来源：https://xz.aliyun.com/news/18006
文章ID：18006

# 一次恶意挖矿样本分析到捕获矿池地址-先知社区

## 样本指纹

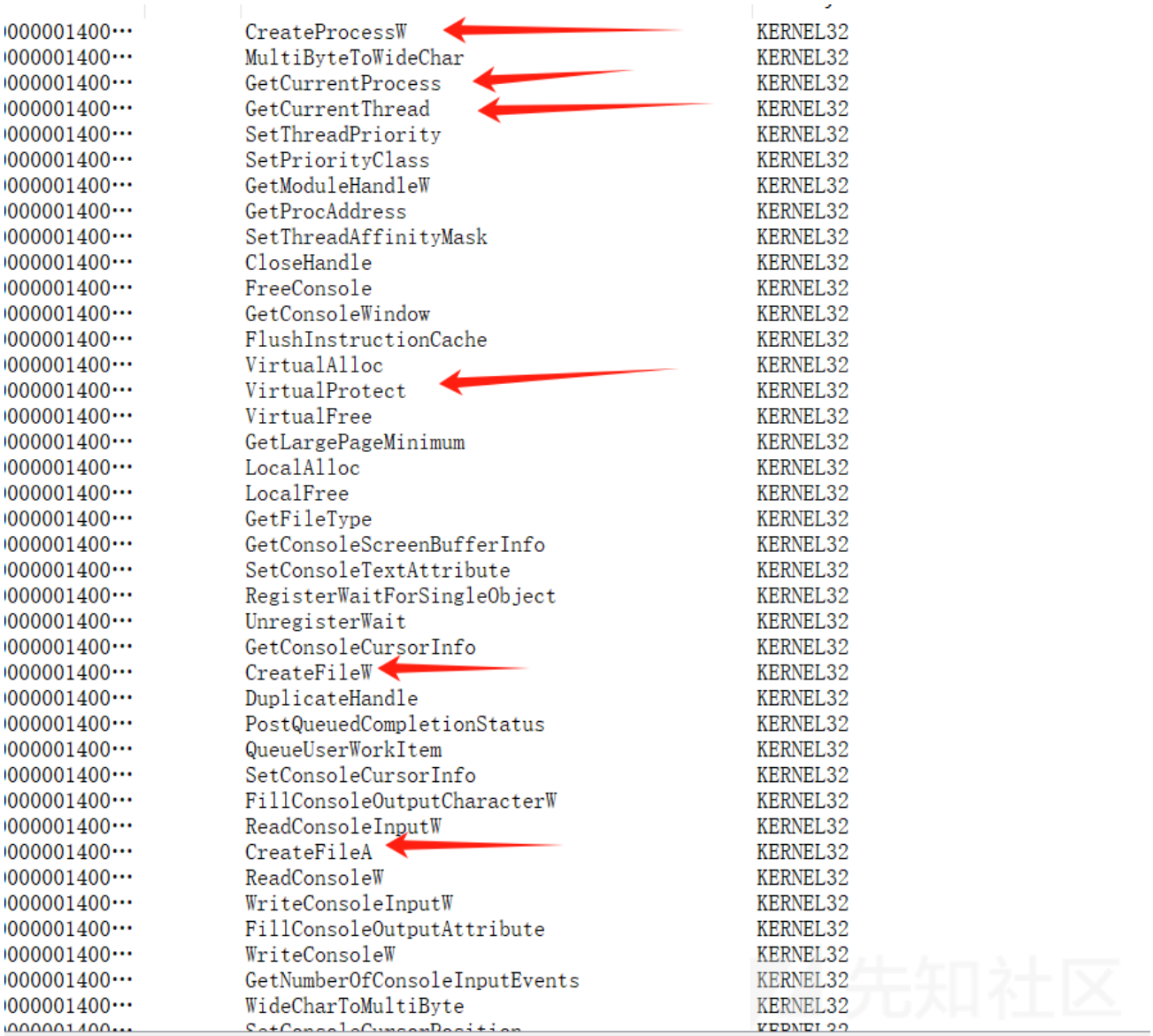
SHA256:938c187c0c566d3ecd0ca594d027cff745688b409d6ab18e7d836d9ef1bd30de

