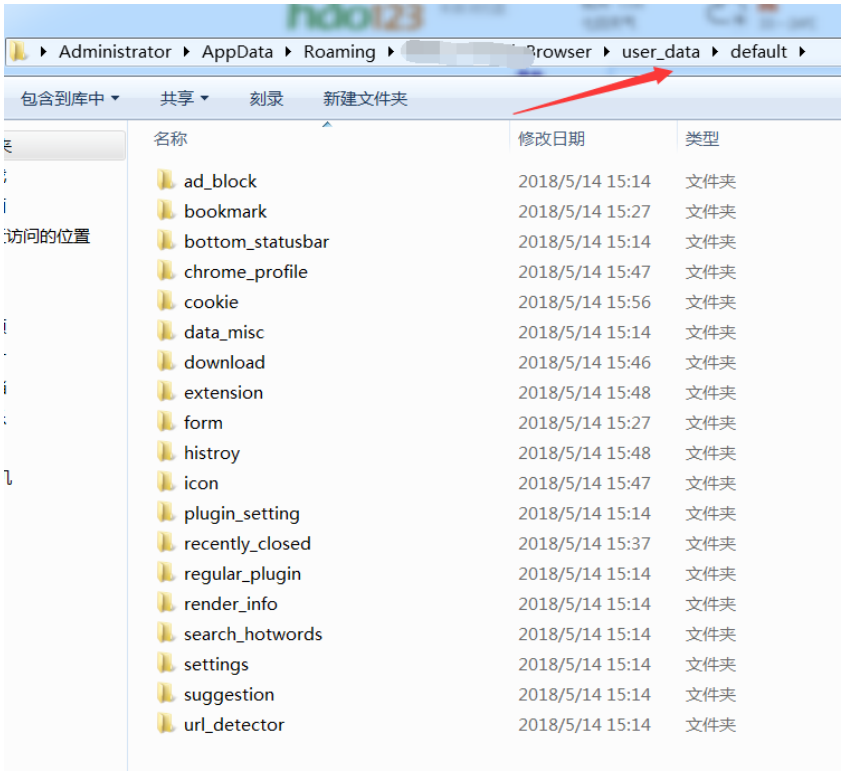
讨论一下从数字取证来看逆向的各种姿势

1. 序言
   a) 本来今天的研究院科普系列文章不是笔者我来写的，应该轮到隔壁的同学写文章，然而隔壁的死党周末回去扯证过上妻奴生活去了，于是只能由笔者代劳，但是我这二进制看来逆去哪有大黑阔门提权 webshell 过狗看的风骚，思来想去我想了半天周末在家冥思苦想不知道写点啥，然而！在我不知道写点啥的时候某个浏览器在虚拟机安装的时候居然稳定的 BSOD 了!!!!! 稳定的 BSOD!!!! 本想着就用这个浏览器 BSOD 开刀，但是老写驱动也没意思，这篇文章也不写那么多深奥的解析了，老大说写简单点，那今天我们来谈谈思路，逆向中的那么一些个思路，就用这个浏览器开刀吧！第二次写文章希望各位看客高抬贵手！老大说了不要写太难，我就不上表情包了！大家自行脑补表情包吧
2. 数字取证是什么
   i. 说简单点通过技术手段用于收集分析最后作为证据的刑侦手段，包括但不限于物理接触或者远程接触去获取必要数据的一些措施，物理接触当然时候直接取得犯罪嫌疑人的作案设备，这远程接触么，嘿嘿嘿你猜猜看！（此处应有斜眼笑表情包！）
3. 逆向在数字取证中的那些事
   a) 我们这次以某浏览器为例探讨下数字取证中关于逆向部分的一些思路技巧，我们先假定一个犯罪环境，某嫌疑犯在通过某浏览器访问一些特定网站，比如说自己的邮箱或者一些网盘的时候留下了访问痕迹，但是刑侦人员需要知道他在这些邮箱中具体和哪些人有来往，又或者他在网盘中保存了怎样的业务资料却一无所知，但是在他的浏览器中却保存了密码，我们来看看逆向手段怎么样读取这些密码，又有什么技巧
   b) 

   c) 首先我们梳理一下思路，程序保存数据一定会落地成为文件，或者写入不会断电就会消失的地方，反正无论怎么样一定是在硬盘上，不会再内存中，除非是无用文件，那么既然是文件的话，那么一定会涉及到高效的管理，如此说来数据库文件就是不二之选了，在成熟的商业软件中如何高效的管理数据呢，我猜不复杂的方式一般来说只有 sqlite 这种小型数据库，所以我们的目标就有了，找到这个文件

