b) % p
math_big___ptr_Int__Cmp((__int64)v25, (__int64)k);// (y ^ 2) % p == (x^3 + ax + b) % p
```

https://blog.csdn.net/HBohan

最终的检测代码就是判断题目 public_key 是否在椭圆曲线（y^2 = x^3 + ax + b）上，其中 a 和 b 是用户可控的值，我们现在有一个在椭圆曲线上的点 生成元 G，那么我们就可以根据这个 G 点的值和 a + b 的约束来反推 a 和 b 的值。

```
(gy^2) % p                                     =   (gx^3 + a * gx + b) % p
(gy^2 - gx^3) % p                              =   (a * gx + b) % p
(gy^2 - gx^3 - (a + b)) % p                    =   (a * (gx - 1)) % p
(a * gx + b) % p - ((a + b) % p)               = (a * (gx - 1)) % p
((a * gx + b) % p - ((a + b) * inv(gx - 1)) % p = a % p
```

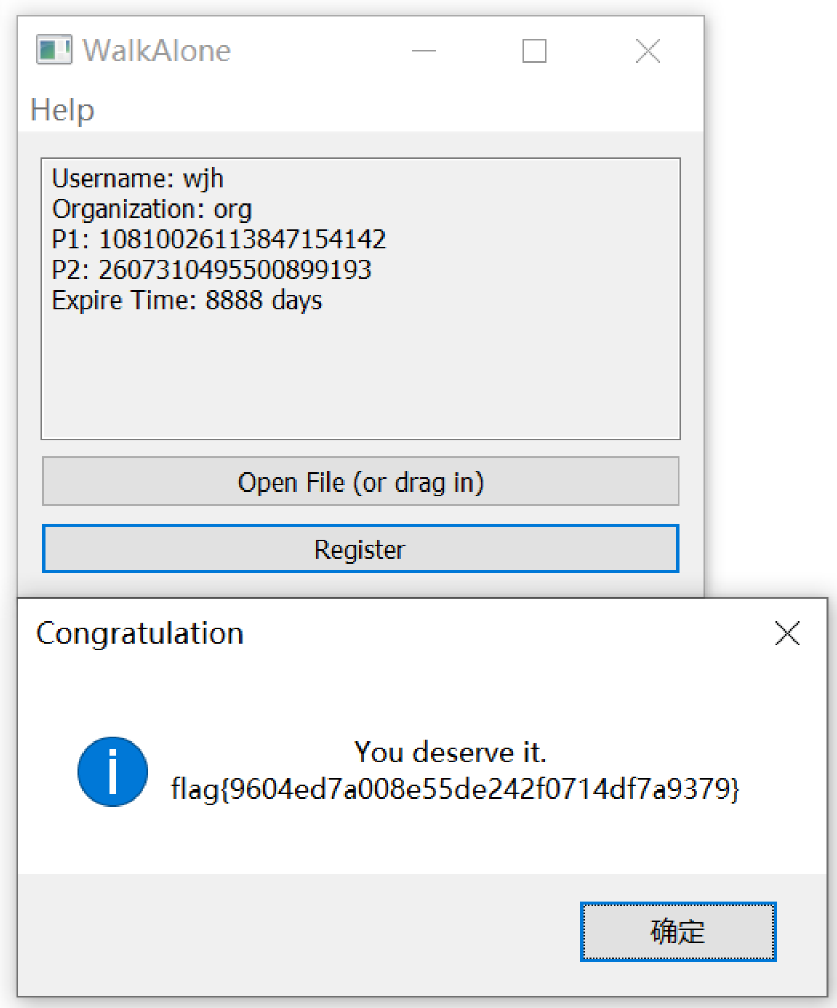
推导过程只是我粗浅的理解，所以可能不是很规范，但是表明了如何推出 a 的值，有了 a 的值后，我们直接相减就可以计算得到 b 的值。

注册机编写

通过上述的逻辑和整理，我们可以快速的编写出一个 Keygen，我的代码如下