MD5:8bb9f094a5c3e8358d931200092e3412

SHA1:fced1103ababf08ea6435f43f597240ac6c357e8

## 动静分析

首先使用IDA看看导入表 图中指出了一部分敏感的API



如图还有进行一些网络连接操作

| | | | |
|---|---|---|---|
| 00000001400… | 111 | WSAGetLastError | WS2_32 |
| 00000001400… | 112 | WSASetLastError | WS2_32 |
| 00000001400… | 115 | WSAStartup | WS2_32 |
| 00000001400… | 18 | select | WS2_32 |
| 00000001400… | | WSARecvFrom | WS2_32 |
| 00000001400… | 2 | bind | WS2_32 |
| 00000001400… | | WSAIoctl | WS2_32 |
| 00000001400… | 3 | closesocket | WS2_32 |
| 00000001400… | | WSASend | WS2_32 |
| 00000001400… | 22 | shutdown | WS2_32 |
| 00000001400… | | WSASocketW | WS2_32 |
| 00000001400… | 8 | htonl | WS2_32 |
| 00000001400… | | GetAddrInfoW | WS2_32 |
| 00000001400… | | FreeAddrInfoW | WS2_32 |
| 00000001400… | 21 | setsockopt | WS2_32 |
| 00000001400… | 10 | ioctlsocket | WS2_32 |
| 00000001400… | 7 | getsockopt | WS2_32 |
| 00000001400… | | WSARecv | WS2_32 |
| 00000001400… | 23 | socket | WS2_32 |
| 00000001400… | 9 | htons | WS2_32 |

然后看执行流程，Tab键简单看看伪代码

main函数开头的这段代码，通过CreateMutex创建一个互斥体，通过GetLastError判断互斥体是否已经存在，如果已存在则进行sleep，然后程序就返回了，这样避免进程重复执行该程序

```
CreateMutexW(0i64, 1, L"sfdkjjhgkdsfhgjksd");
if ( GetLastError() == 183 )
{
  v5 = rand();
  Sleep(1000 * (v5 % 10000));
  return 0;
}
```

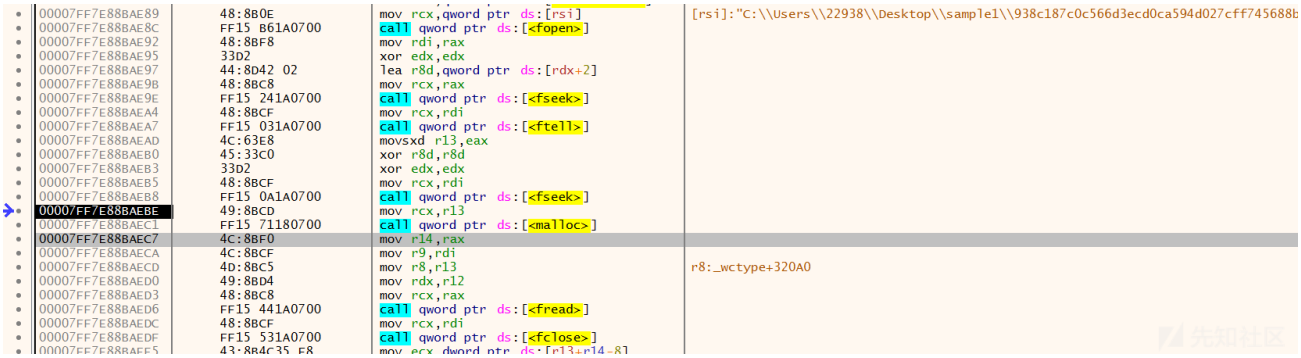
然后开始读取文件操作

```
CreateMutexW(0i64, 1, L"sfdkjjhgkdsfhgjksd");
if ( GetLastError() == 183 )
{
  v5 = rand();
  Sleep(1000 * (v5 % 10000));
  return 0;
}
else
{
  v7 = time64(0i64);
  srand(v7);
  v8 = fopen(*argv, "rb");
  fseek(v8, 0, 2);
  v9 = ftell(v8);
  fseek(v8, 0, 0);
  v10 = (char *)malloc(v9);
  fread(v10, 1ui64, v9, v8);
  fclose(v8);
  v11 = *(_DWORD *)&v10[v9 - 8];
  v21 = *(_DWORD *)&v10[v9 - 4];
  v12 = v11 - 1;
  LODWORD(Buffer[0]) = v12;
  v20 = 0;
```

这里动态调试看一下读取的是哪一个文件，调试之前需要了解一点前置知识。文件名是作为fopen的第一个参数传递的，这是一个x64位程序，函数的第一个参数第二个参数分别放在RCX、RDX寄存器中，fopen有两个参数，第一个

参数是文件路径，那么我们就动态调试看看RCX，如图说明读取的是样本自身，那么有可能真正的二阶段恶意文件就隐藏在样本自身当中（一开始我以为是CS木马就当CS马来分析了）

