数字驱动分析笔记之HookPort

参考资料：

1、腾讯管家攻防驱动分析-TsFltMgr

2、发一个可编译，可替换的hookport代码

3、为什么win32k.sys在System进程空间无法访问

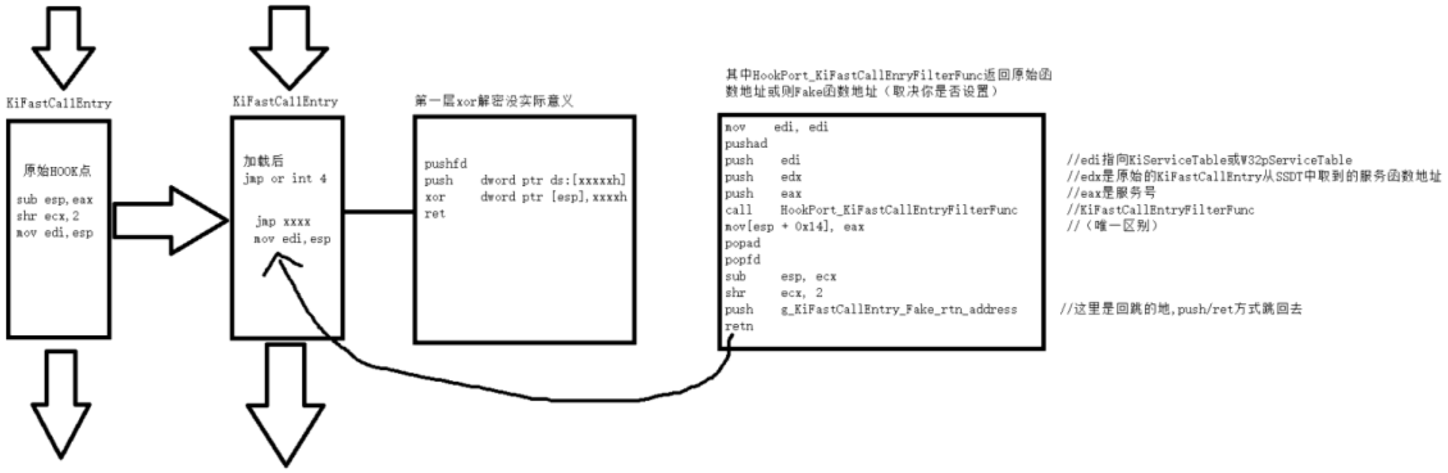
4、明明白白自旋锁

	推荐使用的环境	备注
操作系统	Windows 7 SP3	简体中文版
虚拟机	VM	版本号：15
编译器	VS2013 + WDK8.1	
调试	Windbg	sxe ld xxxx.sys

目录：

1、流程图文介绍

1、Hook前后的对比图


KiFastCallEntry
原始HOOK点
KiFastCallEntry
加载后
第一层xor解密没实际意义
其中HookPort_KiFastCallEnryFilterFunc返回原始函数地址或则Fake函数地址（取决你是否设置）

```
sub  esp, eax
shr  ecx, 2
mov  edi, esp
```

```
jmp or int 4

 jmp xxxx
 mov edi, esp
```

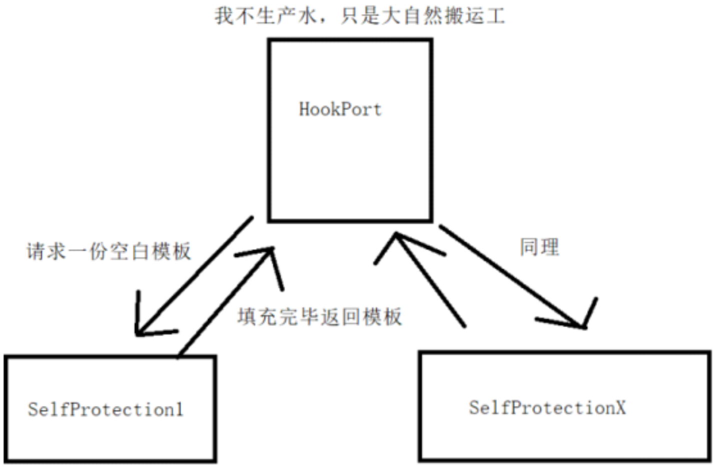
```
pushfd
push    dword ptr ds:[xxxxxh]
xor     dword ptr [esp], xxxxh
ret
```

```
mov     edi, edi
pushad
push     edi                                   //edi指向KiServiceTable或W32pServiceTable
push     edx                                   //edx是原始的KiFastCallEntry从SSDT中取到的服务函数地址
push     eax                                   //eax是服务号
call     HookPort_KiFastCallEntryFilterFunc    //KiFastCallEntryFilterFunc
mov[esp + 0x14], eax                           //（唯一区别）
popad
popfd
sub      esp, ecx
shr      ecx, 2
push     g_KiFastCallEntry_Fake_rtn_address   //这里是回跳的地,push/ret方式跳回去
retn
```

2、HookPort的工作流程

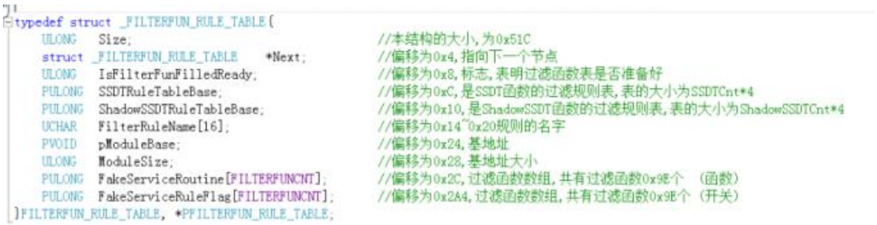
HookPort负责构造一份空白的Hook模板（不负责编写对应的Fake函数，导出给SalfProtectionX用）

可以理解为老板（HookPort）小弟（SelfProtectionX）



理论上我们是可以有无数个SelfProtectionX，但是大数字最大限制16个

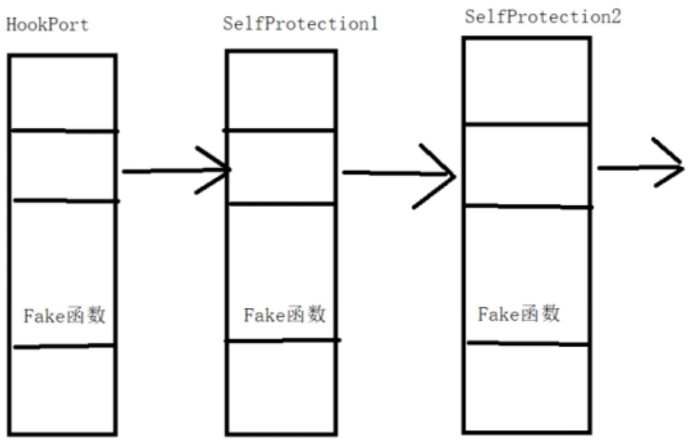
Hook模板结构如下（单向链表结构）：



```
typedef struct _FILTERFUN_RULE_TABLE{
    ULONG   Size;                                      //本结构的大小,为0x51C
    struct _FILTERFUN_RULE_TABLE     *Next;            //偏移为0x4,指向下一个节点
    ULONG   IsFilterFunFilledReady;                    //偏移为0x8,标志,表明过滤函数表是否准备好
    PULONG  SSDTRuleTableBase;                         //偏移为0xC,是SSDT函数的过滤规则表,表的大小为SSDTCnt*4
    PULONG  ShadowSSDTRuleTableBase;                   //偏移为0x10,是ShadowSSDT函数的过滤规则表,表的大小为ShadowSSDTCnt*4
    UCHAR   FilterRuleName[16];                        //偏移为0x14~0x20规则的名字
    PVOID   pModuleBase;                               //偏移为0x24,基地址
    ULONG   ModuleSize;                                //偏移为0x28,基地址大小
    PULONG  FakeServiceRoutine[FILTERFUNCNT];          //偏移为0x2C,过滤函数数组,共有过滤函数0x9E个  (函数)
    PULONG  FakeServiceRuleFlag[FILTERFUNCNT];         //偏移为0x2A4,过滤函数数组,共有过滤函数0x9E个 (开关)
}FILTERFUN_RULE_TABLE, *PFILTERFUN_RULE_TABLE;
```

