

## 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:fcdd1103ababf08ea6435f43f597240ac6c357e8

### 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

000001400...	CreateProcessW	←	KERNEL32
000001400...	MultiByteToWideChar		KERNEL32
000001400...	GetCurrentProcess	←	KERNEL32
000001400...	GetCurrentThread	←	KERNEL32
000001400...	SetThreadPriority		KERNEL32
000001400...	SetPriorityClass		KERNEL32
000001400...	GetModuleHandleW		KERNEL32
000001400...	GetProcAddress		KERNEL32
000001400...	SetThreadAffinityMask		KERNEL32
000001400...	CloseHandle		KERNEL32
000001400...	FreeConsole		KERNEL32
000001400...	GetConsoleWindow		KERNEL32
000001400...	FlushInstructionCache		KERNEL32
000001400...	VirtualAlloc		KERNEL32
000001400...	VirtualProtect	←	KERNEL32
000001400...	VirtualFree		KERNEL32
000001400...	GetLargePageMinimum		KERNEL32
000001400...	LocalAlloc		KERNEL32
000001400...	LocalFree		KERNEL32
000001400...	GetFileType		KERNEL32
000001400...	GetConsoleScreenBufferInfo		KERNEL32
000001400...	SetConsoleTextAttribute		KERNEL32
000001400...	RegisterWaitForSingleObject		KERNEL32
000001400...	UnregisterWait		KERNEL32
000001400...	GetConsoleCursorInfo		KERNEL32
000001400...	CreateFileW	←	KERNEL32
000001400...	DuplicateHandle		KERNEL32
000001400...	PostQueuedCompletionStatus		KERNEL32
000001400...	QueueUserWorkItem		KERNEL32
000001400...	SetConsoleCursorInfo		KERNEL32
000001400...	FillConsoleOutputCharacterW		KERNEL32
000001400...	ReadConsoleInputW		KERNEL32
000001400...	CreateFileA	←	KERNEL32
000001400...	ReadConsoleW		KERNEL32
000001400...	WriteConsoleInputW		KERNEL32
000001400...	FillConsoleOutputAttribute		KERNEL32
000001400...	WriteConsoleW		KERNEL32
000001400...	GetNumberOfConsoleInputEvents		KERNEL32
000001400...	WideCharToMultiByte		KERNEL32
000001400...	SetConsoleCursorPosition		KERNEL32

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

先知社区

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"sfdkjjhgkdsfhgjkds");
if ( GetLastError() == 183 )
{
    v5 = rand();
    Sleep(1000 * (v5 % 10000));
    return 0;
}
```

先知社区

然后开始读取文件操作