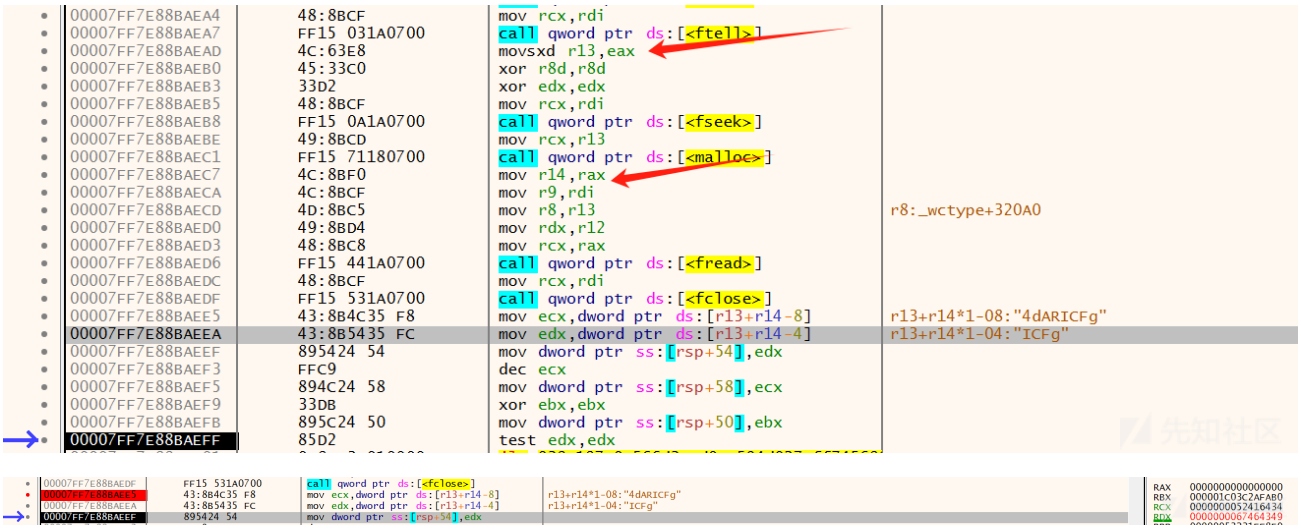

然后使用两个fseek和malloc、ftell将整个文件内容读取到了内存中，第一个fseek是获取文件末尾指针，ftell是获取文件大小，malloc是申请内存并写入内容，因此推测是将文件内容读取到内存中



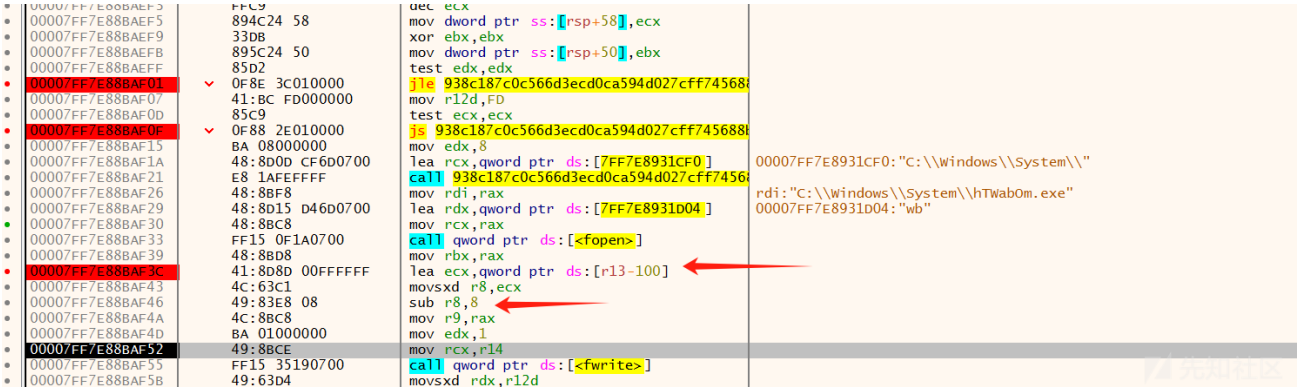

fclose之后初始化了几个变量

```
fclose(v8);
v11 = *(_DWORD *)&v10[v9 - 8];
v21 = *(_DWORD *)&v10[v9 - 4];
v12 = v11 - 1;
LODWORD(Buffer[0]) = v12;
v20 = 0;
if ( v21 > 0 )
{
  v13 = 253;
  while ( v12 >= 0 )
```

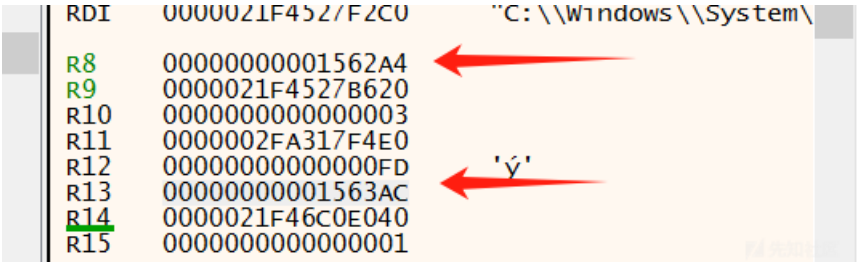
从汇编中可以看出，v11从内存中读取文件末尾倒数第8字节的DWORD值，v21读取文件末尾倒数第4字节的DWORD值，因为R13是ftell的返回值即文件大小，是R14是malloc的发挥着即内存起始地址、或者说就是文件内容的起始地址，毕竟已经把内容写入malloc