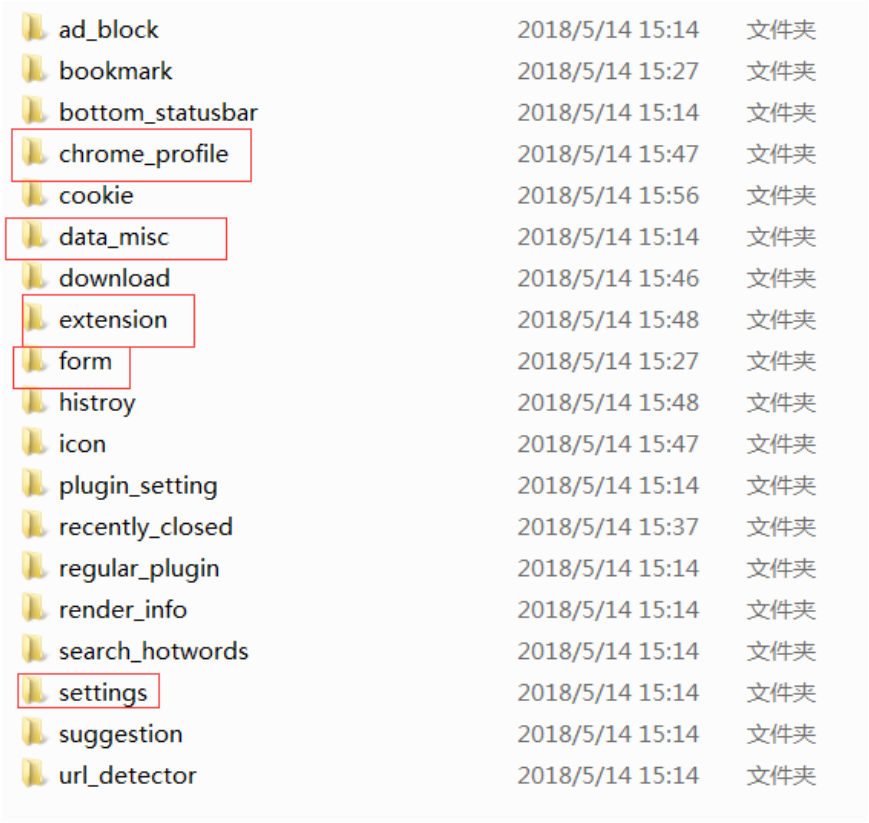
d) 那我们怎么定位这个文件呢，既然说了是技巧那么我们就找最简单最方便的方法，虽然路子野但是比较有效

i. 首先浏览器尤其是 chrome 类的一般会把数据放在 appdata 中，我们打开 appdata，找到对应的浏览器目录，从目录名称来分析（user_data\default），这个应该就是存放用户数据的地方了，而且列出来的文件夹确确实实很像是用户数据



ii.

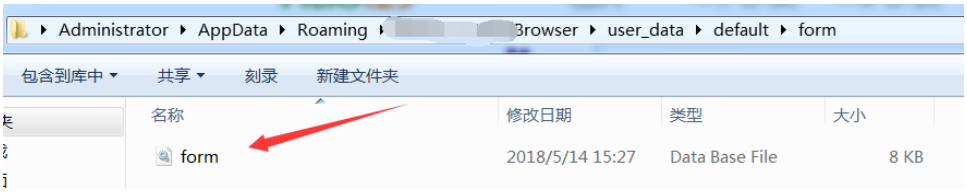
iii. 但是我们如何定位到我们想要的数据呢，野路子么，浏览器既然保存我们的密码一定会去落地成为文件，我们直接删除大法，只要数据文件被删除了，那自然也就成为炮灰了，浏览器也就不知道了，先从长的像的开始删什么 chrome_profile 啊 data_misc 之类的，哪些很明显的一看就不是的可以直接排除掉



