

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](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

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

版权

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



dnf.exe



loaddll32.s
ys



loaddll64.s
ys



MemOpe.
dll



xxx.ini



驱动级别的
dll注入器.
exe



loaddll32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	0xFFFFFE00106062800	0x00000000000312000	10	0x00000000100148F0		1393	等待
8084	0xFFFFFE00FD96F080	0x0000000000030C000	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	0x000000000400000	0x00000000000C8000	
C:\Users\Administrator\Desktop\新建文件夹 (2)\MemOpe.dll	0x000000005EC4000	0x0000000000088000	
C:\Windows\Syswow64\ntdll.dll	0x0000000077E0000	0x0000000000178000	Microsoft Corporation
C:\Windows\Syswow64\KERNEL32.DLL	0x00000000748F000	0x00000000000E0000	Microsoft Corporation
C:\Windows\Syswow64\KERNELBASE.dll	0x000000007517000	0x000000000017E000	Microsoft Corporation
C:\Windows\Syswow64\WINMM.dll	0x000000007499000	0x0000000000024000	Microsoft Corporation
C:\Windows\Syswow64\WINMMBASE.dll	0x00000000752F000	0x0000000000023000	Microsoft Corporation
C:\Windows\Syswow64\msvcrt.dll	0x000000007632000	0x000000000008E000	Microsoft Corporation
C:\Windows\Syswow64\cfgmgr32.dll	0x00000000749C000	0x0000000000037000	Microsoft Corporation
C:\Windows\Syswow64\USER32.dll	0x00000000778E000	0x0000000000147000	Microsoft Corporation
C:\Windows\Syswow64\GDI32.dll	0x00000000754A000	0x000000000014F000	Microsoft Corporation
C:\Windows\Syswow64\WS2_32.dll	0x0000000075D9000	0x000000000005F000	Microsoft Corporation
C:\Windows\Syswow64\sechost.dll	0x000000007512000	0x0000000000044000	Microsoft Corporation
C:\Windows\Syswow64\RPCRT4.dll	0x000000007668000	0x00000000000AD000	Microsoft Corporation
C:\Windows\Syswow64\SspiCli.dll	0x000000007492000	0x000000000001E000	Microsoft Corporation
C:\Windows\Syswow64\CRYPTBASE.dll	0x000000007491000	0x00000000000A0000	Microsoft Corporation
C:\Windows\Syswow64\bcryptPrimitives.dll	0x000000007532000	0x0000000000058000	Microsoft Corporation
C:\Windows\Syswow64\ADVAPI32.dll	0x0000000074DE000	0x0000000000078000	Microsoft Corporation
C:\Windows\Syswow64\SHELL32.dll	0x000000007678000	0x00000000013FE000	Microsoft Corporation
C:\Windows\Syswow64\windows.storage.dll	0x000000007589000	0x00000000004FA000	Microsoft Corporation
C:\Windows\Syswow64\combase.dll	0x00000000763E000	0x00000000001BD000	Microsoft Corporation
C:\Windows\Syswow64\shlwapi.dll	0x000000007494000	0x0000000000045000	Microsoft Corporation
C:\Windows\Syswow64\kernel.appcore.dll	0x0000000074CD000	0x000000000000C000	Microsoft Corporation
C:\Windows\Syswow64\shcore.dll	0x000000007622000	0x000000000008D000	Microsoft Corporation
C:\Windows\Syswow64\powrprof.dll	0x0000000074FE000	0x0000000000044000	Microsoft Corporation
C:\Windows\Syswow64\profapi.dll	0x000000007631000	0x000000000000F000	Microsoft Corporation
C:\Windows\Syswow64\ole32.dll	0x000000007503000	0x000000000008E000	Microsoft Corporation
C:\Windows\Syswow64\OLEAUT32.dll	0x0000000077D6000	0x0000000000092000	Microsoft Corporation
C:\Windows\Syswow64\comdlg32.dll	0x00000000765A000	0x00000000000F2000	Microsoft Corporation
C:\Windows\Syswow64\FirewallAPI.dll	0x0000000074B9000	0x000000000005E000	Microsoft Corporation
C:\Windows\Syswow64\NETAPI32.dll	0x000000007538000	0x0000000000013000	Microsoft Corporation
C:\Windows\Syswow64\WINSPOOL.DRV	0x00000000068F3000	0x00000000000067000	Microsoft Corporation
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144cdf1df_5.82.10586.0_none_811bc006c44242b\COMCTL32...	0x0000000072D4000	0x0000000000092000	Microsoft Corporation
C:\Windows\Syswow64\bcrypt.dll	0x000000007202000	0x000000000001B000	Microsoft Corporation
C:\Windows\Syswow64\DAVHLPR.DLL	0x0000000071E0000	0x0000000000008000	Microsoft Corporation
C:\Windows\Syswow64\IMM32.DLL	0x0000000077D3000	0x000000000002B000	Microsoft Corporation
C:\Windows\Syswow64\fvbase.dll	0x00000000718B000	0x0000000000002C000	Microsoft Corporation
C:\Windows\systemwow64\uxtheme.dll	0x0000000006930000	0x0000000000075000	Microsoft Corporation
C:\Windows\Syswow64\MSCTF.dll	0x000000007577000	0x000000000011F000	Microsoft Corporation
C:\Windows\systemwow64\dwmapi.dll	0x00000000068EF000	0x000000000001D000	Microsoft Corporation
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144cdf1df_6.0.10586.0_none_d3c2e4e965da4528\comctl32.DLL	0x000000000693F000	0x0000000000020F000	Microsoft Corporation
C:\Windows\Syswow64\MSVCP110.dll	0x0000000005EB0000	0x0000000000085000	Microsoft Corporation
C:\Windows\Syswow64\MSVCR110.dll	0x00000000058A0000	0x000000000006F000	Microsoft Corporation
C:\Windows\Syswow64\dbghelp.dll	0x0000000073FE000	0x0000000000013F000	Microsoft Corporation
C:\Windows\SYSTEM32\ntdll.dll	0x00007FFA5AE00000	0x000000000001C1000	Microsoft Corporation
C:\Windows\system32\wow64.dll	0x00000000667A0000	0x0000000000050000	Microsoft Corporation
C:\Windows\system32\wow64win.dll	0x00000000667F0000	0x000000000007A000	Microsoft Corporation
C:\Windows\system32\wow64cpu.dll	0x0000000066870000	0x0000000000080000	Microsoft Corporation