然后又开始fwrite，大小的参数值存放在R8寄存器中，根据汇编可以看出是R13-256-8

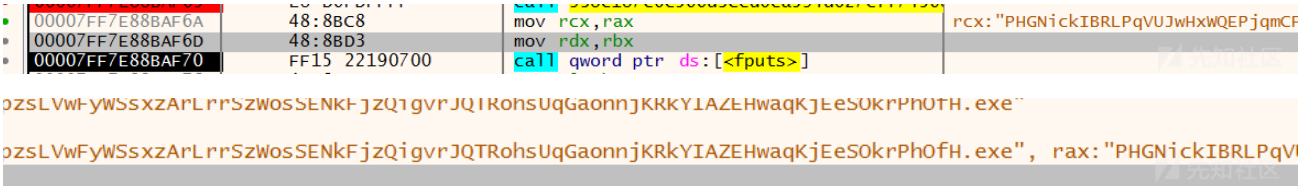



看看右边的寄存器，推测正确

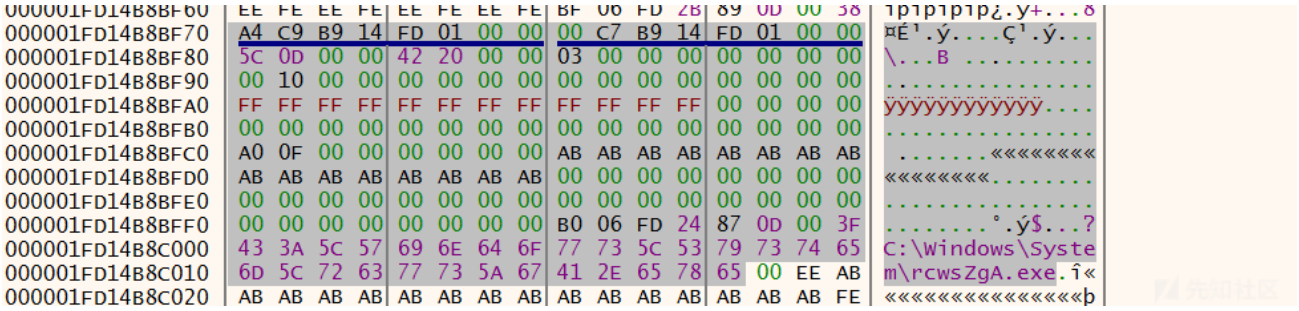



后来也经过了一次fputs和fwrite来复制一个完整的exe，但是关键代码、功能是跟样本是一样的

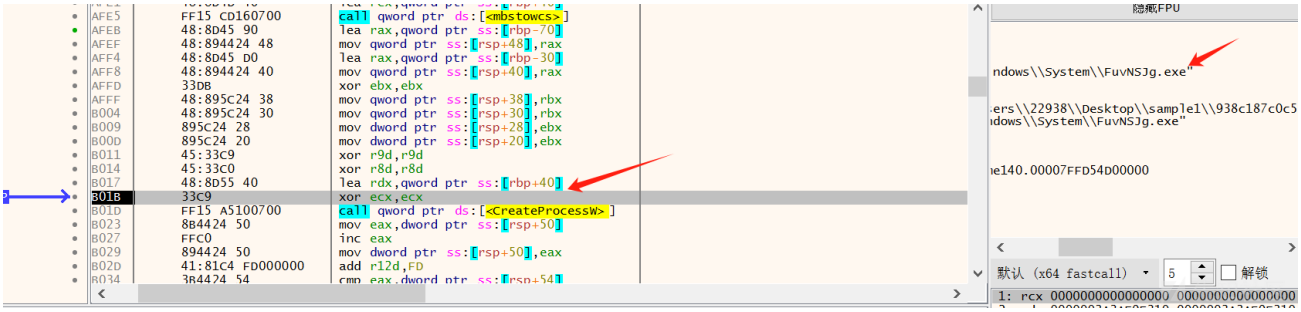
这里将RCX中的内容通过fputs写入文件 RCX就是一个很长的随机文件名 不知道为啥要这样