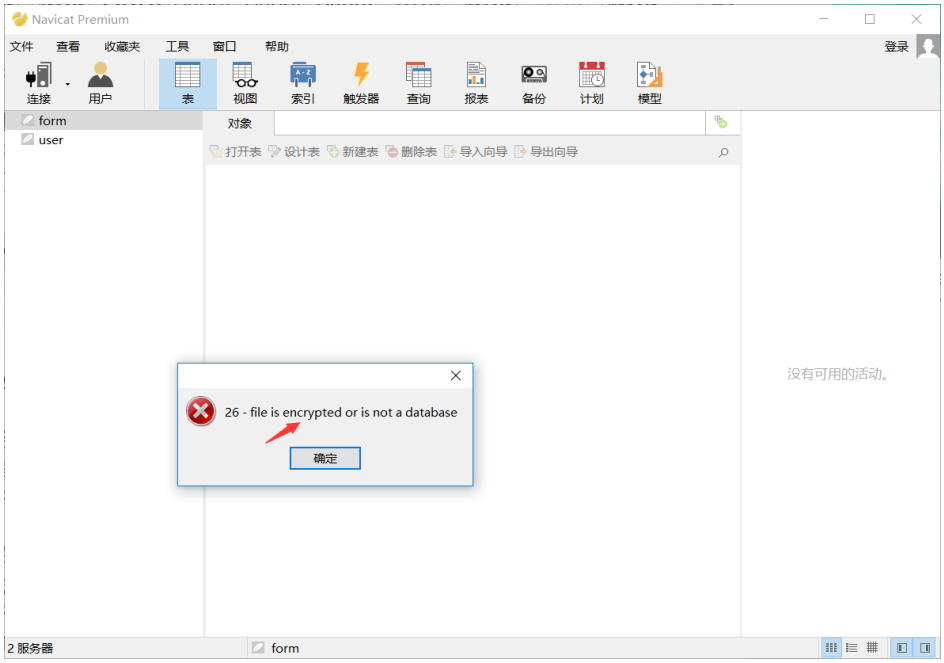
iv.

v. 于是通过几轮删除，我们很成功的定位到了保存用户密码的数据文件 form.db，嗯这名字起的，真是不知道是谁起的名字，你起个 userpwd 多好或者 password 多好，呵呵呵呵呵呵（此处省略 500 个呵呵……），叫什么 form，拖出去枪毙五分钟

vi.

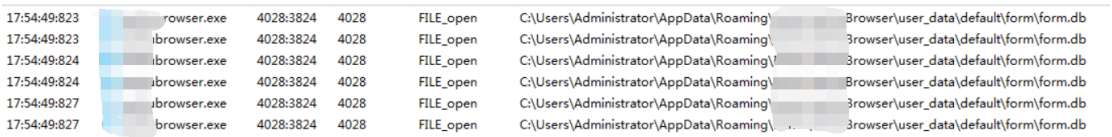


vii. 好了，数据文件有了，照着以往的思路拖出去 Navicat 伺候，你要以为这样就能连上？？？你们啊，naive，图样图森破，全文到此结束就太没意思，这和逆向有鸡毛关系!!!肯定是不行啊，Navicat 表示这个锅他不背，看下报错信息，嗯，英语我不认识，encrypt 还是认识的！这肯定是被加密了啊，得了找密钥吧！啥你们说不是 sqlite3?打开姿势不对？左上角那个退出看到没有，点一下谢谢!