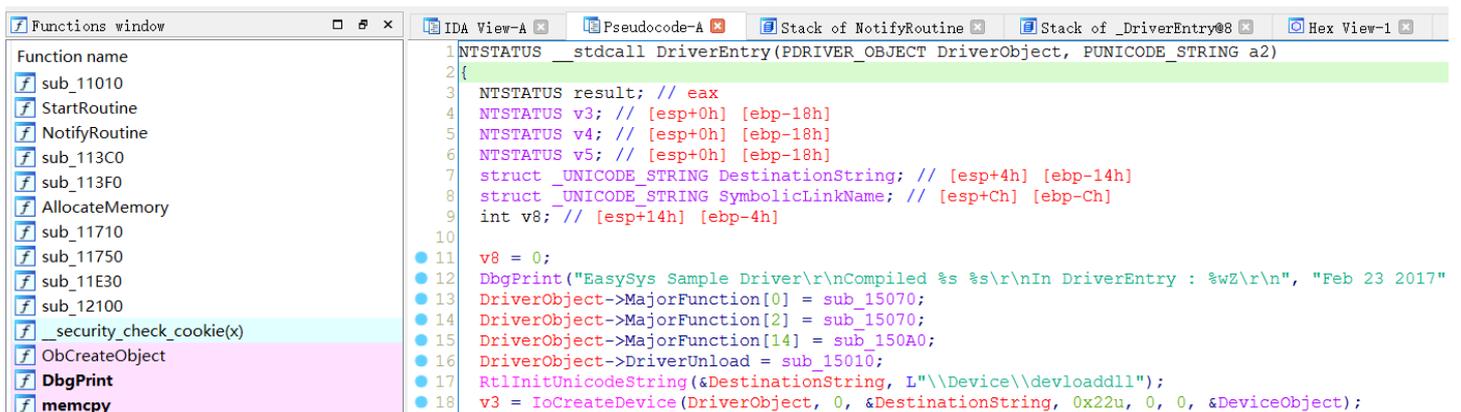
有一个线程Pchunter标红，而且“模块”那里没有，很可疑，先记下。
进程模块这里出现了注入的MemOpe.dll，看来不是内存加载。

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。



```

19  if ( v3 >= 0 )
20  {
21      if ( IoIsWdmVersionAvailable(lu, 0x10u) )
22          RtlInitUnicodeString(&SymbolicLinkName, L"\\DosDevices\\Global\\loaddll");
23      else
24          RtlInitUnicodeString(&SymbolicLinkName, L"\\DosDevices\\loaddll");
25      v4 = IoCreateSymbolicLink(&SymbolicLinkName, &DestinationString);
26      if ( v4 >= 0 )
27      {
28          dword_148E0 = sub_113C0(L"ZwQueryInformationProcess");
29          DbgPrint("ObCreateObject:%p\r\n", ObCreateObject);
30          dword_148B4 = sub_113C0(L"PsGetProcessImageFileName");
31          dword_148DC = sub_113C0(L"PsGetProcessPeb");
32          v5 = PsSetLoadImageNotifyRoutine(NotifyRoutine);