```
CreateMutexW(0i64, 1, L"sfdkjjhgkdsfhgjkds");
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;
}
```

先知社区

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）

00007FF7E88BAE82	48:8D15 5F6E0700	lea rcx,qword ptr ds:[7FF7E8931CE8]	rdx:_wctype+320A0, 00007FF7E8931CE8:"rb"
00007FF7E88BAE89	48:880E	mov rcx,qword ptr ds:[rsi]	[rsi]:C:\Users\...\Desktop\sampl...938c187c0c566d3ecd0ca594d027cff745688b409fca618e7d836d99ef
00007FF7E88BAE8C	FF15 B61A0700	call qword ptr ds:[cfopenx]	

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

00007FF7E88BAE89	48:880E	mov rcx,qword ptr ds:[rsi]	[rsi]:C:\Users\22938\Desktop\sampl...938c187c0c566d3ecd0ca594d027cff745688b
00007FF7E88BAE8C	FF15 B61A0700	call qword ptr ds:[cfopenx]	
00007FF7E88BAE92	48:8BF8	mov rdi,rax	
00007FF7E88BAE95	33D2	xor edx,edx	
00007FF7E88BAE97	44:8D42 02	lea r8d,qword ptr ds:[rdx+2]	
00007FF7E88BAE9B	48:8BC8	mov rcx,rax	
00007FF7E88BAE9E	FF15 241A0700	call qword ptr ds:[cfseekx]	
00007FF7E88BAEA4	48:8BCF	mov rcx,rdi	
00007FF7E88BAEA7	FF15 031A0700	call qword ptr ds:[cfte11x]	
00007FF7E88BAEAD	4C:63E8	movsxd r13,eax	
00007FF7E88BAEB0	45:33C0	xor r8d,r8d	
00007FF7E88BAEB3	33D2	xor edx,edx	
00007FF7E88BAEB5	48:8BCF	mov rcx,rdi	
00007FF7E88BAEB8	FF15 0A1A0700	call qword ptr ds:[cfseekx]	
00007FF7E88BAEBE	49:8BCD	mov rcx,r13	
00007FF7E88BAEC1	FF15 71180700	call qword ptr ds:[mallocx]	
00007FF7E88BAEC7	4C:8BF0	mov r14,rax	
00007FF7E88BAECA	4C:8BCF	mov r9,rdi	
00007FF7E88BAECD	4D:8BC5	mov r8,r13	
00007FF7E88BAED0	49:8BD4	mov rdx,r12	r8:_wctype+320A0
00007FF7E88BAED3	48:8BC8	mov rcx,rax	
00007FF7E88BAED6	FF15 441A0700	call qword ptr ds:[cfreadx]	
00007FF7E88BAEDC	48:8BCF	mov rcx,rdi	
00007FF7E88BAEDF	FF15 531A0700	call qword ptr ds:[cfclosex]	
00007FF7E88BAEF5	43:8B4C 35 F8	mov ecx,dword ptr ds:[r13+r14-8]	

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的返回值即内存起始地址、或者说就是文件内容的起始地址，毕竟已经把内容写入malloc

00007FF7E88BAE4	48:8BCF	mov rcx,rdi	
00007FF7E88BAE7	FF15 031A0700	call qword ptr ds:[cfte11x]	
00007FF7E88BAEAD	4C:63E8	movsxd r13,eax	
00007FF7E88BAEB0	45:33C0	xor r8d,r8d	
00007FF7E88BAEB3	33D2	xor edx,edx	
00007FF7E88BAEB5	48:8BCF	mov rcx,rdi	
00007FF7E88BAEB8	FF15 0A1A0700	call qword ptr ds:[cfseekx]	
00007FF7E88BAEBE	49:8BCD	mov rcx,r13	
00007FF7E88BAEC1	FF15 71180700	call qword ptr ds:[mallocx]	
00007FF7E88BAEC7	4C:8BF0	mov r14,rax	
00007FF7E88BAECA	4C:8BCF	mov r9,rdi	
00007FF7E88BAECD	4D:8BC5	mov r8,r13	
00007FF7E88BAED0	49:8BD4	mov rdx,r12	r8:_wctype+320A0
00007FF7E88BAED3	48:8BC8	mov rcx,rax	
00007FF7E88BAED6	FF15 441A0700	call qword ptr ds:[cfreadx]	
00007FF7E88BAEDC	48:8BCF	mov rcx,rdi	
00007FF7E88BAEDF	FF15 531A0700	call qword ptr ds:[cfclosex]	
00007FF7E88BAEE5	43:8B4C 35 F8	mov ecx,dword ptr ds:[r13+r14-8]	r13+r14*1-08:"4dARICFg"
00007FF7E88BAEEA	43:8B54 35 FC	mov edx,dword ptr ds:[r13+r14-4]	r13+r14*1-04:"ICFg"
00007FF7E88BAEEF	895424 54	mov dword ptr ss:[rsp+54],edx	
00007FF7E88BAEF3	FFC9	dec ecx	
00007FF7E88BAEF5	894C24 58	mov dword ptr ss:[rsp+58],ecx	
00007FF7E88BAEF9	33DB	xor ebx,ebx	
00007FF7E88BAEFB	895C24 50	mov dword ptr ss:[rsp+50],ebx	
00007FF7E88BAEFF	85D2	test edx,edx	

00007FF7E88BAEDF	FF15 531A0700	call qword ptr ds:[cfclosex]	
00007FF7E88BAEE5	43:8B4C 35 F8	mov ecx,dword ptr ds:[r13+r14-8]	r13+r14*1-08:"4dARICFg"
00007FF7E88BAEEA	43:8B54 35 FC	mov edx,dword ptr ds:[r13+r14-4]	r13+r14*1-04:"ICFg"
00007FF7E88BAEEF	895424 54	mov dword ptr ss:[rsp+54],edx	

RAX	0000000000000000
RBX	000001C03C2AFAB0
RCX	0000000052416434
RDY	0000000000000000
REX	0000000000000000

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



此电脑 > 本地磁盘 (C:) > Windows > System				搜索"System"
名称	修改日期	类型	大小	
FuvNSJg	2025/5/4 14:40	应用程序	1,370 KB	

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

