逆向随笔——对可以过TP的注入驱动的一次逆向

原创

Lixinist 于 2019-10-21 18:45:54 发布 1288 收藏 8

文章标签： Windows逆向 驱动逆向 IDA 反汇编 DLL注入

本文链接：https://blog.csdn.net/qq_15059515/article/details/102667351

前言：

逆一些东西，有点收获，就写篇随笔记录一下。因为写的随便，就不发看雪了。
今天一个朋友给我发了一个说是可以过TP的注入驱动，希望我逆一下看看是什么套路。

正文：

接收过来压缩包，看了一下

loadddll32和64分别是用在32和64的驱动，MemOpe和dnf.exe都是测试用例（不过我是用自己写的123.exe进行测试）。

先跑起来，看看线程和模块上有没有什么特别的地方

[123.exe]进程线程(2)

线程Id	ETHREAD	Teb	优先级	线程入口	模块	切换次数	线程状态
5804	0xFFFFE00106062800	0x0000000000312000	10	0x00000000100148F0		1393	等待
8084	0xFFFFE000FD96F080	0x000000000030C000	10	0x0000000000460226	123.exe	506	等待

[123.exe]进程模块(48)

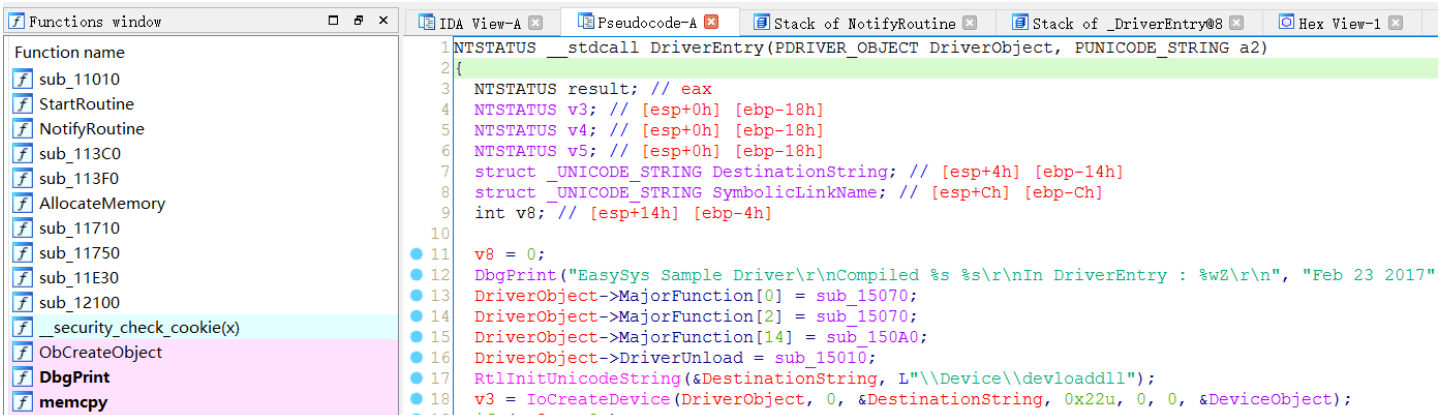
模块路径	基地址	大小	文件厂商
C:\Users\Administrator\Desktop\123.exe	0x0000000000400000	0x00000000000CB000	
C:\Users\Administrator\Desktop\新建文件夹 (2)\MemOpe.dll	0x000000005EC40000	0x00000000008B000	
C:\Windows\SYSwow64\ntdll.dll	0x0000000077E00000	0x00000000017B000	Microsoft Corporation
C:\Windows\SYSwow64\KERNEL32.DLL	0x00000000074BF0000	0x00000000000E0000	Microsoft Corporation
C:\Windows\SYSwow64\KERNELBASE.dll	0x0000000075170000	0x00000000017E000	Microsoft Corporation
C:\Windows\SYSwow64\WINMM.dll	0x0000000074990000	0x0000000000024000	Microsoft Corporation
C:\Windows\SYSwow64\WINMMBASE.dll	0x00000000752F0000	0x0000000000023000	Microsoft Corporation
C:\Windows\SYSwow64\msvcrt.dll	0x0000000076320000	0x00000000000BE000	Microsoft Corporation
C:\Windows\SYSwow64\cfgmgr32.dll	0x00000000749C0000	0x0000000000037000	Microsoft Corporation
C:\Windows\SYSwow64\USER32.dll	0x0000000077BE0000	0x0000000000147000	Microsoft Corporation
C:\Windows\SYSwow64\GDI32.dll	0x0000000754A0000	0x00000000014F000	Microsoft Corporation
C:\Windows\SYSwow64\WS2_32.dll	0x0000000075D90000	0x00000000005F000	Microsoft Corporation
C:\Windows\SYSwow64\sechost.dll	0x0000000075120000	0x0000000000044000	Microsoft Corporation
C:\Windows\SYSwow64\RPCRT4.dll	0x00000000766B0000	0x00000000000AD000	Microsoft Corporation
C:\Windows\SYSwow64\SspiCli.dll	0x0000000074920000	0x0000000000001E000	Microsoft Corporation
C:\Windows\SYSwow64\CRYPTBASE.dll	0x0000000074910000	0x0000000000000A000	Microsoft Corporation
C:\Windows\SYSwow64\bcryptPrimitives.dll	0x0000000075320000	0x0000000000058000	Microsoft Corporation
C:\Windows\SYSwow64\ADVAPI32.dll	0x0000000074DE0000	0x00000000007B000	Microsoft Corporation
C:\Windows\SYSwow64\SHELL32.dll	0x0000000076780000	0x00000000013FE000	Microsoft Corporation
C:\Windows\SYSwow64\windows.storage.dll	0x0000000075890000	0x00000000004FA000	Microsoft Corporation
C:\Windows\SYSwow64\combase.dll	0x00000000763E0000	0x00000000001BD000	Microsoft Corporation