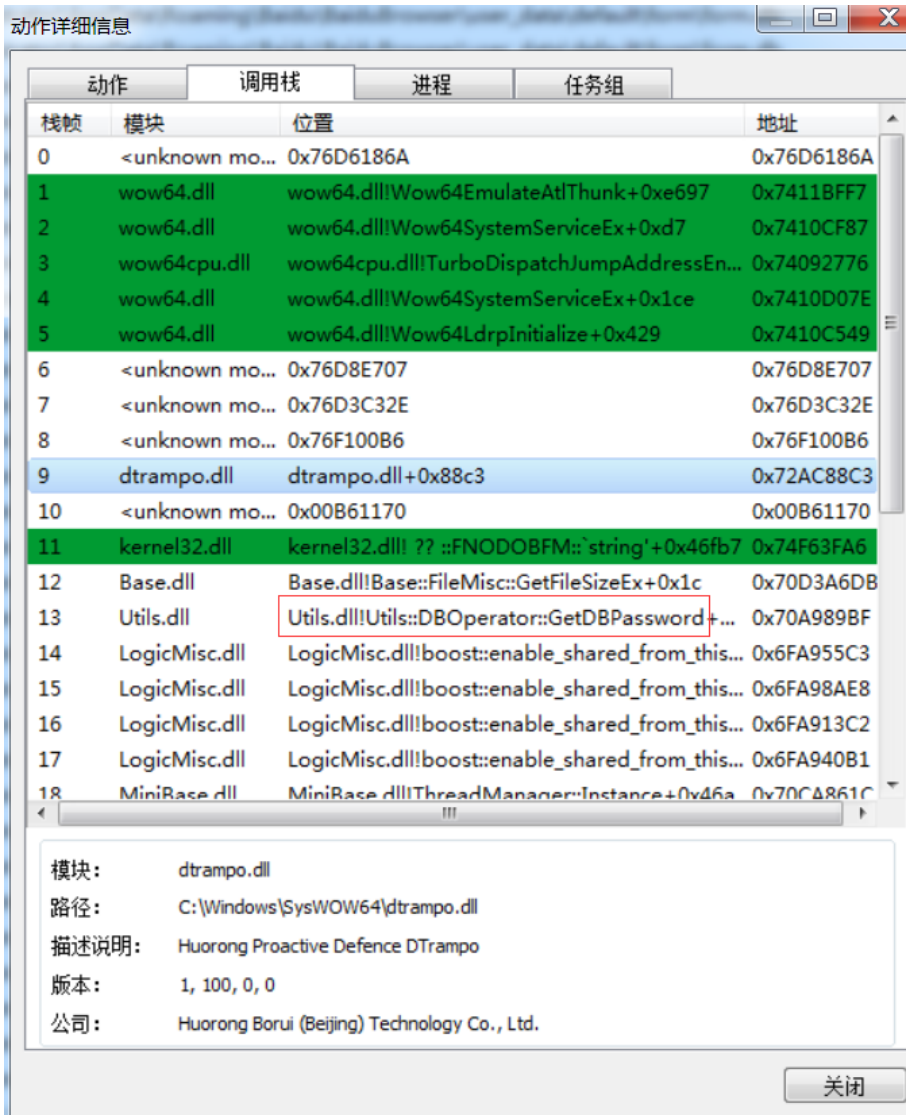
viii. 

ix. 一般来说正统的逆向思路肯定是打开 od，断 CreateFileW，对参数设置文件路径作为条件断点啊，然后巴拉巴拉巴拉巴拉的，你们累不累啊！你们能不能不要动不动就 OD 啊 windbg 啊，杀鸡焉用牛刀对不对，能使唤 IDA 何必上手 OD 啊，能使唤火绒剑何必祭出 IDA 啊，都说了野路子，谈方法对不对，来火绒剑走起

x. 一般来说呢 chrome 系的浏览器都是多进程带参数启动，这里如果贸然使用 od 的话很可能不是主 chrome 打开加密的数据文件，这也就意味着你的断点是无效的（关于这个处理我会在之后给出对应方法），所以很多新手在没有思路的时候贸然动用 OD 会发现很奇怪程序起来了，为什么 createfile 没有断下来，我们先要确定下到底是谁打开了这个文件，还有一个原因是因为浏览器的数据文件过多，而且大量的文件需要 CreateFile，这很有可能导致卡死，而且 od 在 x64 上支持的其实并不是非常好，尤其是他的字符串条件断点（笔者在这里吃过很多亏），所以能少用断点少用断点火绒剑有一个非常好的过滤功能和一个栈回溯功能，我们直接对其设定条件，在这三个条件的过滤下，我们很快就能发现，对这个文件处理的程序是浏览器主进程，因为他的父进程 ID 为 Explorer