```

47 v21 = *(_DWORD *)v9[v8 - 4];
48 v11 = v10 - 1;
49 Buffer = v11;
50 v20 = 0;
51 if ( v21 > 0 )
52 {
53     v12 = 253;
54     while ( v11 >= 0 )
55     {
56         v13 = (const char *)sub_14002AD40("C:\\Windows\\System\\", 8i64);
57         v14 = fopen(v13, "wb");
58         fwrite(v9, 1ui64, (int)v8 - 256 - 8i64, v14);
59         v15 = (const char *)sub_14002AD40(&unk_14009DFE0, v12);
60         fputs(v15, v14);
61         fwrite(&Buffer, 4ui64, 1ui64, v14);
62         fwrite(&v21, 4ui64, 1ui64, v14);
63         fclose(v14);
64         memset(&StartupInfo, 0, sizeof(StartupInfo));
65         StartupInfo.cb = 104;
66         memset(&ProcessInformation, 0, sizeof(ProcessInformation));
67         v16 = strlen(v13);
68         mbstowcs(Dest, v13, v16 + 1);
69         CreateProcessW(0i64, Dest, 0i64, 0i64, 0, 0, 0i64, 0i64, &StartupInfo, &ProcessInformation);
70         ++v20;
71         v12 += 253;
72         if ( v20 >= v21 )
73             break;
74         v11 = Buffer;
75     }
76 }
77 sub_140017070(v24, (unsigned int)argv, argv);
78 v17 = sub_140016F10(v24);
if ( v17 )

```

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 )
{

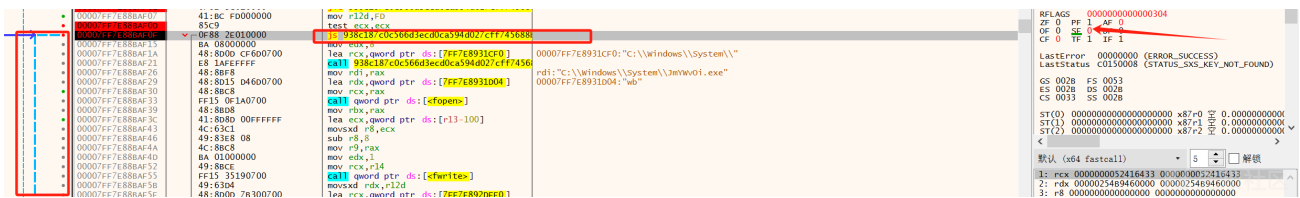
```