```

Line 20 of 22

00002210_DriverEntry@8:2 (16010)

https://blog.csdn.net/qq_18059515

Address	Ordinal	Name	Library
00013014		IoCreateDevice	ntoskrnl
0001300C		IoCreateSymbolicLink	ntoskrnl
00013008		IoDeleteDevice	ntoskrnl
00013024		IoDeleteSymbolicLink	ntoskrnl
00013010		IoIsWdmVersionAvailable	ntoskrnl
00013028		IoofCompleteRequest	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

https://blog.csdn.net/qq_18059515

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

```

1 NTSTATUS __stdcall sub_150A0(int a1, PIRP Irp)
2 {
3     _DWORD *v2; // ST2C_4
4     int v4; // [esp+8h] [ebp-1Ch]
5     unsigned int v5; // [esp+Ch] [ebp-18h]
6     int v6; // [esp+10h] [ebp-14h]
7     NTSTATUS v7; // [esp+14h] [ebp-10h]
8     IRP *v8; // [esp+18h] [ebp-Ch]

```

```

9  ULONG_PTR v9; // [esp+20h] [ebp-4h]
10
11 v2 = sub_11010(Irp);
12 v5 = v2[3];
13 v8 = Irp->AssociatedIrp.MasterIrp;
14 v6 = v2[2];
15 v9 = v2[1];
16 v4 = v2[3];
17 if ( v5 == 2236416 )
18 {
19     DbgPrint("MY_CTL_CODE(0)=%d\r\n",MY_CTL_CODE);
20     v7 = 0;
21 }
22 else if ( v4 == 2236428 )
23 {
24     if ( v8 && v6 == 512 )
25         qmemcpy(&dll, v8, 0x200u);
26     DbgPrint("[DispatchIoctl] pdll:%ws", &dll);
27     v7 = 0;
28 }
29 else if ( v4 == 2236432 )
30 {
31     if ( v8 && v6 == 512 )
32         qmemcpy(&exe, v8, 0x200u);
33     DbgPrint("[DispatchIoctl] pexe:%ws", &exe);
34     v7 = 0;
35 }
36 else

```

https://blog.csdn.net/qq_15059515

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, 0x5C);
        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, 0x1FFFFFFu, 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);
            }
        }
    }
}
}
```

https://blog.csdn.net/qq_15059515

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, 0x1FFFFFF, 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);
}

```

77E7864F	90	nop	
77E78650	E9 AB7918F8	jmp 0x70000000	ZwTestAlert
77E78655	BA D0B5E877	mov edx, ntdll.77E8B5D0	

77E7865A	FFD2	call edx
77E7865C	C3	ret

效果图:

The screenshot displays a debugger's assembly window and a memory dump. The assembly window shows the following instructions:

```

70000010 50 push eax
70000011 E8 0F000000 call 0x70000025
70000016 83C4 20 add esp, 0x20
70000019 9D popfd
7000001A 61 popad
7000001B B8 A6010200 mov eax, 0x201A6
70000020 E9 3086E707 jmp ntdll.77E78655
70000025 55 push ebp
70000026 8BEC mov ebp, esp
70000028 81EC 00120000 sub esp, 0x1200
7000002E 68 98000070 push 0x70000098
70000033 68 C8000070 push 0x700000C8
70000038 6A 00 push 0x0
7000003A 6A 00 push 0x0
7000003C B8 60E6E477 mov eax, <ntdll.LdrLoadDll>
70000041 FFD0 call eax
70000043 68 A0000070 push 0x700000A0
70000048 6A 40 push 0x40
7000004A 68 A8000070 push 0x700000A8
7000004F 68 90000070 push <&ZwTestAlert>
70000054 6A FF push 0xFFFFFFFF
70000056 B8 F070E777 mov eax, <ntdll.ZwProtectVirtualMemory>
7000005B FFD0 call eax
7000005D 68 B8000070 push 0x700000B8
70000062 6A 05 push 0x5
70000064 68 1B000070 push 0x7000001B
70000069 68 5086E777 push <ntdll.ZwTestAlert>
7000006E 6A FF push 0xFFFFFFFF
70000070 B8 906FE777 mov eax, <ntdll.NtWriteVirtualMemory>
70000075 FFD0 call eax
70000077 83C4 04 add esp, 0x4
7000007A 81C4 00120000 add esp, 0x1200

```

The memory dump below shows hex and ASCII values. A red box highlights the path:

```

十六进制  UNICODE
66 00 66 00 D0 00 00 70 43 00 3A 00 5C 00 55 00 FD.C:\U
73 00 65 00 72 00 73 00 5C 00 41 00 64 00 6D 00 sers\Admin
69 00 6E 00 69 00 73 00 74 00 72 00 61 00 74 00 inistrat
6F 00 72 00 5C 00 44 00 65 00 73 00 6B 00 74 00 or\Desktop
6F 00 70 00 5C 00 B0 65 FA 5E 87 65 F6 4E 39 59 op\....
20 00 28 00 32 00 29 00 5C 00 4D 00 65 00 6D 00 (2)\Mem
4F 00 70 00 65 00 2E 00 64 00 6C 00 6C 00 00 00 Ope...

```

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ntdllZwTestAlert下Hook, 同时写入shellcode>>
- 4.Ntdll线程经过ntdllZwTestAlert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



[创作打卡挑战赛](#) >

[赢取流量/现金/CSDN周边激励大奖](#)