C:\Windows\SYSwow64\shlwapi.dll	0x0000000074940000	0x0000000000045000	Microsoft Corporation
C:\Windows\SYSwow64\kernel.appcore.dll	0x0000000074CD0000	0x0000000000000C000	Microsoft Corporation
C:\Windows\SYSwow64\shcore.dll	0x0000000076220000	0x00000000000008D000	Microsoft Corporation
C:\Windows\SYSwow64\powrprof.dll	0x0000000074FE0000	0x0000000000044000	Microsoft Corporation
C:\Windows\SYSwow64\profapi.dll	0x0000000076310000	0x0000000000000F000	Microsoft Corporation
C:\Windows\SYSwow64\ole32.dll	0x0000000075030000	0x00000000000EB000	Microsoft Corporation
C:\Windows\SYSwow64\OLEAUT32.dll	0x0000000077D60000	0x0000000000092000	Microsoft Corporation
C:\Windows\SYSwow64\comdlg32.dll	0x00000000765A0000	0x00000000000F2000	Microsoft Corporation
C:\Windows\SYSwow64\FirewallAPI.dll	0x0000000074B90000	0x00000000005E000	Microsoft Corporation
C:\Windows\SYSwow64\NETAPI32.dll	0x0000000075380000	0x0000000000013000	Microsoft Corporation
C:\Windows\SYSwow64\WINSPOOL.DRV	0x0000000068F30000	0x0000000000067000	Microsoft Corporation
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_5.82.10586.0_none_811bc0006c44242b\COMCTL32...	0x0000000072D40000	0x0000000000092000	Microsoft Corporation
C:\Windows\SYSwow64\bcrypt.dll	0x0000000072020000	0x00000000001B000	Microsoft Corporation
C:\Windows\SYSwow64\DAVHLPR.DLL	0x0000000071E00000	0x00000000000B000	Microsoft Corporation
C:\Windows\SYSwow64\IMM32.DLL	0x0000000077D30000	0x00000000002B000	Microsoft Corporation
C:\Windows\SYSwow64\fwbase.dll	0x0000000718B0000	0x00000000002C000	Microsoft Corporation
C:\Windows\syswow64\uxtheme.dll	0x0000000069300000	0x0000000000075000	Microsoft Corporation
C:\Windows\SYSwow64\MSCTF.dll	0x0000000075770000	0x00000000011F000	Microsoft Corporation
C:\Windows\syswow64\dwmapi.dll	0x0000000068EC0000	0x00000000001D000	Microsoft Corporation
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.10586.0_none_d3c2e4e965da4528\comctl32.DLL	0x0000000693F0000	0x0000000000020F000	Microsoft Corporation
C:\Windows\SYSwow64\MSVCP110.dll	0x000000005EBB0000	0x0000000000085000	Microsoft Corporation
C:\Windows\SYSwow64\MSVCR110.dll	0x0000000058AE0000	0x00000000000D6000	Microsoft Corporation
C:\Windows\SYSwow64\dbghelp.dll	0x0000000073FE0000	0x00000000013F000	Microsoft Corporation
C:\Windows\SYSTEM32\ntdll.dll	0x00007FFA5AE00000	0x00000000001C1000	Microsoft Corporation
C:\Windows\system32\wow64.dll	0x0000000667A0000	0x0000000000050000	Microsoft Corporation
C:\Windows\system32\wow64win.dll	0x0000000667F0000	0x00000000007A000	Microsoft Corporation
C:\Windows\system32\wow64cpu.dll	0x0000000066870000	0x0000000000008000	Microsoft Corporation