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

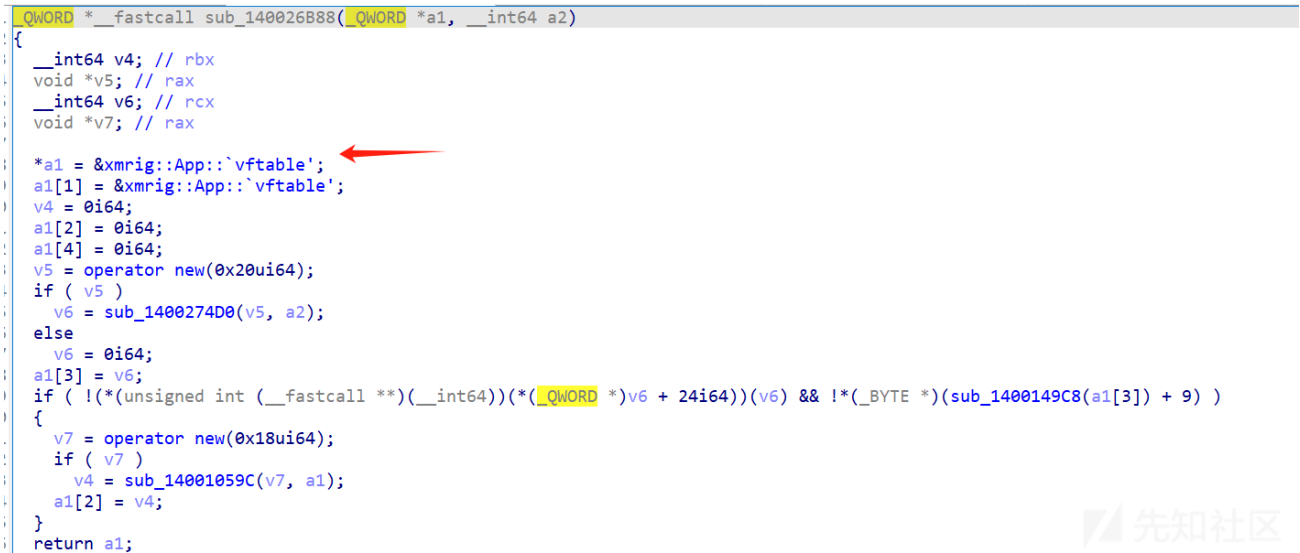
UXUEAsV	2025/5/4 23:15	应用程序	1,402 KB
bhpydrE	2025/5/4 23:15	应用程序	1,397 KB
DjrVcHo	2025/5/4 23:15	应用程序	1,399 KB
elzEqbV	2025/5/4 23:15	应用程序	1,399 KB
GrpeDEe	2025/5/4 23:15	应用程序	1,398 KB
IAjDHxJ	2025/5/4 23:15	应用程序	1,397 KB
NIUYBQY	2025/5/4 23:15	应用程序	1,397 KB
pvPCqHS	2025/5/4 23:15	应用程序	1,399 KB
uSyJyky	2025/5/4 23:15	应用程序	1,398 KB
VdKfWKB	2025/5/4 23:15	应用程序	1,396 KB
xOEygle	2025/5/4 23:15	应用程序	1,398 KB
YSIhCnt	2025/5/4 23:15	应用程序	1,397 KB
yTPNYvi	2025/5/4 23:15	应用程序	1,398 KB
DrNeKPB	2025/5/4 23:15	应用程序	1,396 KB
nBtvOIh	2025/5/4 23:15	应用程序	1,395 KB
TYwbovW	2025/5/4 23:15	应用程序	1,396 KB
ZnfaXSA	2025/5/4 23:15	应用程序	1,396 KB
iURRLyr	2025/5/4 23:15	应用程序	1,395 KB
LZnZRjC	2025/5/4 23:15	应用程序	1,394 KB
mtOOGvc	2025/5/4 23:15	应用程序	1,395 KB
NDeLxph	2025/5/4 23:15	应用程序	1,395 KB

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

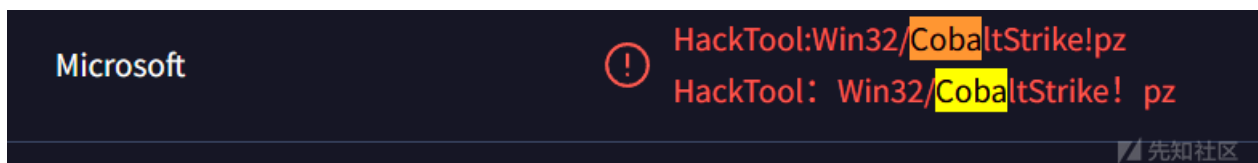




后面继续看伪代码发现了xmrig, xmrig是知名的开源矿工程序, 常被恶意软件用于隐藏挖矿



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



快速修复	⚠ Trojan.Miner.S28484635 木马矿工 S284...	Rising 上升	⚠ Trojan.CoinMiner!1.C2B5 (CLASSIC) 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 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
赛门铁克	⚠ Packed.Generic.696	Tencent 腾讯	⚠ Trojan.Win64.CoinMiner.b 木马.Win64.C...
陷阱矿	⚠ Suspicious.low.ml.score 可疑的 .low.ml...	Trellix (ENS) 格子 (ENS)	⚠ Dropper-FXA!8BB9F094A5C3 滴管-FXA...
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 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 启发式 HEUR/AGEN.1320164
燕子	⚠ Trojan.MansabotXUBuF46rW+0 特洛伊木马 Mansabot XUBuF46rW+0	Zillya 齐利亚	⚠ Trojan.CoinMiner.Win64.15235 木马 CoinMiner.Win64.15235

这里defender又识别是挖矿木马



## 发现威胁 - 需要采取措施。

严重

2025/5/4 12:28

状态: 活动

活动的威胁未得到处理, 并且仍在你的设备上运行。

已检测到威胁: Trojan:Win64/XmrighMiner.RP!MTB

警报级别: 严重

日期: 2025/5/4 12:29

类别: 特洛伊木马

详细信息: 这个程序很危险, 而且执行来自攻击者的命令。

[了解更多信息](#)