```python
import base64
import gmpy2
from Crypto.Util.number import *

def calc_ab():
    gx = 0xf20553f3b02d1cad6aa8f895cc331a84b78f9bded26ecd9170662d3251d8d8a2
    gy = 0xa5c2e0fca8853a37f651726d719dd734421d0e01adf23c12c921e9060bc4c832
    p = 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
    sum = 0xba33f48ee008e957  # a + b
    # y^2 = x^3 + ax + b
    p1 = (gy * gy) % p   # y^2
    p2 = (gx * gx * gx) % p  # x^3
    p3 = (p1 - p2) % p   # ax + b
    p4 = (p3 - sum) % p  # ax + b - (a + b) = a * (x - 1)
    inv = gmpy2.invert(gx - 1, p)
    a = (p4 * inv) % p
    b = (sum - a) % p
    return a, b


def generate(Username, Organization, ExpireTime):
    if len(Username) >= 16:
        return ""
    if len(Organization) >= 16:
        return ""
    if ExpireTime > 0xffff:
        return ""
    reg_info = ""
    reg_info += chr(len(Username)) + Username
    reg_info += chr(len(Organization)) + Organization
    a, b = calc_ab()
    reg_info += long_to_bytes(a).rjust(8, '\x00') + long_to_bytes(b).rjust(8, '\x00')
    reg_info += long_to_bytes(ExpireTime).rjust(2, '\x00')[::-1]

    en_info = ''
    for i in reg_info:
        en_info += chr(ord(i) ^ 0xAA) + chr(ord(i) ^ 0x55)
    en_info = base64.b64encode(en_info)
    return '---BEGIN CERT---\n' + en_info + "\n----END CERT----"


with open("C:\\reg.crt", "w") as f:
    f.write(generate('wjh', 'org', 8888))
```

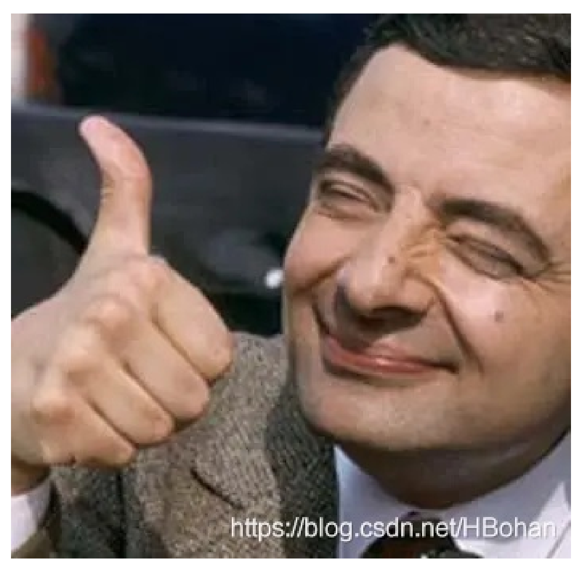
接下来把 KeyFile 拖入程序，点击 Register，即可成功通过验证得到 Flag



可以发现 flag 的值其实就是 main_a 和 main_b 的 hex 编码后的值，这样可以保证 flag 的唯一性。

总结

在这之前其实也遇到过一些 Golang 的题目，但是因为 Golang 难以分析，所以这些题目的核心算法相对来说都比较简单，都是一些比较简单的逻辑问题。这道题虽然分析过程看似简单轻松，但是实际上我对其内涵的原理和程序的用法进行了深入的研究，消耗了大量的时间和精力。虽然本题最终展现的并不是一个 ECC 难题，但是在逆向分析的过程中，我也学习到了一些 ECC 的内涵和代码实现。希望各位师傅可以借此题来开篇学习 Golang 逆向和 ECC 算法。



最后

我整理了相关的资料，有需要的朋友可以私我哦！！！

【资料】