1. 文件的路径，设置为包含
2. 行为，设置为读取打开或者创建
3. 设置程序名称
4. 

xi. 随着对于栈的观查我们找到了非常关键的一个突破口接下来祭出 IDA，看看暴露出来的这个接口是个什么东西（笔者多嘴一句，其实逆向中成也 C++败也 C++，C++在逆向中可是自带混淆属性的，尤其是虚表多重继承这一块，如果不跑起来非常难确定函数的具体位置，但是往往 C++的名称粉碎是可以还原的这也导致我们可以对其管中窥豹，通过对于名称的命名规则来看出一些函数架构和模块,本程序就是这样被一步定位到了关键点）



xii.

xiii. 将这个关键的 dll 直接拖入 IDA 来进行分析，我们不难发现出有一个非常关键的函数 Utils::DBOperator::GetDBPassword，我的天这不是明目张胆的告诉我密码么，双击跳转过去，直接 F5 一波，我们可以看到返回值这里是一个 char 型但是结合初始化 v8 为 0，后面有地方改为 1 的过程并且直接 ret v8，所以这里应该是 bool 型

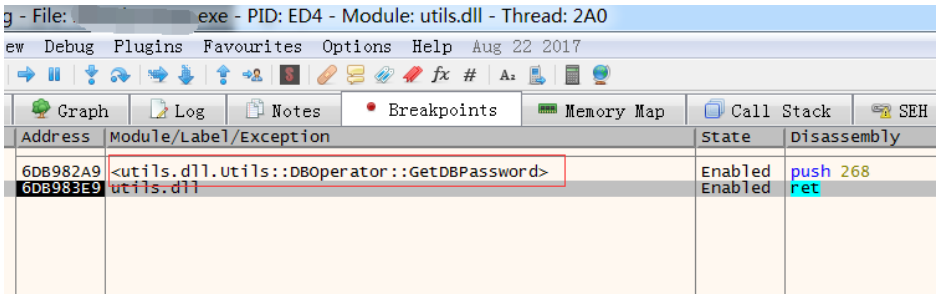
```
bool __cdecl Utils::DBOperator::GetDBPassword(int arg0, int a2)
{
  bool v2; // cf
  _BYTE *v3; // eax
  int v4; // ST04_4
  int *v5; // edx
  int v6; // eax
  bool v8; // [esp+13h] [ebp-271h]
  int v9; // [esp+14h] [ebp-270h]
  int v10; // [esp+24h] [ebp-260h]
  int v11; // [esp+30h] [ebp-254h]
  int v12; // [esp+40h] [ebp-244h]
  int v13; // [esp+4Ch] [ebp-238h]
  unsigned int v14; // [esp+60h] [ebp-224h]
  wchar_t Str; // [esp+68h] [ebp-21Ch]
  char Dst; // [esp+6Ah] [ebp-21Ah]
  char DstBuf; // [esp+268h] [ebp-1Ch]
  int v18; // [esp+280h] [ebp-4h]

  v2 = *(_DWORD *)(a2 + 20) < 0x10u;
  v8 = 0;
  *(_DWORD *)(a2 + 16) = 0;
  if ( v2 )
    v3 = (_BYTE *)a2;
  else
    v3 = *(_BYTE **)a2;
  *v3 = 0;
  Str = 0;
  memset(&Dst, 0, 0x1FEu);
  sub_1FC481B5((int)&Str);
  sub_1FC3364F((int)&v13);
  v18 = 0;
  sub_1FC480E1(&Str, v4);
  LOBYTE(v18) = 1;
  v5 = (int *)v13;
  if ( v14 < 0x10 )
    v5 = &v13;
  v6 = 0;
  while ( *(_BYTE *)v5 )
  {
    v6 = *(char *)v5 + 131 * v6;
    v5 = (int *)((char *)v5 + 1);
  }
  itoa(v6 & 0x7FFFFFFF, &DstBuf, 16);
  sub_1FC31D6D(&v9, &DstBuf);
  LOBYTE(v18) = 2;
  if ( v10 && v12 )
  {
    sub_1FC39A89((_DWORD *)a2, (int)&v9, 0, 0xFFFFFFFF);
    sub_1FC39A89((_DWORD *)a2, (int)&v11, 0, 0xFFFFFFFF);
    v8 = 1;
  }
  sub_1FC31E38(&v9, 1, 0);
  sub_1FC31E38(&v11, 1, 0);
  sub_1FC31E38(&v13, 1, 0);
  return v8;
}
```