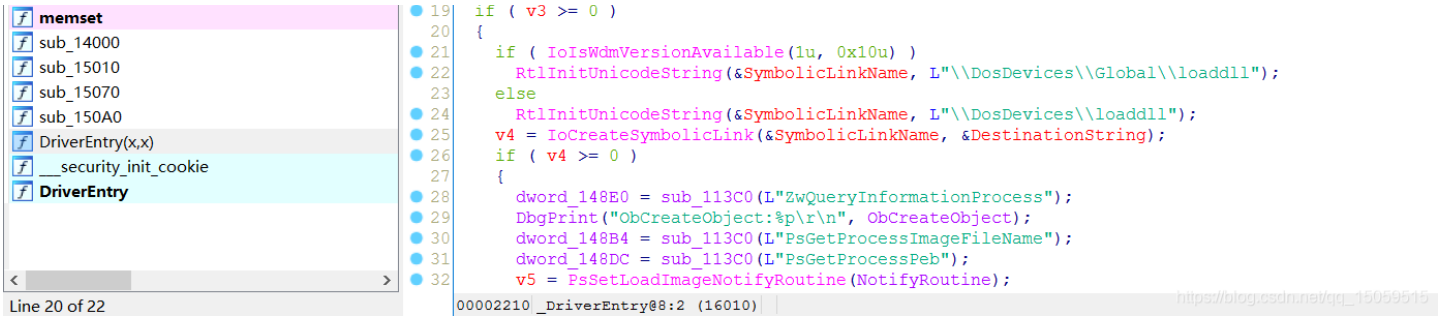
有一个线程Pchunter标红，而且“模块”那里没有，很可疑，先记下。
进程模块这里出现了注入的MemOpe.dll，看来不是内存加载。