举个例子：

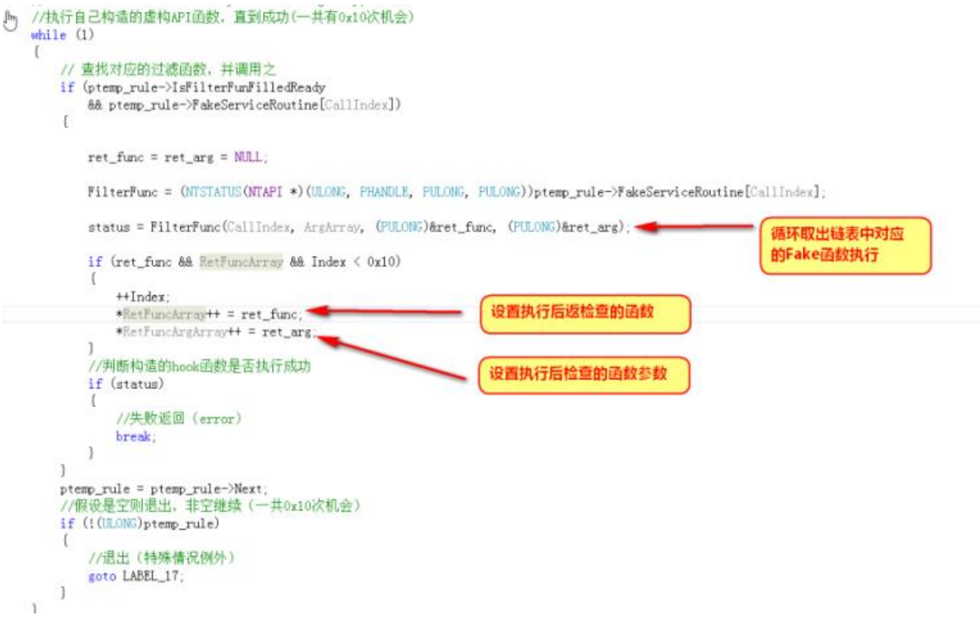
假设我们一共有SelfProtection1、SelfProtection2两个驱动设置了对应的Fake_CraeteProcess函数



原始CreateProcess->KiFastCallEntry->Filter_CreateProcess代理函数->HookPort_DoFilter

循环将链表中所有Fake函数取出来并执行，直到链表下一个为零终止

必须全部所有Fake函数合法返回才算正确，其中一个返回错误都算错误



```
//执行自己构造的虚构API函数，直到成功(一共有0x10次机会)
while (1)
{
    // 查找对应的过滤函数，并调用之
    if (ptemp_rule->IsFilterFunFilledReady
        && ptemp_rule->FakeServiceRoutine[CallIndex])
    {

        ret_func = ret_arg = NULL;

        FilterFunc = (NTSTATUS(NTAPI *)(ULONG, PHANDLE, PULONG, PULONG))ptemp_rule->FakeServiceRoutine[CallIndex];

        status = FilterFunc(CallIndex, ArgArray, (PULONG)&ret_func, (PULONG)&ret_arg);

        if (ret_func && RetFuncArray && Index < 0x10)
        {
            ++Index;
            *RetFuncArray++ = ret_func;
            *RetFuncArgArray++ = ret_arg;
        }
        //判断构造的hook函数是否执行成功
        if (status)
        {
            //失败返回 (error)
            break;
        }
    }
    ptemp_rule = ptemp_rule->Next;
    //假设是空则退出，非空继续 (一共0x10次机会)
    if (!(ULONG)ptemp_rule)
    {
        //退出 (特殊情况例外)
        goto LABEL_17;
    }
}
```

2、驱动入口点DriverEntry

如何调试：
首先输入：sxe ld xxxxx.sys　　　中断
然后输入：lmvm xxxxx　　　　　获取基地址，后面基地址+偏移

代码逻辑流程：

1、获取系统版本信息，假设是win10将Global_Version_Win10_Flag变量置1

2、安全模式下禁止启动

3、创建\\Device\\***HookPort设备和\\DosDevices\\***HookPort符号链接

4、设备DeviceExtension驱动接口，为3600SelfProtection服务

5、注册IRP_MJ_CREATE、IRP_MJ_CLOSE、IRP_MJ_DEVICE_CONTROL

6、执行HookPort_InitSDT函数该函数实现功能如下：

6、1 设置内核API过滤函数

6、2 挂钩KiFastCallEntry

6、3 创建线程、进程、模块回调

6、4 IAT方式挂钩KeUserModeCallback，可以拦截DLL注入、键盘劫持等等。

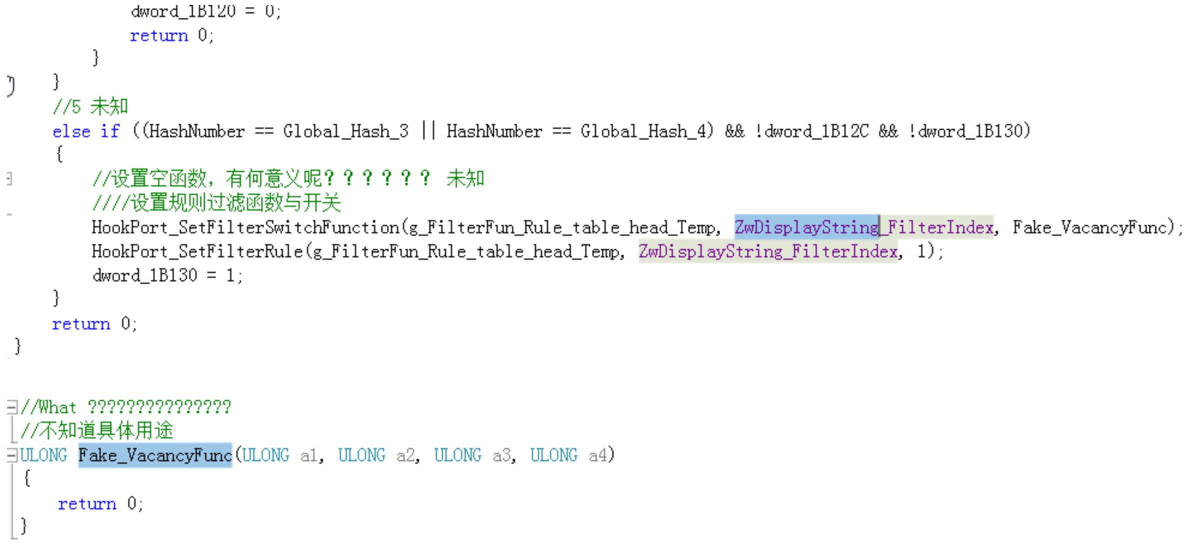
7、初始化驱动导出接口

8、执行HookPort_19230函数（不知取什么名字好）

8、1 假设是Win2K（Int 2E）就挂钩KiSystemService

8、2 实现LoadImageNotifyRoutine对应的Fake函数

8、3 LoadImageNotifyRoutine的Fake函数是根据你打开个某个进程设置ZwDisplayString对应的Fake函数为空函数



```
            dword_1B120 = 0;
            return 0;
        }
    }
    //5 未知
    else if ((HashNumber == Global_Hash_3 || HashNumber == Global_Hash_4) && !dword_1B12C && !dword_1B130)
    {
        //设置空函数，有何意义呢？？？？？？ 未知
        ////设置规则过滤函数与开关
        HookPort_SetFilterSwitchFunction(g_FilterFun_Rule_table_head_Temp, ZwDisplayString_FilterIndex, Fake_VacancyFunc);
        HookPort_SetFilterRule(g_FilterFun_Rule_table_head_Temp, ZwDisplayString_FilterIndex, 1);
        dword_1B130 = 1;
    }
    return 0;
}

//What ???????????????
//不知道具体用途
ULONG Fake_VacancyFunc(ULONG a1, ULONG a2, ULONG a3, ULONG a4)
{
    return 0;
}
```