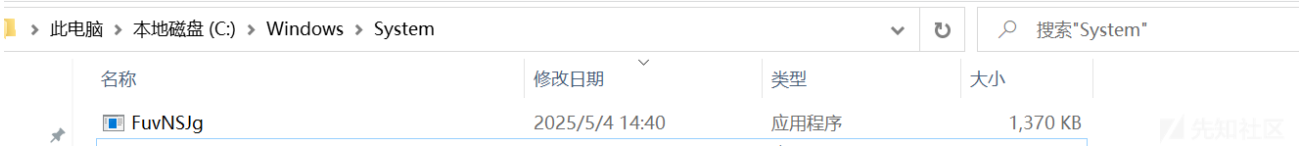



似乎是把这个文件名字给覆盖 跟了之后发现又没有变化

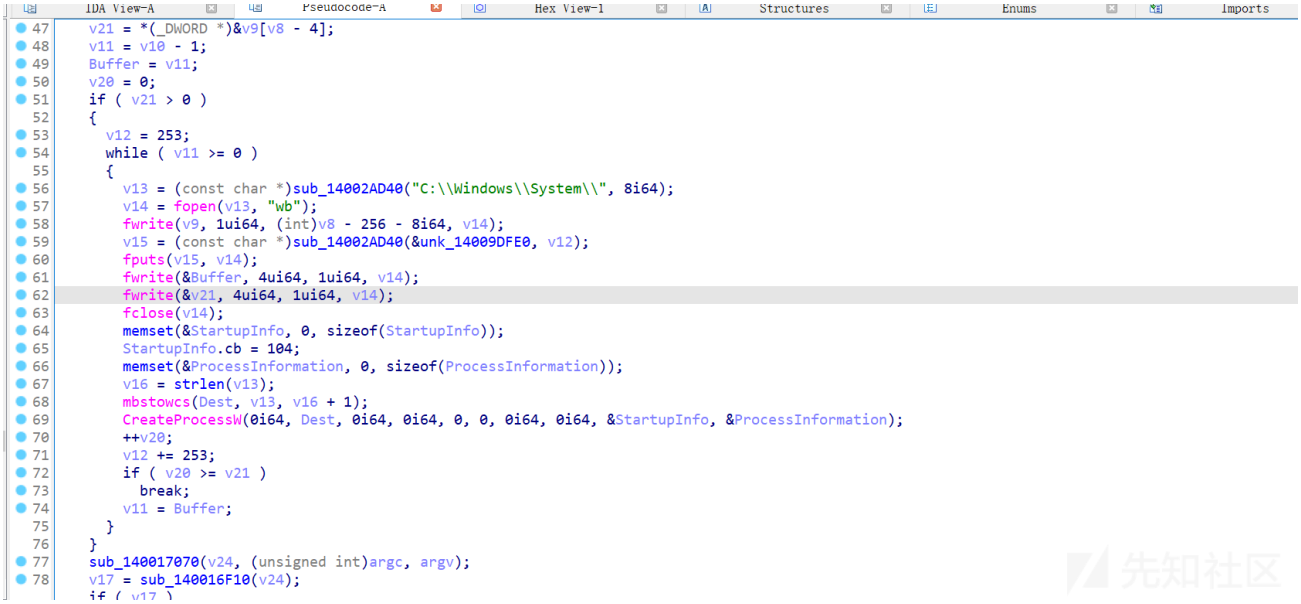



这里通过createprocess来执行exe





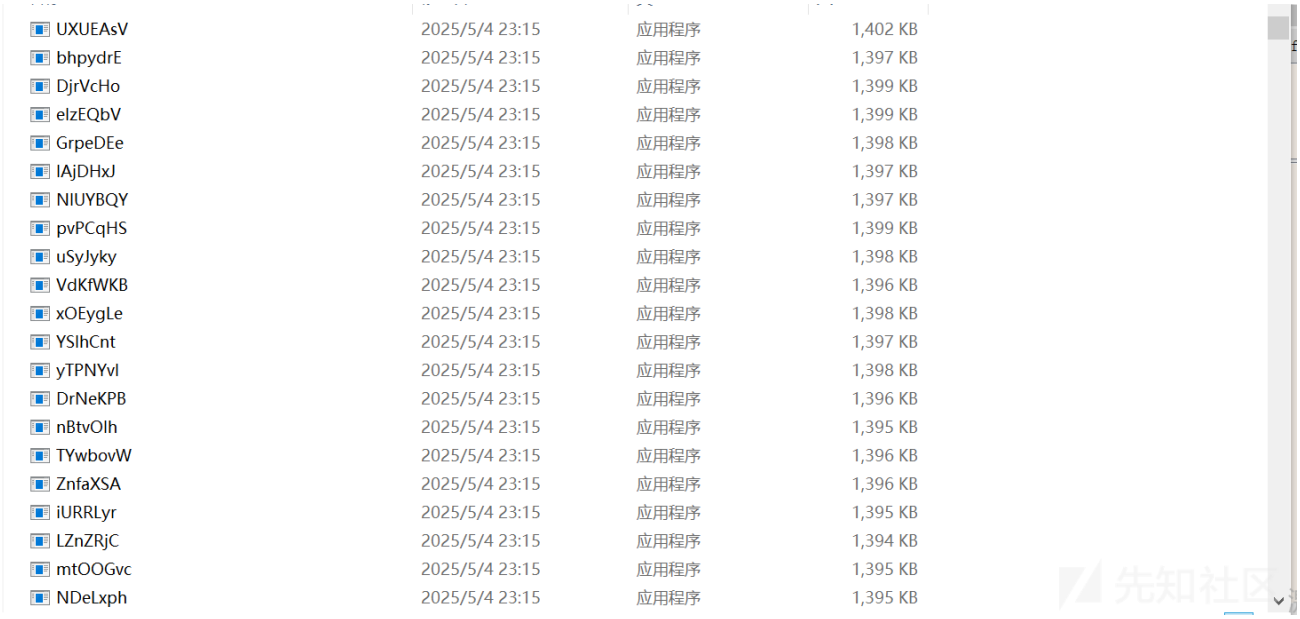

生成的exe跟一开始的样本代码是一样的但是他会循环不断生成进行自复制到C:\Windows\System\u4e2d




这里程序一直while循环进行自我复制，复制了很多次，如果绕过这个重复的过程呢？

```
v13 = 253;
while ( v12 >= 0 )
{
```