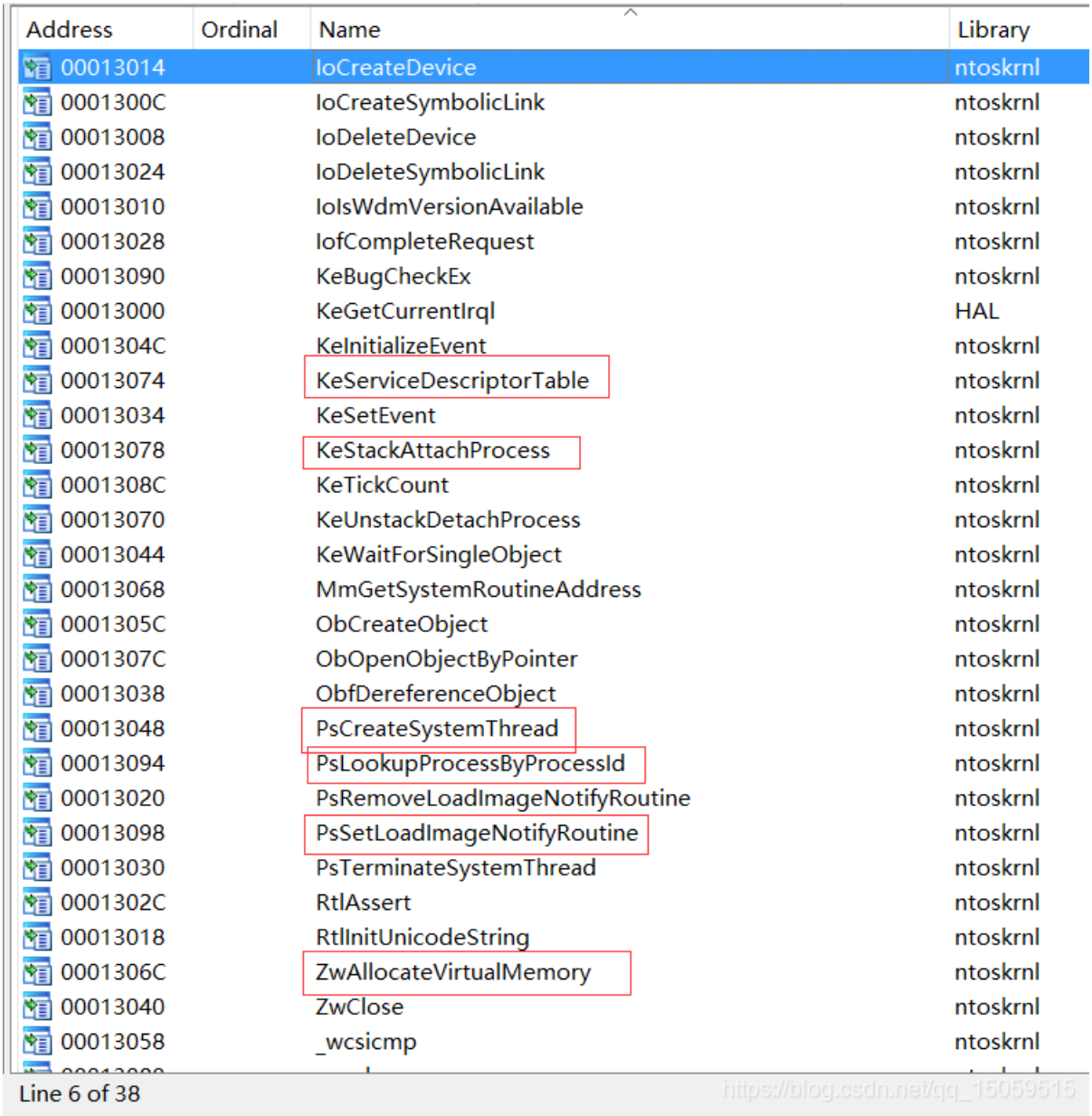
为了减小干扰，先试试把“驱动级别的dll注入器.exe”进程强关了，看驱动还有没有效果。
测试了下仍可以正常注入。说明整个注入过程全由驱动完成，其加载驱动的exe不参与。

由于是x86sys程序，x64dbg用不上，只好拿IDA
（顺便吐槽一下，之前尝试用windbg动态调试看看。那用的叫一个难受，单步1步，windbg的反汇编窗口卡1秒。watch窗口也看不了一些[ebp-xx]之类的变量数据，可能是我还不会用）