代码实现：

```
//**************************************
// 函数名称: DriverEntry
// 函数说明：驱动程序入口
// 作    者：Mr.M
// 参考网址：
// 作成日期：2019/11/29
// 返 回 值: NTSTATUS
// 参    数: IN PDRIVER_OBJECT DriverObj
// 参    数: IN PUNICODE_STRING RegPath
//**************************************
NTSTATUS DriverEntry(
        IN PDRIVER_OBJECT  DriverObject,            //代表本驱动的驱动对象
        IN PUNICODE_STRING RegPath                          //驱动的路径，在注册表中
        )
{
```

```
NTSTATUS Status = STATUS_INVALID_DEVICE_REQUEST;
UNICODE_STRING          SymbolicLinkName;
UNICODE_STRING          DestinationString;
PDEVICE_OBJECT          DeviceObject = NULL;
Global_DriverObject = (ULONG)DriverObject;
//1、获取版本信息
Status = HookPort_PsGetVersion();
if (!NT_SUCCESS(Status))
{
        return Status;
}
//2、安全模式下不启动
if (*(ULONG*)InitSafeBootMode)
{
        if (*(ULONG*)InitSafeBootMode == 1)
        {
                Status = RtlCheckRegistryKey(RTL_REGISTRY_CONTROL, HookPort_Minimal);
        }
        else
        {
                if (*(ULONG*)InitSafeBootMode <= 1u || *(ULONG*)InitSafeBootMode > 3u)
                        return STATUS_NOT_SAFE_MODE_DRIVER;
                Status = RtlCheckRegistryKey(RTL_REGISTRY_CONTROL, HookPort_Network);
        }
        if (Status < 0)
                return STATUS_NOT_SAFE_MODE_DRIVER;
}
//2、创建设备
RtlInitUnicodeString(&DestinationString, HookPort_DeviceName);
RtlInitUnicodeString(&SymbolicLinkName, HookPort_LinkName);
Status = IoCreateDevice(
        DriverObject,
        sizeof(HOOKPORT_EXTENSION),             //扩展18u
        &DestinationString,
        FILE_DEVICE_UNKNOWN,                    //#define FILE_DEVICE_UNKNOWN          0x00000022
        FILE_DEVICE_SECURE_OPEN,                // DeviceCharacteristics   ,#define FILE_DEVICE_SECURE_OPEN             0x00000100
        FALSE,
        &DeviceObject);
if (!NT_SUCCESS(Status))
{

KdPrint(("HookPort: DriverEntry IoCreateDevice failed,err=%08x\n", Status));
return Status;
}

//3、给设备创建一个符号链接
Status = IoCreateSymbolicLink(&SymbolicLinkName, &DestinationString);
if (!NT_SUCCESS(Status)){
        KdPrint(("HookPort: DriverEntry IoCreateSymbolicLink failed,err=%08x\n", Status));
        IoDeleteDevice(DeviceObject);
        return Status;
}

//4、DeviceControl都是些开启调试信息相关的直接无视
DriverObject->MajorFunction[IRP_MJ_CREATE] = (PDRIVER_DISPATCH)HookPort_Create;
DriverObject->MajorFunction[IRP_MJ_CLOSE] = (PDRIVER_DISPATCH)HookPort_Close;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = (PDRIVER_DISPATCH)HookPort_DeviceControl;

//5、初始化部分各种hook、创建进程、线程回调等等
if (!NT_SUCCESS(HookPort_InitSDT()))
{
        IoDeleteSymbolicLink(&SymbolicLinkName);
        IoDeleteDevice(DeviceObject);
        return STATUS_UNSUCCESSFUL;
}
//6、初始化导出接口函数
HookPort_InitDeviceExtInterface(DeviceObject);
//7、
//1、根据条件判断是否启用FakeKiSystemService的hook
//2、初始化扩展结构，导出给另外一个sys使用
HookPort_19230();
```

```
    KdPrint(("360HookPort驱动加载成功\t\n"));
    DriverObject->DriverUnload = DriverUnload;
    return STATUS_SUCCESS;
}
```