xiv.

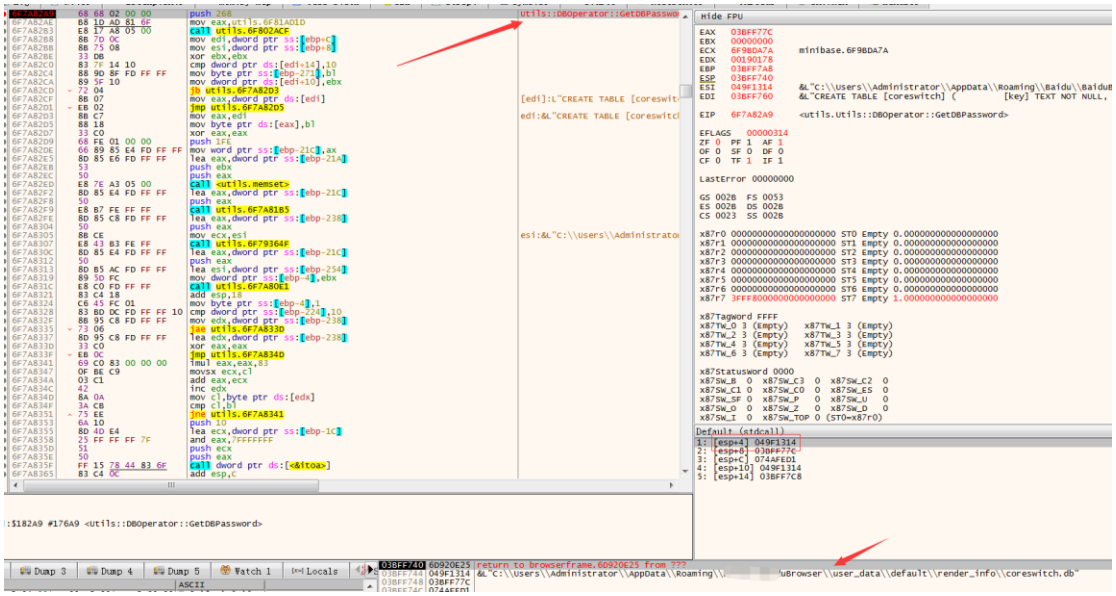
xv. 按照这个函数的思路给了两个参数，但是我们这里直接需要密码就可以了，我们完全不需要去 care 他的密码算法，两参我也不想去跟了直接 x32dbg 一波，断点就下在这个函数头

xvi.