代码很干净，函数很少。







Address	Ordinal	Name	Library
00013014		IoCreateDevice	ntoskrnl
0001300C		IoCreateSymbolicLink	ntoskrnl
00013008		IoDeleteDevice	ntoskrnl
00013024		IoDeleteSymbolicLink	ntoskrnl
00013010		IoIsWdmVersionAvailable	ntoskrnl
00013028		IofCompleteRequest	ntoskrnl
00013090		KeBugCheckEx	ntoskrnl
00013000		KeGetCurrentIrql	HAL
0001304C		KeInitializeEvent	ntoskrnl
00013074		KeServiceDescriptorTable	ntoskrnl
00013034		KeSetEvent	ntoskrnl
00013078		KeStackAttachProcess	ntoskrnl
0001308C		KeTickCount	ntoskrnl
00013070		KeUnstackDetachProcess	ntoskrnl
00013044		KeWaitForSingleObject	ntoskrnl
00013068		MmGetSystemRoutineAddress	ntoskrnl
0001305C		ObCreateObject	ntoskrnl
0001307C		ObOpenObjectByPointer	ntoskrnl
00013038		ObfDereferenceObject	ntoskrnl
00013048		PsCreateSystemThread	ntoskrnl
00013094		PsLookupProcessByProcessId	ntoskrnl
00013020		PsRemoveLoadImageNotifyRoutine	ntoskrnl
00013098		PsSetLoadImageNotifyRoutine	ntoskrnl
00013030		PsTerminateSystemThread	ntoskrnl
0001302C		RtlAssert	ntoskrnl
00013018		RtlInitUnicodeString	ntoskrnl
0001306C		ZwAllocateVirtualMemory	ntoskrnl
00013040		ZwClose	ntoskrnl
00013058		_wcsicmp	ntoskrnl

Line 6 of 38

先看看输入表。

注意框起来的API，看到这几个api。我想到了ssdt，attach进程，创建线程，设置模块加载回调，申请r3内存空间。

接下来看看都有什么操作。

1.和加载驱动的exe通信，获取目标进程名和要注入的dll路径

```
NTSTATUS __stdcall sub_150A0(int a1, PIRP Irp)
{
  _DWORD *v2; // ST2C_4
  int v4; // [esp+8h] [ebp-1Ch]
  unsigned int v5; // [esp+Ch] [ebp-18h]
  int v6; // [esp+10h] [ebp-14h]
  NTSTATUS v7; // [esp+14h] [ebp-10h]
  IRP *v8; // [esp+18h] [ebp-Ch]
```

```
ULONG_PTR v9; // [esp+20h] [ebp-4h]

v2 = sub_11010(Irp);
v5 = v2[3];
v8 = Irp->AssociatedIrp.MasterIrp;
v6 = v2[2];
v9 = v2[1];
v4 = v2[3];
if ( v5 == 2236416 )
{
  DbgPrint("MY_CTL_CODE(0)=%d\r\n,MY_CTL_CODE");
  v7 = 0;
}
else if ( v4 == 2236428 )
{
  if ( v8 && v6 == 512 )
    qmemcpy(&dll, v8, 0x200u);
  DbgPrint("[DispatchIoctl] pdll:%ws", &dll);
  v7 = 0;
}
else if ( v4 == 2236432 )
{
  if ( v8 && v6 == 512 )
    qmemcpy(&exe, v8, 0x200u);
  DbgPrint("[DispatchIoctl] pexe:%ws", &exe);
  v7 = 0;
}
else
```