3、驱动高低版本区别

正文：

1、高低版本HookPort代码区别

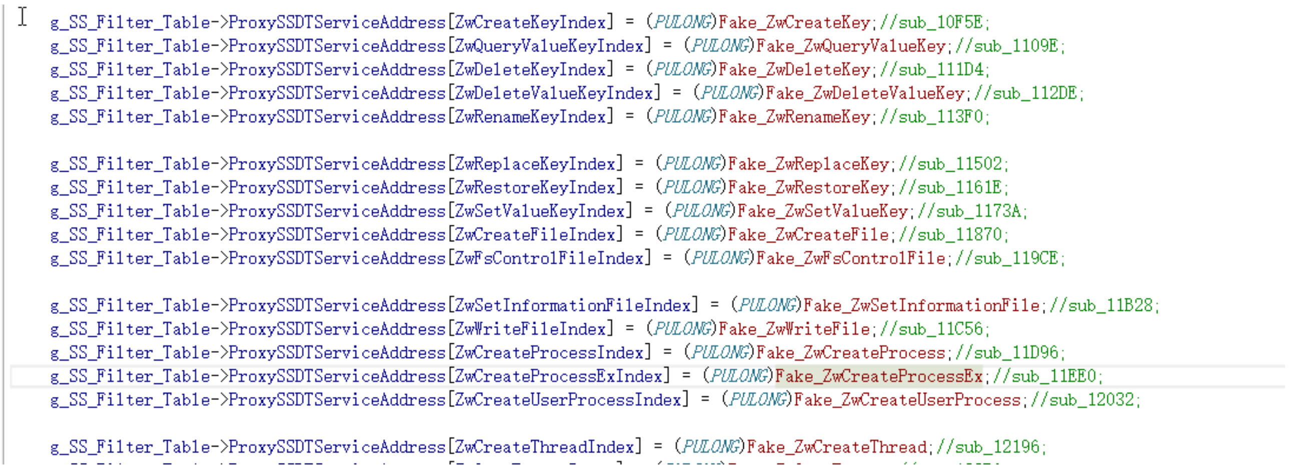
底：layerfsd作者发布的Hookport为蓝本作为低版本（2010年）

高：笔者逆向的Hookport高版本（2019年）

2、Hook代理函数优化

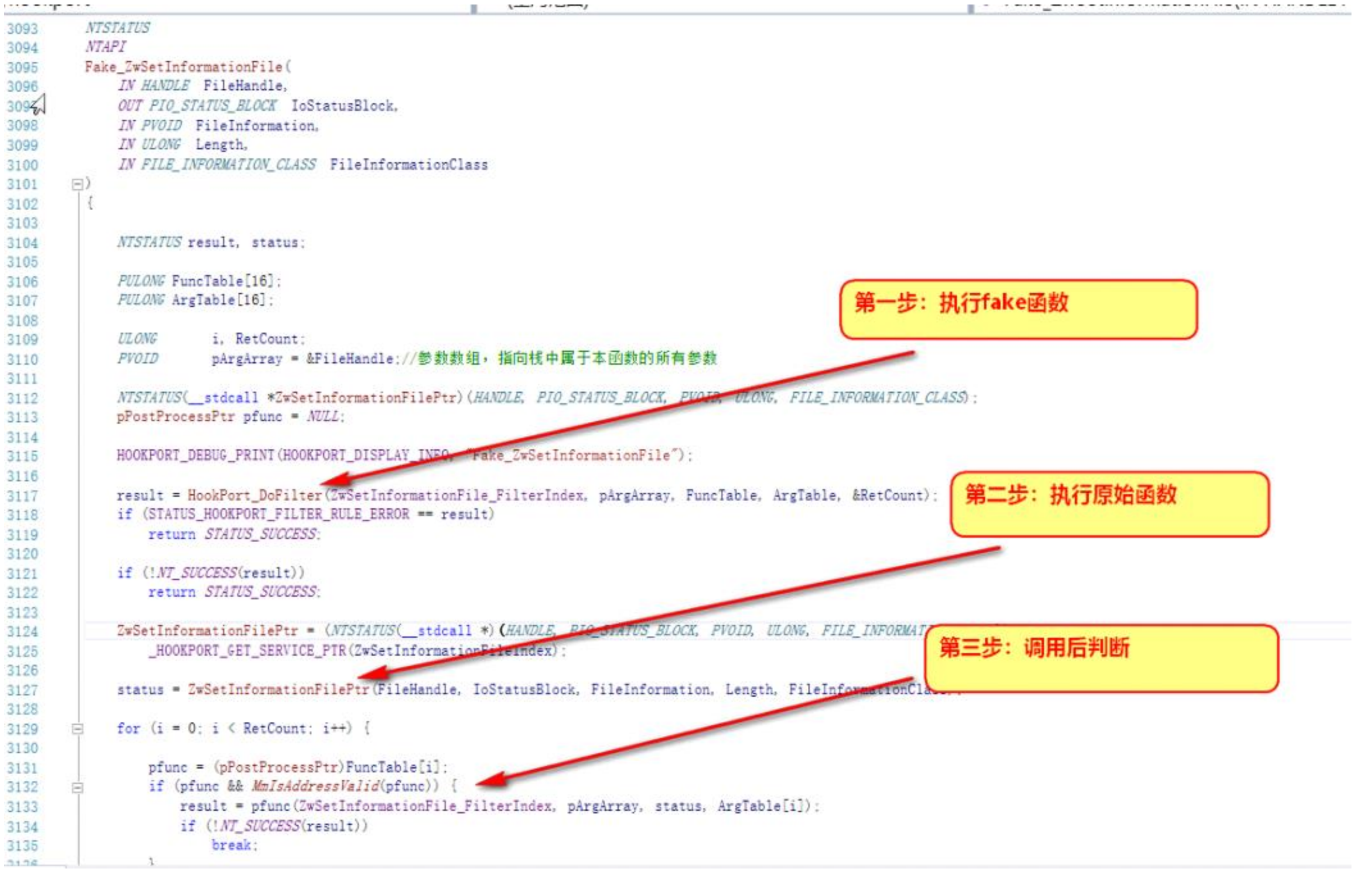
2、1 低版本

1、我们发现一个问题，就是有多少个fake函数定义多少个相同的代理函数



```
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwCreateKeyIndex] = (PULONG)Fake_ZwCreateKey;//sub_10F5E;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwQueryValueKeyIndex] = (PULONG)Fake_ZwQueryValueKey;//sub_1109E;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwDeleteKeyIndex] = (PULONG)Fake_ZwDeleteKey;//sub_111D4;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwDeleteValueKeyIndex] = (PULONG)Fake_ZwDeleteValueKey;//sub_112DE;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwRenameKeyIndex] = (PULONG)Fake_ZwRenameKey;//sub_113F0;

g_SS_Filter_Table->ProxySSDTServiceAddress[ZwReplaceKeyIndex] = (PULONG)Fake_ZwReplaceKey;//sub_11502;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwRestoreKeyIndex] = (PULONG)Fake_ZwRestoreKey;//sub_1161E;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwSetValueKeyIndex] = (PULONG)Fake_ZwSetValueKey;//sub_1173A;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwCreateFileIndex] = (PULONG)Fake_ZwCreateFile;//sub_11870;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwFsControlFileIndex] = (PULONG)Fake_ZwFsControlFile;//sub_119CE;

g_SS_Filter_Table->ProxySSDTServiceAddress[ZwSetInformationFileIndex] = (PULONG)Fake_ZwSetInformationFile;//sub_11B28;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwWriteFileIndex] = (PULONG)Fake_ZwWriteFile;//sub_11C56;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwCreateProcessIndex] = (PULONG)Fake_ZwCreateProcess;//sub_11D96;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwCreateProcessExIndex] = (PULONG)Fake_ZwCreateProcessEx;//sub_11EE0;
g_SS_Filter_Table->ProxySSDTServiceAddress[ZwCreateUserProcessIndex] = (PULONG)Fake_ZwCreateUserProcess;//sub_12032;

g_SS_Filter_Table->ProxySSDTServiceAddress[ZwCreateThreadIndex] = (PULONG)Fake_ZwCreateThread;//sub_12196;
```

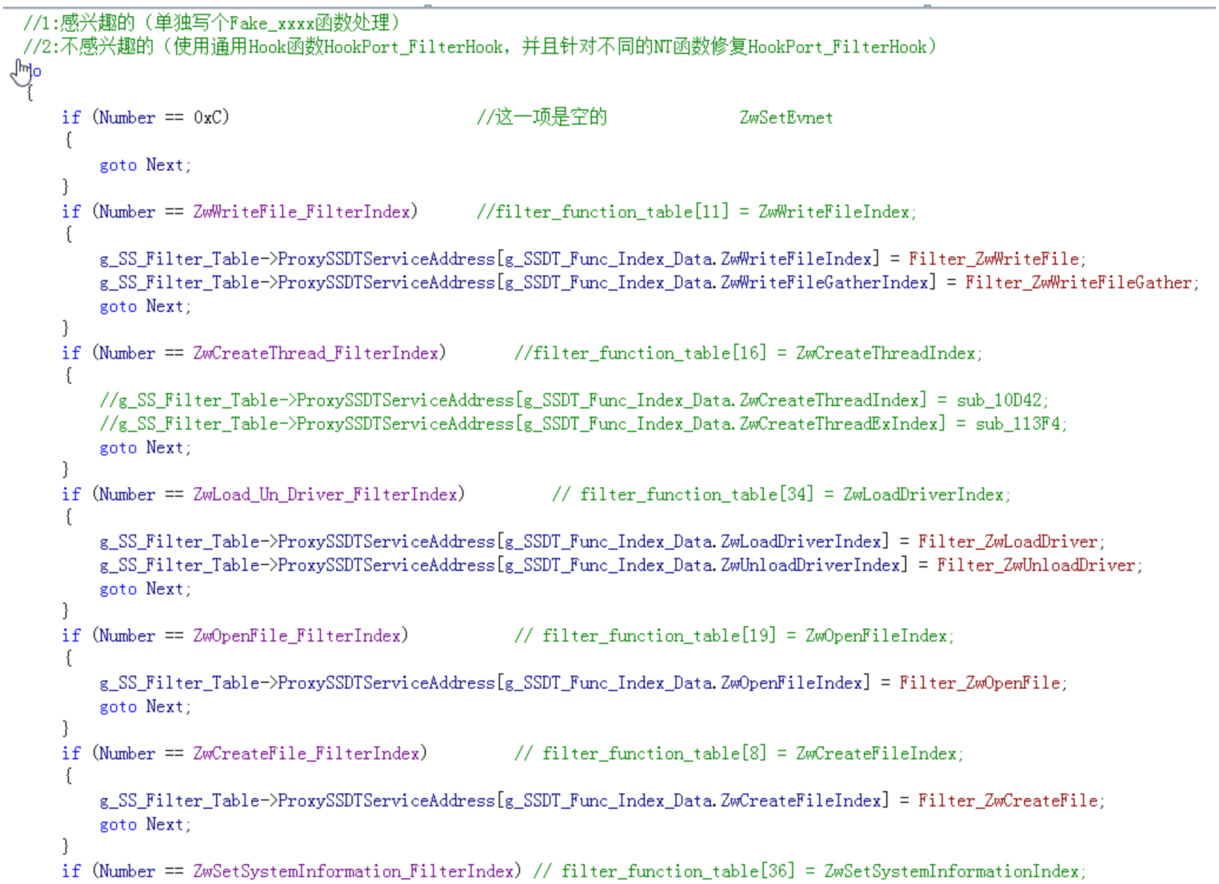
2、我们发现代理函数逻辑基本相同，除了参数个数不一致



```
NTSTATUS
NTAPI
Fake_ZwSetInformationFile(
    IN HANDLE  FileHandle,
    OUT PIO_STATUS_BLOCK  IoStatusBlock,
    IN PVOID  FileInformation,
    IN ULONG  Length,
    IN FILE_INFORMATION_CLASS  FileInformationClass
)
{

    NTSTATUS result, status;

    PULONG FuncTable[16];
    PULONG ArgTable[16];

    ULONG       i, RetCount;
    PVOID       pArgArray = &FileHandle;//参数数组，指向栈中属于本函数的所有参数

    NTSTATUS(__stdcall *ZwSetInformationFilePtr)(HANDLE, PIO_STATUS_BLOCK, PVOID, ULONG, FILE_INFORMATION_CLASS);
    pPostProcessPtr pfunc = NULL;

    HOOKPORT_DEBUG_PRINT(HOOKPORT_DISPLAY_INFO, "Fake_ZwSetInformationFile");

    result = HookPort_DoFilter(ZwSetInformationFile_FilterIndex, pArgArray, FuncTable, ArgTable, &RetCount);
    if (STATUS_HOOKPORT_FILTER_RULE_ERROR == result)
        return STATUS_SUCCESS;

    if (!NT_SUCCESS(result))
        return STATUS_SUCCESS;

    ZwSetInformationFilePtr = (NTSTATUS(__stdcall *)(HANDLE, PIO_STATUS_BLOCK, PVOID, ULONG, FILE_INFORMAT
        _HOOKPORT_GET_SERVICE_PTR(ZwSetInformationFileIndex);

    status = ZwSetInformationFilePtr(FileHandle, IoStatusBlock, FileInformation, Length, FileInformationClass);

    for (i = 0; i < RetCount; i++) {

        pfunc = (pPostProcessPtr)FuncTable[i];
        if (pfunc && MmIsAddressValid(pfunc)) {
            result = pfunc(ZwSetInformationFile_FilterIndex, pArgArray, status, ArgTable[i]);
            if (!NT_SUCCESS(result))
                break;
        }
```