直接在复制代码区域外面下断点然后直接运行到断点位置，这是一种不过这样的话他还是会复制很多次



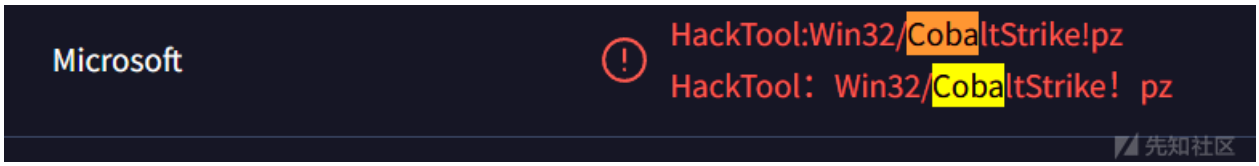

这里选择修改寄存器的方式 如图代码是通过js命令来判断次数的 判断的结果会返回给SF寄存器，如果是0则继续循环，这里直接鼠标双击设置为1就跳过循环只会进行自我复制一次了

后面继续看伪代码发现了xmrig，xmrig是知名的开源矿工程序，常被恶意软件用于隐蔽挖矿

```
_QWORD *__fastcall sub_140026B88(_QWORD *a1, __int64 a2)
{
  __int64 v4; // rbx
  void *v5; // rax
  __int64 v6; // rcx
  void *v7; // rax

  *a1 = &xmrig::App::`vftable';
  a1[1] = &xmrig::App::`vftable';
  v4 = 0i64;
  a1[2] = 0i64;
  a1[4] = 0i64;
  v5 = operator new(0x20ui64);
  if ( v5 )
    v6 = sub_1400274D0(v5, a2);
  else
    v6 = 0i64;
  a1[3] = v6;
  if ( !(*(unsigned int (__fastcall **)(__int64))(*(_QWORD *)v6 + 24i64))(v6) && !*(_BYTE *)(sub_1400149C8(a1[3]) + 9) )
  {
    v7 = operator new(0x18ui64);
    if ( v7 )
      v4 = sub_14001059C(v7, a1);
    a1[2] = v4;
  }
  return a1;
}
```

这里才发现，这实际上不是CS木马其实就是挖矿程序，只有微软报是CS木马就一直当CS木马来看了



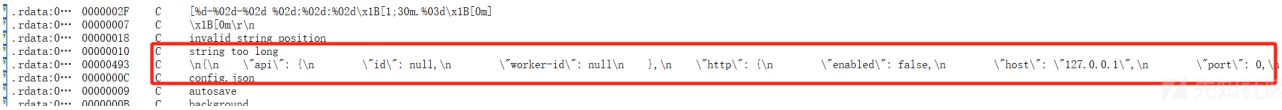

| | | | |
|---|---|---|---|
| al 快速修复 | Trojan.Miner.S28484635 木马矿工 S284... | Rising 上升 | Trojan.CoinMiner!1.C2B5 (CLASSIC)<br>Trojan.CoinMiner！1.C2B5 （经典版） |
| Engine Zero 深信服零号引擎 | Trojan.Win32.Save.a 木马.Win32.Save.a | SecureAge 安全时代 | Malicious 恶意 |
| One (Static ML) SentinelOne ML) | Static AI - Malicious PE<br>静态 AI - 恶意 PE | Skyhigh (SWG) 天高 （SWG） | BehavesLike.Win64.Dropper.th |
| Sophos 公司 | Mal/VB-AQW | SUPERAntiSpyware SUPERAntiSpyware 软件 | Hack.Tool/Gen-BitCoinMiner |
| ec 赛门铁克 | Packed.Generic.696 | Tencent 腾讯 | Trojan.Win64.CoinMiner.b 木马.Win64.C... |
| e 陷阱矿 | Suspicious.low.ml.score 可疑的 .low.ml... | Trellix (ENS) 格子 （ENS） | Dropper-FXA!8BB9F094A5C3 滴管-FXA... |
| cro 趋势科技 | Coinminer.Win64.TOOLXMR.SMA | TrendMicro-HouseCall TrendMicro-HouseCall 公司 | Coinminer.Win64.TOOLXMR.SMA |
| 里斯特 | W64/Coinminer.GT | VBA32 VBA32 的 | Trojan.BtcMine |
| 普雷 | Trojan.GenericKD.69370767<br>木马 GenericKD.69370767 | VirIT 维尔特 | Trojan.Win64.Genus.CHIY 木马 Win64.G... |
| Webroot 网站 | W32.Coinminer.Xmrig W32.币矿机.Xmrig | WithSecure WithSecure 安全 | Heuristic.HEUR/AGEN.1320164<br>启发式 HEUR/AGEN.1320164 |
| 燕子 | Trojan.Mansabo!XUBuF46rW+0<br>特洛伊木马 Mansabo！XUBuF46rW+0 | Zillya 齐利亚 | Trojan.CoinMiner.Win64.15235<br>木马.CoinMiner.Win64.15235 |