2.设置模块加载回调

```
v5 = PsSetLoadImageNotifyRoutine(NotifyRoutine);
```

再看看回调里面有什么东西

3.判断LoadImage的进程是不是目标进程>>判断加载的模块是不是ntdll.dll>>开一个内核线程

注意这个ntdll

```
if ( ProcessId )
{
  if ( PsLookupProcessByProcessId(ProcessId, &Object) >= 0 )
    sub_11E30(Object, &v22);
  if ( !wcsicmp(&exe, &v22) )
  {
    DbgPrint("exename:%ws moudule:%wZ pexe:%ws", &v22, FullImageName, &exe);
    if ( FullImageName->MaximumLength > 512 )
    {
      P = ExAllocatePool(0, FullImageName->MaximumLength);
      v25 = P;
    }
    memcpy(v25, FullImageName->Buffer, FullImageName->MaximumLength);
    v24 = wcsrchr(v25, 0x5Cu);
    if ( v24 )
    {
      ++v24;
      if ( !wcsicmp(v24, L"ntdll.dll") )
      {
        KeInitializeEvent(&Event, SynchronizationEvent, 0);
        v6 = 0;
        StartContext = &Event;
        v4 = Object;
        v5 = v25;
        v6 = ImageInfo->ImageBase;
        v10 = PsCreateSystemThread(&ThreadHandle, 0x1FFFFFu, 0, 0, 0, StartRoutine, &StartContext);
        if ( v10 < 0 )
          DbgPrint("[LoadImageNotifyRoutine]内核线程创建失败\t\n");
        KeWaitForSingleObject(&Event, 0, 0, 0, 0);
        ZwClose(ThreadHandle);
      }
    }
  }
```


再来看看内核线程执行的函数里面有什么

4.Attach到目标进程（之前Pchunter看到的标红线程应该就是这个原因）

取R3和R0的

ZwProtectVirtualMemory，ZwWriteVirtualMemory，ZwReadVirtualMemory，ZwQueryVirtualMemory，LdrLoadDll（无R0），ZwTestAlert（无R0） 地址

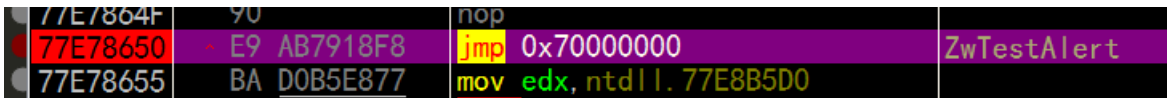
```
ProcessHandle = 0;
v53 = ObOpenObjectByPointer(a1, 512, 0, 0x1FFFFF, 0, 0, &ProcessHandle);
if ( v53 >= 0 )
{
  KeStackAttachProcess(a1, &v52);
  if ( !ZwProtectVirtualMemory || !ZwWriteVirtualMemory || !ZwReadVirtualMemory )
  {
    ZwProtectVirtualMemory = GetProcAddress(a2, "ZwProtectVirtualMemory");
    ZwWriteVirtualMemory = GetProcAddress(a2, "ZwWriteVirtualMemory");
    ZwReadVirtualMemory = GetProcAddress(a2, "ZwReadVirtualMemory");
    ZwQueryVirtualMemory = GetProcAddress(a2, "ZwQueryVirtualMemory");
    DbgPrint(
      "rin3 ZwProtectVirtualMemory:%p ZwWriteVirtualMemory:%p ZwReadVirtualMemory:%p,ZwQueryVirtualMemory:%p\r\n",
      ZwProtectVirtualMemory,
      ZwWriteVirtualMemory,
      ZwReadVirtualMemory,
      ZwQueryVirtualMemory);
    LdrLoadDll = GetProcAddress(a2, "LdrLoadDll");
    ZwTestAlert = GetProcAddress(a2, "ZwTestAlert");
    if ( ZwProtectVirtualMemory && ZwWriteVirtualMemory && ZwReadVirtualMemory && ZwQueryVirtualMemory )
    {
      R0_ZwProtectVirtualMemory = GetR0AddressBySSDT(*(ZwProtectVirtualMemory + 1));
      R0_ZwWriteVirtualMemory = GetR0AddressBySSDT(*(ZwWriteVirtualMemory + 1));
      R0_ZwReadVirtualMemory = GetR0AddressBySSDT(*(ZwReadVirtualMemory + 1));
      R0_ZwQueryVirtualMemory = GetR0AddressBySSDT(*(ZwQueryVirtualMemory + 1));
      DbgPrint(
        "[InjectByHook32]ring0 NtProtectVirtualMemory:%p NtReadVirtualMemory:%p  NtWriteVirtualMemory:%p NtQueryVirtualMemory:%p\r\n",
        R0_ZwProtectVirtualMemory,
        R0_ZwReadVirtualMemory,
        R0_ZwWriteVirtualMemory,
        R0_ZwQueryVirtualMemory);
    }
  }
  KeUnstackDetachProcess(&v52);
```