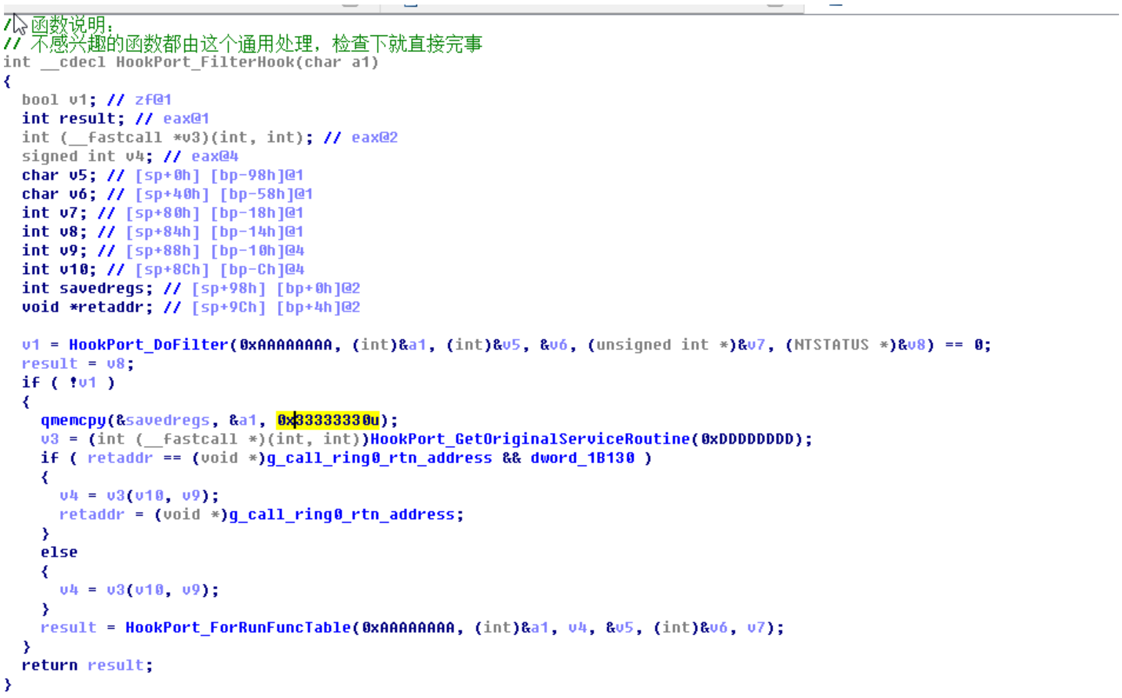
2、2 高版本

1、除了个别感兴趣的其他都通用函数模板处理



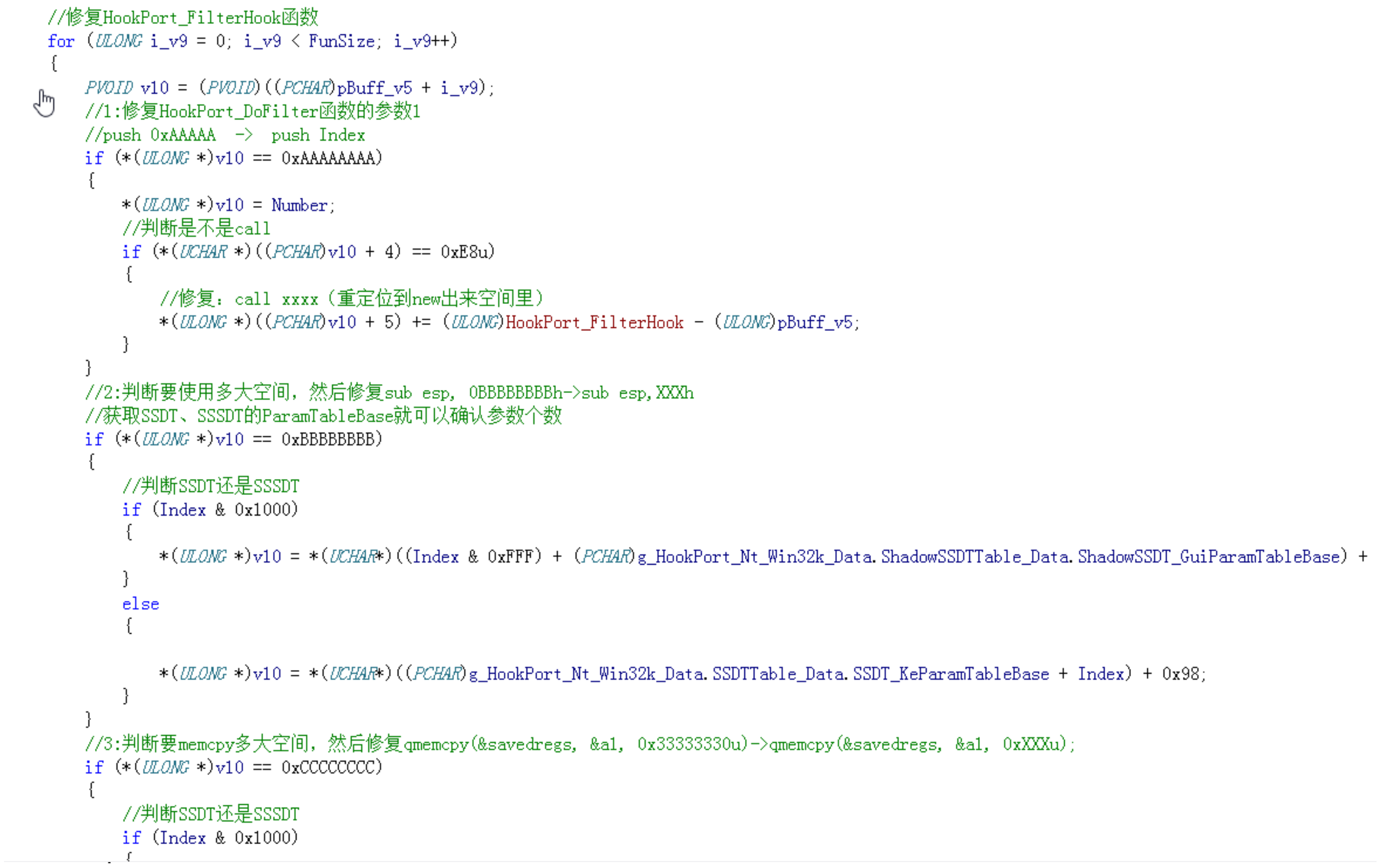
```
  //1:感兴趣的（单独写个Fake_xxxx函数处理）
  //2:不感兴趣的（使用通用Hook函数HookPort_FilterHook，并且针对不同的NT函数修复HookPort_FilterHook）
  do
  {
      if (Number == 0xC)                        //这一项是空的              ZwSetEvnet
      {
          goto Next;
      }
      if (Number == ZwWriteFile_FilterIndex)      //filter_function_table[11] = ZwWriteFileIndex;
      {
          g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwWriteFileIndex] = Filter_ZwWriteFile;
          g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwWriteFileGatherIndex] = Filter_ZwWriteFileGather;
          goto Next;
      }
      if (Number == ZwCreateThread_FilterIndex)       //filter_function_table[16] = ZwCreateThreadIndex;
      {
          //g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwCreateThreadIndex] = sub_10D42;
          //g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwCreateThreadExIndex] = sub_113F4;
          goto Next;
      }
      if (Number == ZwLoad_Un_Driver_FilterIndex)         // filter_function_table[34] = ZwLoadDriverIndex;
      {
          g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwLoadDriverIndex] = Filter_ZwLoadDriver;
          g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwUnloadDriverIndex] = Filter_ZwUnloadDriver;
          goto Next;
      }
      if (Number == ZwOpenFile_FilterIndex)           // filter_function_table[19] = ZwOpenFileIndex;
      {
          g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwOpenFileIndex] = Filter_ZwOpenFile;
          goto Next;
      }
      if (Number == ZwCreateFile_FilterIndex)         // filter_function_table[8] = ZwCreateFileIndex;
      {
          g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwCreateFileIndex] = Filter_ZwCreateFile;
          goto Next;
      }
      if (Number == ZwSetSystemInformation_FilterIndex) // filter_function_table[36] = ZwSetSystemInformationIndex;
```