这里defender又识别是挖矿木马


# 捕获矿池地址

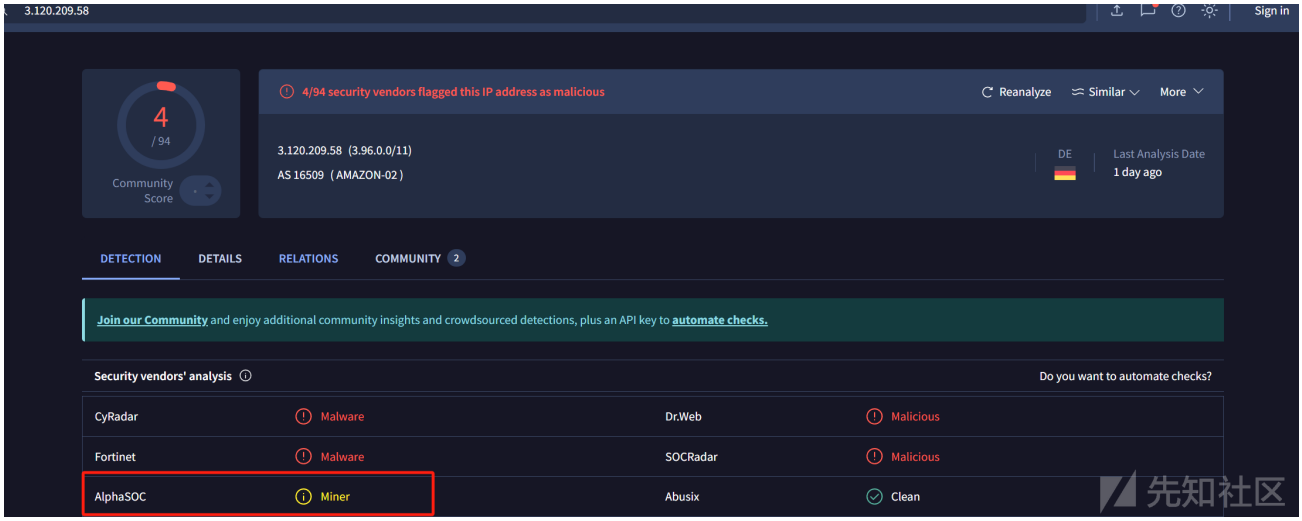
确定是挖矿木马了，那就尝试找一下矿池地址吧 最简单的方式是运行然后看看wireshark，当然这存在一定风险

也可以直接ida看看字符串有没有相关的信息，如图，复制下来看看 很明显是挖矿程序的一些配置信息。其中"algo": "cn/r"：使用CryptoNight算法变种（如CryptoNightR），常用于门罗币

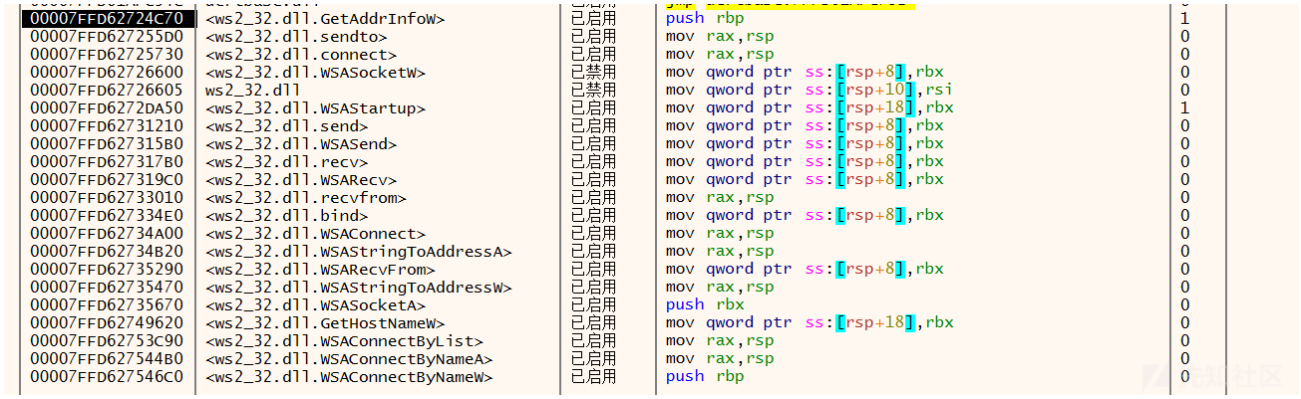


```
.rdata:000000014009DA30 00000493 C
{
    "api": {
        "id": null,
        "worker-id": null
    },
    "http": {
        "enabled": false,
        "host": "127.0.0.1",
        "port": 0,
        "access-token": null,
        "restricted": true
    },
    "autosave": false,
    "version": 1,
    "background": false,
    "colors": true,
    "randomx": {
        "init": -1,
        "numa": true
    },
    "cpu": {
        "enabled": true,
        "huge-pages": true,
        "hw-aes": null,
        "priority": null,
        "asm": true,
        "argon2-impl": null,
        "cn/0": false,
        "cn-lite/0": false
    },
    "donate-level": 0,
    "donate-over-proxy": 1,
    "log-file": null,
    "pools": [
        {
            "algo": "cn/r",
            "url": "3.120.209.58:8080",
            "user": "x",
            "pass": "x",
            "rig-id": null,
            "nicehash": false,
            "keepalive": false.............
```