5.往目标进程申请一片full_access内存放shellcode，并对其用到的API地址进行重定位，同时对ntdll!ZwTestAlert下了一个HOOK，用来跳转到自己的shellcode

（一个进程被Create，ntdll最先加载，再由其加载kernel，user等其他系统DLL。由于ntdll上的线程会经过ntdll!ZwTestAlert这个API，所以对其下Hook来跳转执行自己的shellcode实现DLL加载）

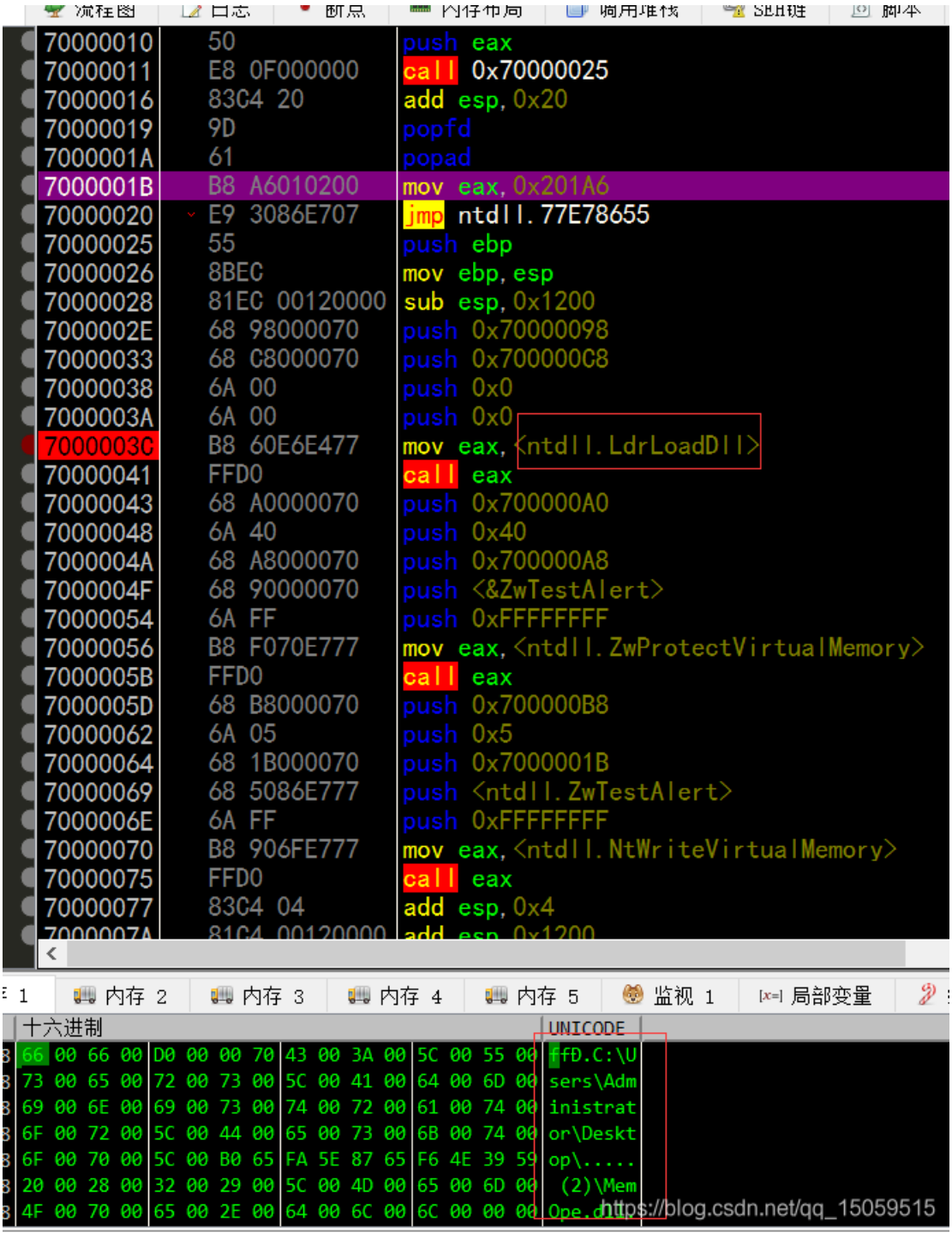
```
v51 = R3_AllocateMemory(ProcessHandle, a2, 4096);
if ( !v51 )
  return -1;
DbgPrint("[InjectByHook32] pBuffer:%x\r\n", v51);
memset(&v10, 0, 0x241u);
qmemcpy(&v8, &loc_14060, 0x1Bu);        // shellcode
qmemcpy(&v14, sub_14000, 0x5Fu);        // shellcode
v27 = ZwTestAlert;
v10 = *ZwTestAlert;
v11 = *(ZwTestAlert + 4);
v28 = v51 + 144;
v29 = 5;
v30 = ZwProtectVirtualMemory;
v26 = LdrLoadDll;
v31 = v51 + 156;
v32 = ZwWriteVirtualMemory;
v7 = &dll;
v6 = &v36;
do
{
  v4 = *v7;
  *v6 = *v7;
  ++v7;
  v6 += 2;
}
while ( v4 );
v33 = 2 * wcslen(&dll);
v34 = 2 * wcslen(&dll);
v49 = 172;
v35 = v51 + 172;
v50 = v51 + 32;
LOBYTE(v5) = -23;
v37 = ZwTestAlert - (v51 + 32);
*(&v5 + 1) = ZwTestAlert - (v51 + 32);
```

```
v25 = v32;