2、通用函数模板

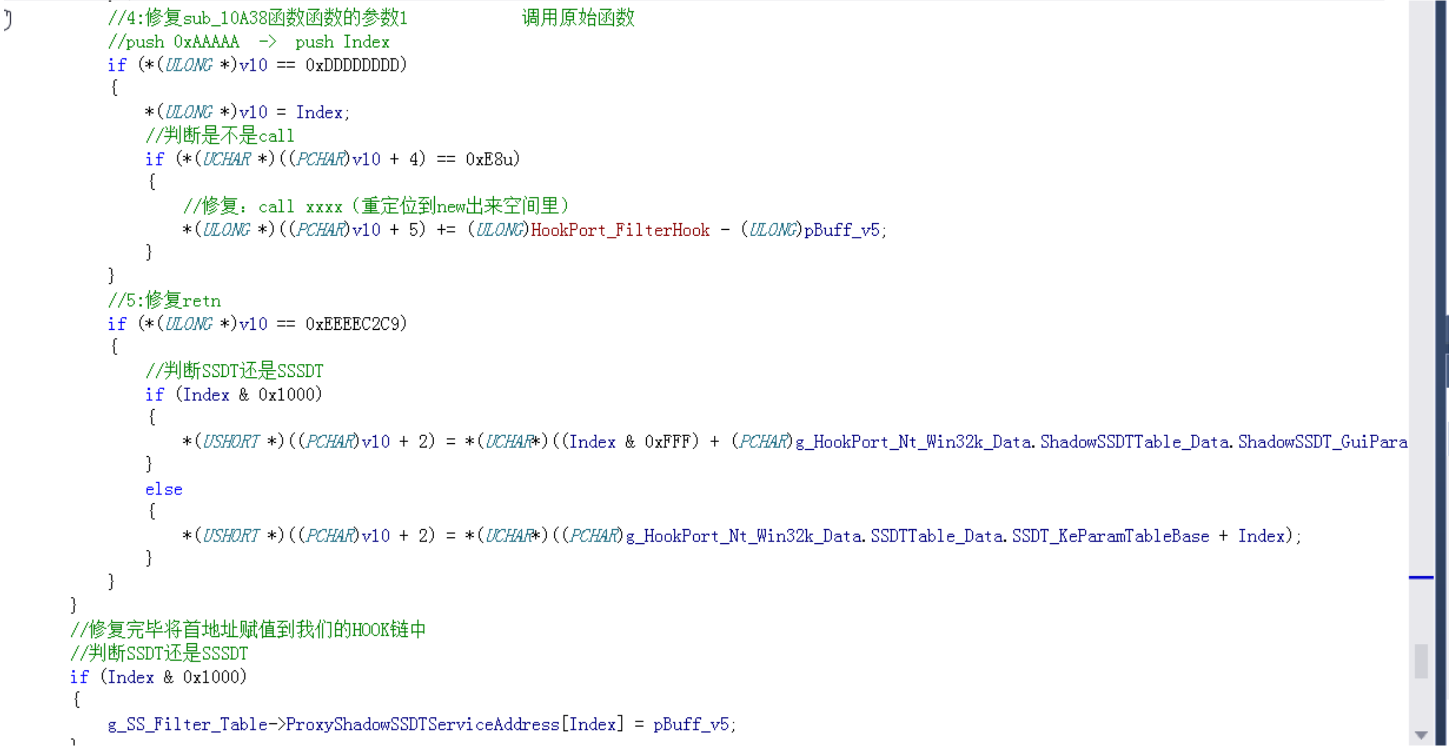


```
// 函数说明：
// 不感兴趣的函数都由这个通用处理，检查下就直接完事
int __cdecl HookPort_FilterHook(char a1)
{
  bool v1; // zf@1
  int result; // eax@1
  int (__fastcall *v3)(int, int); // eax@2
  signed int v4; // eax@4
  char v5; // [sp+0h] [bp-98h]@1
  char v6; // [sp+40h] [bp-58h]@1
  int v7; // [sp+80h] [bp-18h]@1
  int v8; // [sp+84h] [bp-14h]@1
  int v9; // [sp+88h] [bp-10h]@4
  int v10; // [sp+8Ch] [bp-Ch]@4
  int savedregs; // [sp+98h] [bp+0h]@2
  void *retaddr; // [sp+9Ch] [bp+4h]@2

  v1 = HookPort_DoFilter(0xAAAAAAAA, (int)&a1, (int)&v5, &v6, (unsigned int *)&v7, (NTSTATUS *)&v8) == 0;
  result = v8;
  if ( !v1 )
  {
    qmemcpy(&savedregs, &a1, 0x33333330u);
    v3 = (int (__fastcall *)(int, int))HookPort_GetOriginalServiceRoutine(0xDDDDDDDD);
    if ( retaddr == (void *)g_call_ring0_rtn_address && dword_1B130 )
    {
      v4 = v3(v10, v9);
      retaddr = (void *)g_call_ring0_rtn_address;
    }
    else
    {
      v4 = v3(v10, v9);
    }
    result = HookPort_ForRunFuncTable(0xAAAAAAAA, (int)&a1, v4, &v5, (int)&v6, v7);
  }
  return result;
}
```

3、修复通用函数分为5个部分：



```
//修复HookPort_FilterHook函数
for (ULONG i_v9 = 0; i_v9 < FunSize; i_v9++)
{
    PVOID v10 = (PVOID)((PCHAR)pBuff_v5 + i_v9);
    //1:修复HookPort_DoFilter函数的参数1
    //push 0xAAAAA  ->  push Index
    if (*(ULONG *)v10 == 0xAAAAAAAA)
    {
        *(ULONG *)v10 = Number;
        //判断是不是call
        if (*(UCHAR *)((PCHAR)v10 + 4) == 0xE8u)
        {
            //修复: call xxxx (重定位到new出来空间里)
            *(ULONG *)((PCHAR)v10 + 5) += (ULONG)HookPort_FilterHook - (ULONG)pBuff_v5;
        }
    }
    //2:判断要使用多大空间，然后修复sub esp, 0BBBBBBBBh->sub esp,XXXh
    //获取SSDT、SSSDT的ParamTableBase就可以确认参数个数
    if (*(ULONG *)v10 == 0xBBBBBBBB)
    {
        //判断SSDT还是SSSDT
        if (Index & 0x1000)
        {
            *(ULONG *)v10 = *(UCHAR*)((Index & 0xFFF) + (PCHAR)g_HookPort_Nt_Win32k_Data.ShadowSSDTTable_Data.ShadowSSDT_GuiParamTableBase) +
        }
        else
        {

            *(ULONG *)v10 = *(UCHAR*)((PCHAR)g_HookPort_Nt_Win32k_Data.SSDTTable_Data.SSDT_KeParamTableBase + Index) + 0x98;
        }
    }
    //3:判断要memcpy多大空间，然后修复qmemcpy(&savedregs, &a1, 0x33333330u)->qmemcpy(&savedregs, &a1, 0xXXXu);
    if (*(ULONG *)v10 == 0xCCCCCCCC)
    {
        //判断SSDT还是SSSDT
        if (Index & 0x1000)
        {
```



```
}       //4:修复sub_10A38函数函数的参数1            调用原始函数
        //push 0xAAAAA  ->  push Index
        if (*(ULONG *)v10 == 0xDDDDDDDD)
        {
            *(ULONG *)v10 = Index;
            //判断是不是call
            if (*(UCHAR *)((PCHAR)v10 + 4) == 0xE8u)
            {
                //修复: call xxxx (重定位到new出来空间里)
                *(ULONG *)((PCHAR)v10 + 5) += (ULONG)HookPort_FilterHook - (ULONG)pBuff_v5;
            }
        }
        //5:修复retn
        if (*(ULONG *)v10 == 0xEEEEC2C9)
        {
            //判断SSDT还是SSSDT
            if (Index & 0x1000)
            {
                *(USHORT *)((PCHAR)v10 + 2) = *(UCHAR*)((Index & 0xFFF) + (PCHAR)g_HookPort_Nt_Win32k_Data.ShadowSSDTTable_Data.ShadowSSDT_GuiPara
            }
            else
            {
                *(USHORT *)((PCHAR)v10 + 2) = *(UCHAR*)((PCHAR)g_HookPort_Nt_Win32k_Data.SSDTTable_Data.SSDT_KeParamTableBase + Index);
            }
        }
    }
    //修复完毕将首地址赋值到我们的HOOK链中
    //判断SSDT还是SSSDT
    if (Index & 0x1000)
    {
        g_SS_Filter_Table->ProxyShadowSSDTServiceAddress[Index] = pBuff_v5;
```

3、Hook方式（加密解密4已经有图文介绍了13章）

数字hook点:

// 保存特征指令之后的那个地址，即钩子处理之后的返回地址

```
//840541a4 2be1        sub    esp, ecx     此时address                          = 840541a4
//840541a6 c1e902      shr    ecx, 2       此时g_KiFastCallEntry_360HookPoint     = 840541a6
//840541a9 8bfc        mov    edi, esp     此时g_KiFastCallEntry_Fake_rtn_address = 840541a9
```

hook方式有两种:

IdtHook4号中断

InlineHook

4、驱动接口使用

注意HookPort只是初始化接口，是导出给N个类似于SelfProtection的驱动使用

导出接口结构体定义如下：

```
/*
// sizeof(HOOKPORT_EXTENSION) = 0x18
设备扩展包含了添加规则的接口
1、其他驱动需要增加规则时只需要获取Hookport的驱动扩展访问里面的HookPort_FilterRule_Init初始化一条规则
2、HookPort_SetFilterSwitchFunction 设置规则过滤函数
3、HookPort_SetFilterRuleFlag 设置开关表示启动or 关闭
State                                   启动标识
HookPort_FilterRule_Init                初始化规则，新建规则会加到规则链中
HookPort_SetFilterSwitchFunction        设置规则过滤函数
HookPort_SetFilterRuleFlag              设置规则开关
HookPort_SetFilterRuleName              设置规则名字
Value3F1                                该驱动版本
*/
typedef struct _HOOKPORT_EXTENSION
{
	_DWORD State;
	_DWORD HookPort_FilterRule_Init;
	_DWORD HookPort_SetFilterSwitchFunction;
	_DWORD HookPort_SetFilterRule;
	_DWORD HookPort_SetFilterRuleName;
	_DWORD Value3F1;
}HOOKPORT_EXTENSION, *PHOOKPORT_EXTENSION;
```

然后进行初始化操作

```
//初始化导出接口
ULONG NTAPI HookPort_InitDeviceExtInterface(IN PDEVICE_OBJECT DeviceObject)
{
	PHOOKPORT_EXTENSION pHookPortExt;
	pHookPortExt = DeviceObject->DeviceExtension;
	pHookPortExt->State = (PULONG)3;
	pHookPortExt->HookPort_FilterRule_Init = HookPort_AllocFilterRuleTable;                 //初始化规则
	pHookPortExt->HookPort_SetFilterSwitchFunction = HookPort_SetFilterSwitchFunction;      //设置规则过滤函数
	pHookPortExt->HookPort_SetFilterRule = HookPort_SetFilterRule;                          //设置规则开关
	pHookPortExt->HookPort_SetFilterRuleName = HookPort_SetFilterRuleName;                  //设置规则名字
	pHookPortExt->Value3F1 = 0x3F1;                                                         //版本
	return pHookPortExt;
}
```

使用：

FakeServiceRoutine[X]	Fake_xxxxx	设置单独的Fake函数
FakeServiceRoutine[X]	1	设置单独的Fake函数开关
IsFilterFunFilledReady	1	设置总开关

关闭：

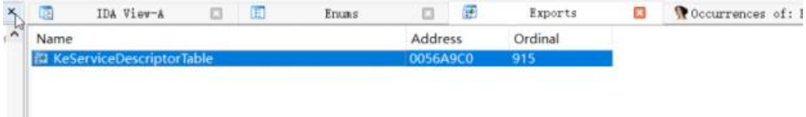
FakeServiceRoutine[X]	0	设置单独的Fake函数，随你清不清
FakeServiceRoutine[X]	0	设置单独的Fake函数开关，随你清不清 直接拉总闸省事
IsFilterFunFilledReady	0	设置总开关，嫌一个个关闭麻烦直接关这个（拉总闸直接GG）

5、HookPort_InitSDT

1、首先获取SSDT和ShadowSSDT地址

SSDT表获取方法：

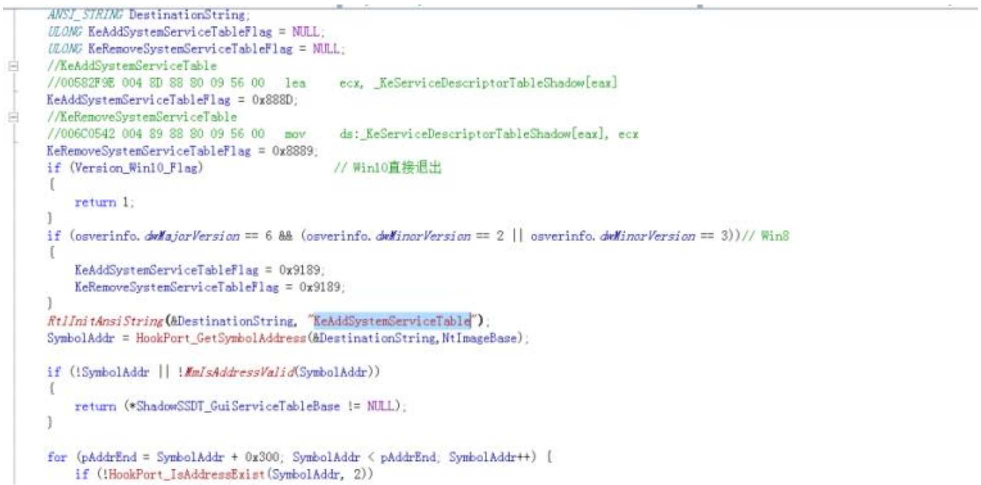
直接NT内核里面找KeServiceDescriptorTable，KeServiceDescriptorTable是导出的遍历下导出表



Name	Address	Ordinal
KeServiceDescriptorTable	0056A9C0	915

ShadowSSDT表获取方法：

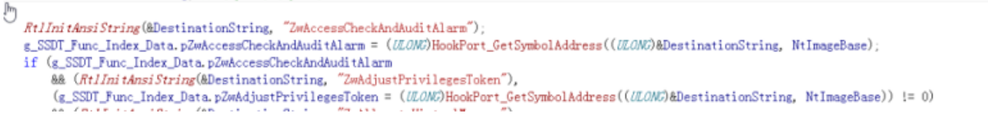
首先获取win32k基地址，然后再通过特征码查找代码如下



```
ANSI_STRING DestinationString;
ULONG KeAddSystemServiceTableFlag = NULL;
ULONG KeRemoveSystemServiceTableFlag = NULL;
//KeAddSystemServiceTable
//00582F9E 004 8D 88 80 09 56 00   lea     ecx, _KeServiceDescriptorTableShadow[eax]
KeAddSystemServiceTableFlag = 0x888D;
//KeRemoveSystemServiceTable
//006C0542 004 89 88 80 09 56 00   mov     ds:_KeServiceDescriptorTableShadow[eax], ecx
KeRemoveSystemServiceTableFlag = 0x8889;
if (Version_Win10_Flag)                    // Win10直接退出
{
    return 1;
}
if (osverinfo.dwMajorVersion == 6 && (osverinfo.dwMinorVersion == 2 || osverinfo.dwMinorVersion == 3))// Win8
{
    KeAddSystemServiceTableFlag = 0x9189;
    KeRemoveSystemServiceTableFlag = 0x9189;
}
RtlInitAnsiString(&DestinationString, "KeAddSystemServiceTable");
SymbolAddr = HookPort_GetSymbolAddress(&DestinationString,NtImageBase);

if (!SymbolAddr || !MmIsAddressValid(SymbolAddr))
{
    return (*ShadowSSDT_GuiServiceTableBase != NULL);
}

for (pAddrEnd = SymbolAddr + 0x300; SymbolAddr < pAddrEnd; SymbolAddr++) {
    if (!HookPort_IsAddressExist(SymbolAddr, 2))
```

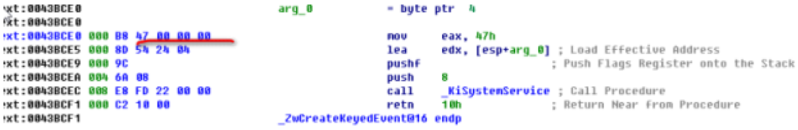
2、获取函数索引

1、通过导出表找到zwXXX的地址，然后再取对应的索引号B8+1就是索引号



```
RtlInitAnsiString(&DestinationString, "ZwAccessCheckAndAuditAlarm");
g_SSDT_Func_Index_Data.pZwAccessCheckAndAuditAlarm = (ULONG)HookPort_GetSymbolAddress((ULONG)&DestinationString, NtImageBase);
if (g_SSDT_Func_Index_Data.pZwAccessCheckAndAuditAlarm
    && (RtlInitAnsiString(&DestinationString, "ZwAdjustPrivilegesToken"),
    (g_SSDT_Func_Index_Data.pZwAdjustPrivilegesToken = (ULONG)HookPort_GetSymbolAddress((ULONG)&DestinationString, NtImageBase)) != 0)
```

2、取B8后面4个字节就是索引号

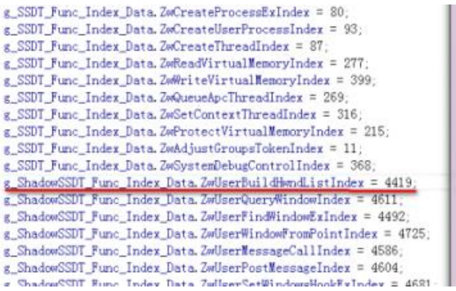


```
xt:0043BCE0                         arg_0           = byte ptr  4
xt:0043BCE0
xt:0043BCE0 000 B8 47 00 00 00                      mov     eax, 47h
xt:0043BCE5 000 8D 54 24 04                         lea     edx, [esp+arg_0] ; Load Effective Address
xt:0043BCE9 000 9C                                  pushf                    ; Push Flags Register onto the Stack
xt:0043BCEA 004 6A 08                               push    8
xt:0043BCEC 008 E8 FD 22 00 00                      call    _KiSystemService ; Call Procedure
xt:0043BCF1 000 C2 10 00                            retn    10h              ; Return Near from Procedure
xt:0043BCF1                         _ZwCreateKeyedEvent@16 endp
```