有个url：3.120.209.58:8080，pools一般就是矿池相关的参数了 微步是显示安全的 放在VT看看 有一个显示Miner也就是矿池的意思

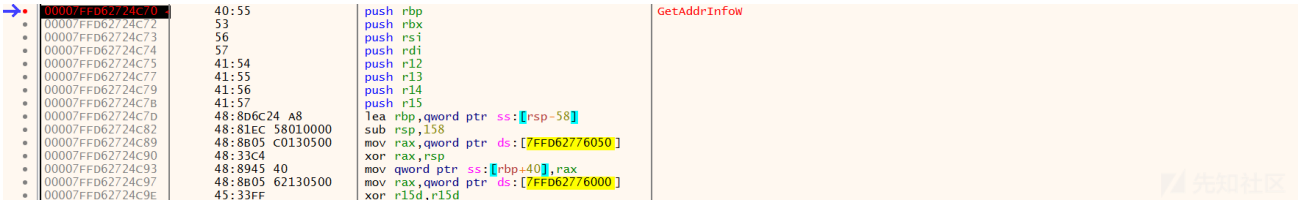



除了看字符串以外呢，我们还可以通过xdbg动态调试的方式获取矿池地址，那么我们就需要在一些进行网络连接相关的API上进行断点，然后去分析他的参数传入顺序、以及参数值，从而捕获矿池的地址，主要是找ws2_32.dll里的API，这里面的API大多都是跟网络连接相关的，可以尝试对这些API进行断点




```
<ws2_32.dll.GetAddrInfoW>
<ws2_32.dll.sendto>
<ws2_32.dll.connect>
<ws2_32.dll.WSASocketW>
ws2_32.dll
<ws2_32.dll.WSAStartup>
<ws2_32.dll.send>
<ws2_32.dll.WSASend>
<ws2_32.dll.recv>
<ws2_32.dll.WSARecv>
<ws2_32.dll.recvfrom>
<ws2_32.dll.bind>
<ws2_32.dll.WSAConnect>
<ws2_32.dll.WSAStringToAddressA>
<ws2_32.dll.WSARecvFrom>
<ws2_32.dll.WSAStringToAddressW>
<ws2_32.dll.WSASocketA>
<ws2_32.dll.GetHostNameW>
<ws2_32.dll.WSAConnectByList>
<ws2_32.dll.WSAConnectByNameA>
<ws2_32.dll.WSAConnectByNameW>
```

这里开始进行动态调试，这里就进入了GetAddrInfow这个API




就来学习一下这个API 如图官方解释

**GetAddrInfoW** 函数提供从 Unicode 主机名到地址的与协议无关的转换。

# 语法

C++ 复制

```
INT WSAAPI GetAddrInfoW(
  [in, optional] PCWSTR           pNodeName,
  [in, optional] PCWSTR           pServiceName,
  [in, optional] const ADDRINFOW *pHints,
  [out]          PADDRINFOW      *ppResult
);
```

# 参数

`[in, optional] pNodeName`

指向 **以 NULL** 结尾的 Unicode 字符串的指针，该字符串包含主机 (节点) 名称或数字主机地址字符串。 对于 Internet 协议，数字主机地址字符串是点十进制 IPv4 地址或 IPv6 十六进制地址。

`[in, optional] pServiceName`

指向以 **NULL** 结尾的 Unicode 字符串的指针，该字符串包含表示为字符串的服务名称或端口号。

服务名称是端口号的字符串别名。 例如，“http”是由 Internet 工程任务组定义的端口 80 的别名， (IETF) 作为 Web 服务器用于 HTTP 协议的默认端口。 以下文件中列出了未指定端口号时 *pServiceName* 参数的可能值：

```
INT WSAAPI GetAddrInfoW(
  [in, optional] PCWSTR           pNodeName,
  [in, optional] PCWSTR           pServiceName,
  [in, optional] const ADDRINFOW *pHints,
  [out]          PADDRINFOW      *ppResult
);
```

那么第一个参数也就是矿池的IP或者域名了，根据X64的调用约定可知，第一个参数是放在RCX里的，那么我们就可以确定了，3.120.209.58:8080就是矿池地址