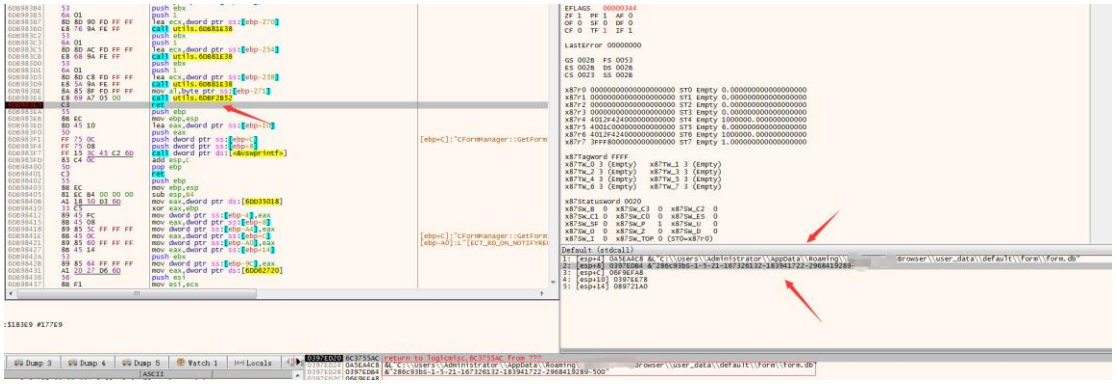
xvii. 通过刚才的栈回溯我们可以知道 CreateFile 之前调用了 GetDBPassword 这个函数，那么这也就意味着 GetDBPassword 这个函数必然是知道文件路径的，所以我猜测其中一个值为文件路径，另外一个为密码的返回值，因为只有数据库才会调用这个函数所以该函数的调用频率会大幅减小很方便我们定位。并且我会着重关注 ESP+4 和 ESP+8 这两个值，因为只有这两个值传参，在第一次调用完成之后很清晰的可以看到 ESP+4 为文件路径

xviii.

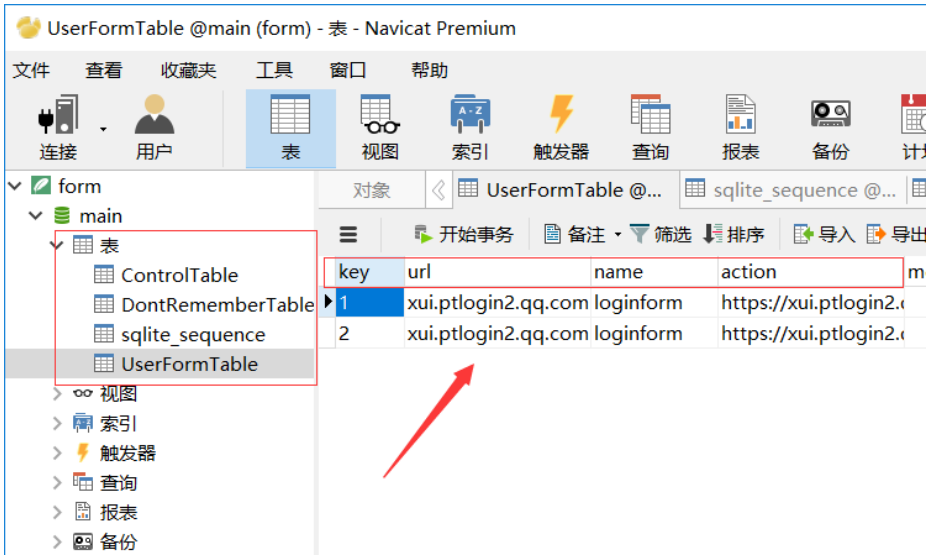


xix. 那么既然返回值不是密码只有第二个参数会是密码，我们 F8，直到我们的文件地址作为参数被传入这个函数，直接找到函数的返回地址下断，F9，这个时候栈中就已经出现了第二个字符串，也就是我们怀疑的密码

xx.