v53 = R0_ZwWriteVirtualMemory(ProcessHandle, v51, &v8, 604, 0);// 写入Shellcode
LOBYTE(v44) = -23;
v42 = 0;
v47 = ZwTestAlert;
v45 = 5;
v43 = v51 - ZwTestAlert - 5;
*(&v44 + 1) = v51 - ZwTestAlert - 5;
v53 = R0_ZwProtectVirtualMemory(ProcessHandle, &v47, &v45, 64, &v42);
if ( v53 >= 0 )
{
  R0_ZwWriteVirtualMemory(ProcessHandle, ZwTestAlert, &v44, 5, &v54);// 对ZwTestAlert下Hook
  R0_ZwProtectVirtualMemory(ProcessHandle, &v47, &v45, v42, &v42);
}
```



```
77E7865A    FFD2        call edx
77E7865C    C3          ret
```

效果图：



总结一下大致流程：

1.目标进程创建>>

2.驱动通过Load_Image回调发现目标进程及其Ntdll>>

3.对其Ntdll线程必经的ntdll!ZwTestAlert下Hook，同时写入shellcode>>

4.Ntdll线程经过ntdll!ZwTestAlert并跳转到shellcode>>

5.执行LdrLoadDll加载目标DLL>>

6.DLL注入完成

尾声：

感谢xjun前辈的指点

~~（小声bb一句，这全是破绽的驱动注入能过TP，我觉得TP真是放水了。。。）~~
说实话，一开始逆的时候。逆到那个Shellcode操作的地方，怎么都看不明白，还以为注入是靠的内核线程attach了目标进程，然后亲自调用了什么API才注入的。这里看不懂，就只能先看其他函数，全部看遍了也没有相关的API和操作。没办法，试试windbg，动态看看对shellcode的操作。。。结果怎么样，前面也吐槽了。
最后还是配合着看线程环境（看到注入dll的是ntdll模块上的线程），看目标进程里的shellcode才分析出来具体原理。

虽然很想发到看雪，不过拙文一篇，就不去献丑了。权当是自娱自乐了