受影响的项目:

file: C:\Windows\System\akmAouj.exe

file: C:\Windows\System\ATfyjUz.exe

file: C:\Windows\System\AXeIRTo.exe



## 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:0000002F C [0x0000002F] 0x0000002F 0x0000002F 0x0000002F 0x0000002F 0x0000002F 0x0000002F 0x0000002F
.rdata:00000007 C [0x00000007] 0x00000007 0x00000007 0x00000007 0x00000007 0x00000007 0x00000007 0x00000007
.rdata:00000018 C [0x00000018] 0x00000018 0x00000018 0x00000018 0x00000018 0x00000018 0x00000018 0x00000018
.rdata:00000010 C [0x00000010] 0x00000010 0x00000010 0x00000010 0x00000010 0x00000010 0x00000010 0x00000010
.rdata:00000493 C [0x00000493] 0x00000493 0x00000493 0x00000493 0x00000493 0x00000493 0x00000493 0x00000493
.rdata:0000000C C [0x0000000C] 0x0000000C 0x0000000C 0x0000000C 0x0000000C 0x0000000C 0x0000000C 0x0000000C
.rdata:00000009 C [0x00000009] 0x00000009 0x00000009 0x00000009 0x00000009 0x00000009 0x00000009 0x00000009
.rdata:00000008 C [0x00000008] 0x00000008 0x00000008 0x00000008 0x00000008 0x00000008 0x00000008 0x00000008
```

```
.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





00007FFD62724C70	40:55	push rbp	GetAddrInfoW
00007FFD62724C72	53	push rbx	
00007FFD62724C73	56	push rsi	
00007FFD62724C74	57	push rdi	
00007FFD62724C75	41:54	push r12	
00007FFD62724C77	41:55	push r13	
00007FFD62724C79	41:56	push r14	
00007FFD62724C7B	41:57	push r15	
00007FFD62724C7D	48:806C24 A8	lea rbp,qword ptr ss:[rsp-58]	
00007FFD62724C82	48:81EC 58010000	sub rsp,158	
00007FFD62724C89	48:8805 C0130500	mov rax,qword ptr ds:[7FFD62776050]	
00007FFD62724C90	48:33C4	xor rax,rsp	
00007FFD62724C93	48:8945 40	mov qword ptr ss:[rbp+40],rax	
00007FFD62724C97	48:8805 62130500	mov rax,qword ptr ds:[7FFD62776000]	
00007FFD62724C9E	45:33FF	xor r15d,r15d	

先知社区

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 WINAPI 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时 pServiceName 参数的可能值：

```
INT WINAPI 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

00007FFD62724C70	40:55	push rbp	GetAddrInfoW
00007FFD62724C72	53	push rbx	
00007FFD62724C73	56	push rsi	
00007FFD62724C74	57	push rdi	
00007FFD62724C75	41:54	push r12	
00007FFD62724C77	41:55	push r13	
00007FFD62724C79	41:56	push r14	
00007FFD62724C7B	41:57	push r15	
00007FFD62724C7D	48:806C24 A8	lea rbp,qword ptr ss:[rsp-58]	
00007FFD62724C82	48:81EC 58010000	sub rsp,158	
00007FFD62724C89	48:8805 C0130500	mov rax,qword ptr ds:[7FFD62776050]	
00007FFD62724C90	48:33C4	xor rax,rsp	
00007FFD62724C93	48:8945 40	mov qword ptr ss:[rbp+40],rax	
00007FFD62724C97	48:8805 62130500	mov rax,qword ptr ds:[7FFD62776000]	
00007FFD62724C9E	45:33FF	xor r15d,r15d	
00007FFD62724CA1	48:88F1	mov r15d,rax	

RAX	0000000000000000
RBX	0000000000000000
RCX	0000000000000000
RDX	0000000000000000
RIP	0000007F7E8942A0
RSP	0000007F7E8942A0
RBP	0000007F7E8942A0
R01	0000000000000000
R02	0000000000000000
R03	0000000000000000
R04	0000000000000000
R05	0000000000000000
R06	0000000000000000
R07	0000000000000000
R08	0000000000000000
R09	0000000000000000
R10	0000000000000000
R11	0000000000000000
R12	0000000000000000
R13	0000000000000000
R14	0000000000000000
R15	0000000000000000