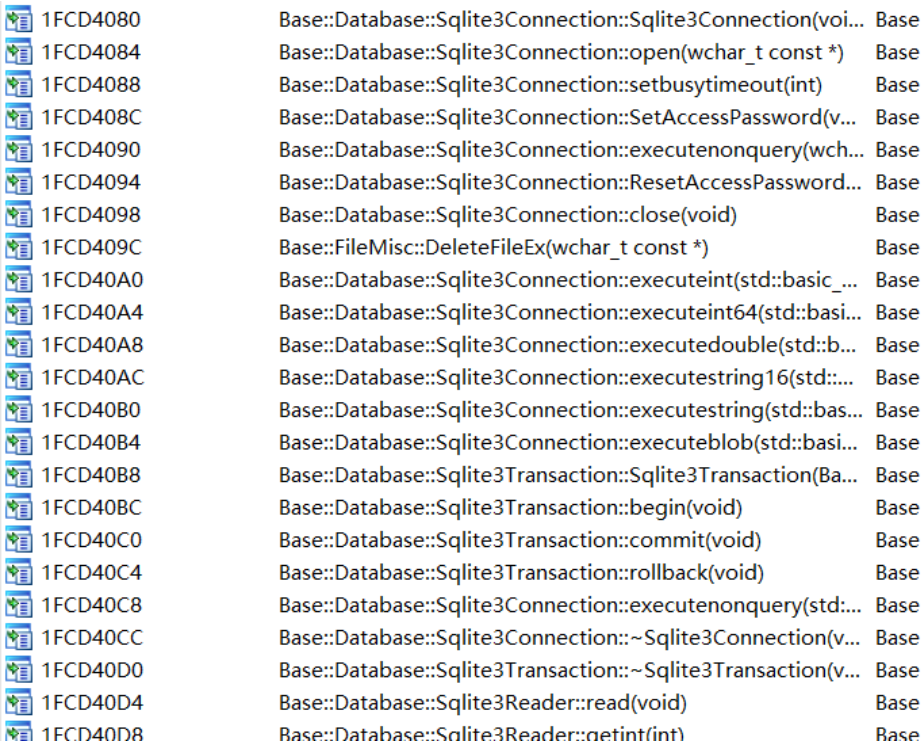


xxi. 是不是我们要的密码呢，祭出 Navicat，一波推翻对方老家，GG！拿到了我们想要的东西



xxii.

xxiii. 至于笔者为什么推断是 sqlite3 呢，你们呀看看这个调用的 dll，随便看一下导入表面都写了一堆 sqlite3 执行函数好么！！！ 明目张胆的 C++名称啊 还 DataBase



xxiv.

xxv. 通过对于这些密码的读取，JC 叔叔成功分析出了再犯罪嫌疑人邮箱中关于接头的一些信息，然后然后就没有然后了！

4. 总结

a) 本文的核心关键不在于分析算法，而是和大家探讨一些思路如何快速的去定位这些点以及如何善用正确的工具去帮助我们快速高效的分析一些关键点，在逆向中从来没有谁说自己就是死板的去使用一些工具，去走一些正统的路线，工欲善其事必先利其器，但是刀磨好了不会使往往比不会磨刀更可怕，希望通过自己的一点点思路来给新手们一些启发，老鸟请飘过无视我！

b) 最后关于多进程的问题，其实多进程和远程注入代码调试是一个问题，很多朋友包

括自己在最开始学习逆向调试的时候经常需要涉及到进程的注入啊，以及线程的注入代码，但是这个代码注入怎么调试？又怎么调试被调试进程创建的进程呢，其实很简单，无论是创建进程还是线程，都有一个属性，直接在那个属性中设置 suspend 即可，等到自己的调试 attach 上去之后下好断点再 resume 即可调试注入的代码或者被调试进程创建的进程

c) 笔者曾在去年的这个时候吃饱了撑着了将国内的主流浏览器逆了个遍，在这里做一个归类总结，作为抛砖引玉，国内的浏览器数据加密大概分有四种，1、裸奔型（chrome 为代表） 2、sqlite3 接口加密型（以今天这种浏览器为代表）3、自写 sqlite3 混合加密型（搜狗为代表，这个是最难破，非常麻烦！）4、驱动保护型（以猎豹为代表），裸奔型就不说了，sqlite3 接口加密型和 sqlite3 自写混合加密型的区别在于前者的加密算法是公用的，后者直接重写了加密接口对数据库文件的格式进行了变化，最后谢谢大家看我啰嗦到现在！不说了搬砖了！！！！

d) 最后最后最后，这个文章只是提出一些逆向中的思路和奇技淫巧，并不只是针对浏览器。毕竟有些浏览器没有做到一机一密是一个通杀密码，希望大家不要吐槽这个最关键的槽点，这真的只是我拿出来开刀的！因为谁让他稳定 BSOD 的 23333333！