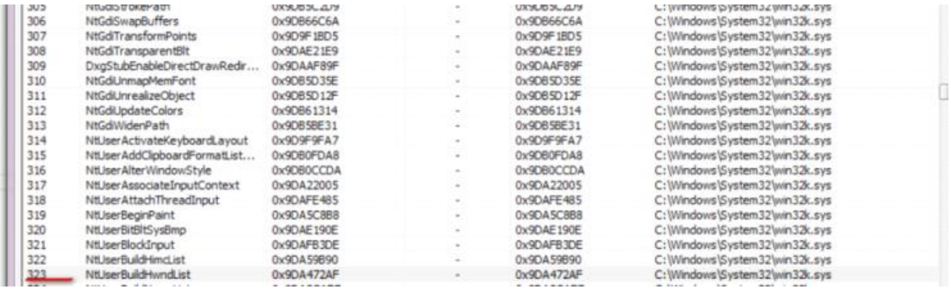
3、SSDT和ShadowSSDT区别

ShadowSSDT的索引号要-0x1000才是真正的索引

例如4419-4096=323



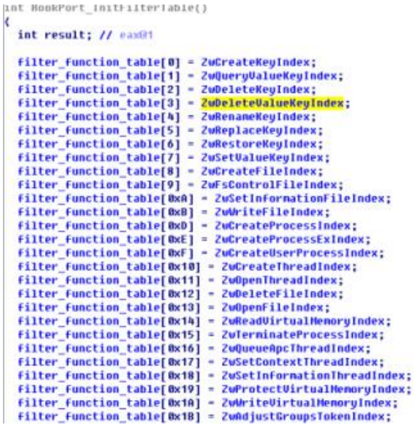
```
g_SSDT_Func_Index_Data.ZwCreateProcessExIndex = 80;
g_SSDT_Func_Index_Data.ZwCreateUserProcessIndex = 93;
g_SSDT_Func_Index_Data.ZwCreateThreadIndex = 87;
g_SSDT_Func_Index_Data.ZwReadVirtualMemoryIndex = 277;
g_SSDT_Func_Index_Data.ZwWriteVirtualMemoryIndex = 399;
g_SSDT_Func_Index_Data.ZwQueueApcThreadIndex = 269;
g_SSDT_Func_Index_Data.ZwSetContextThreadIndex = 316;
g_SSDT_Func_Index_Data.ZwProtectVirtualMemoryIndex = 215;
g_SSDT_Func_Index_Data.ZwAdjustGroupsTokenIndex = 11;
g_SSDT_Func_Index_Data.ZwSystemDebugControlIndex = 368;
g_ShadowSSDT_Func_Index_Data.ZwUserBuildHwndListIndex = 4419;
g_ShadowSSDT_Func_Index_Data.ZwUserQueryWindowIndex = 4611;
g_ShadowSSDT_Func_Index_Data.ZwUserFindWindowExIndex = 4492;
g_ShadowSSDT_Func_Index_Data.ZwUserWindowFromPointIndex = 4725;
g_ShadowSSDT_Func_Index_Data.ZwUserMessageCallIndex = 4586;
g_ShadowSSDT_Func_Index_Data.ZwUserPostMessageIndex = 4604;
g_ShadowSSDT_Func_Index_Data.ZwUserSetWindowsHookExIndex = 4681;
```



305	NtGdiStrokePath	0x9DB5C2D9	-	0x9DB5C2D9	C:\Windows\System32\win32k.sys
306	NtGdiSwapBuffers	0x9DB66C6A	-	0x9DB66C6A	C:\Windows\System32\win32k.sys
307	NtGdiTransformPoints	0x9D9F1BD5	-	0x9D9F1BD5	C:\Windows\System32\win32k.sys
308	NtGdiTransparentBlt	0x9DAE21E9	-	0x9DAE21E9	C:\Windows\System32\win32k.sys
309	DxgStubEnableDirectDrawRedir...	0x9DAAF89F	-	0x9DAAF89F	C:\Windows\System32\win32k.sys
310	NtGdiUnmapMemFont	0x9DB5D35E	-	0x9DB5D35E	C:\Windows\System32\win32k.sys
311	NtGdiUnrealizeObject	0x9DB5D12F	-	0x9DB5D12F	C:\Windows\System32\win32k.sys
312	NtGdiUpdateColors	0x9DB61314	-	0x9DB61314	C:\Windows\System32\win32k.sys
313	NtGdiWidenPath	0x9DB5BE31	-	0x9DB5BE31	C:\Windows\System32\win32k.sys
314	NtUserActivateKeyboardLayout	0x9D9F9FA7	-	0x9D9F9FA7	C:\Windows\System32\win32k.sys
315	NtUserAddClipboardFormatList...	0x9DB0FDA8	-	0x9DB0FDA8	C:\Windows\System32\win32k.sys
316	NtUserAlterWindowStyle	0x9DB0CCDA	-	0x9DB0CCDA	C:\Windows\System32\win32k.sys
317	NtUserAssociateInputContext	0x9DA22005	-	0x9DA22005	C:\Windows\System32\win32k.sys
318	NtUserAttachThreadInput	0x9DAFE485	-	0x9DAFE485	C:\Windows\System32\win32k.sys
319	NtUserBeginPaint	0x9DA5C8B8	-	0x9DA5C8B8	C:\Windows\System32\win32k.sys
320	NtUserBitBltSysBmp	0x9DAE190E	-	0x9DAE190E	C:\Windows\System32\win32k.sys
321	NtUserBlockInput	0x9DAFB3DE	-	0x9DAFB3DE	C:\Windows\System32\win32k.sys
322	NtUserBuildHimcList	0x9DA59B90	-	0x9DA59B90	C:\Windows\System32\win32k.sys
323	NtUserBuildHwndList	0x9DA472AF	-	0x9DA472AF	C:\Windows\System32\win32k.sys

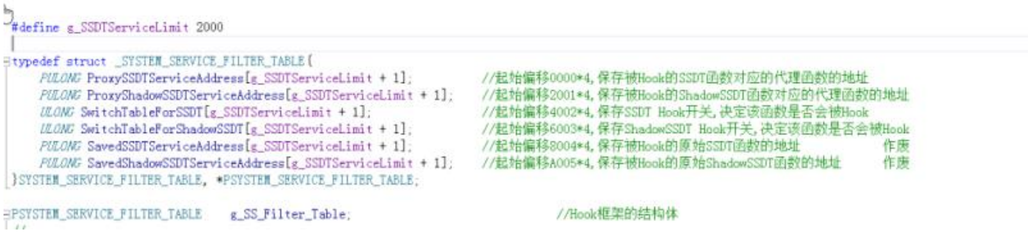
3、填写函数过滤数组

该数组一共有0x9E个



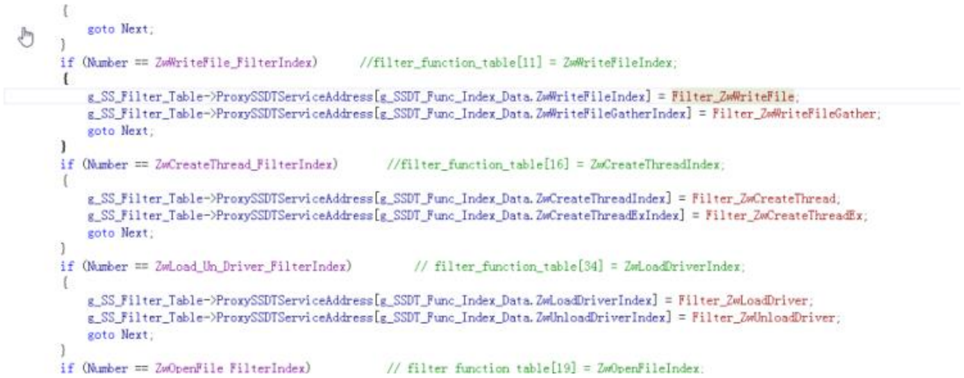
```
int HookPort_InitFilterTable()
{
  int result; // eax@1

  filter_function_table[0] = ZwCreateKeyIndex;
  filter_function_table[1] = ZwQueryValueKeyIndex;
  filter_function_table[2] = ZwDeleteKeyIndex;
  filter_function_table[3] = ZwDeleteValueKeyIndex;
  filter_function_table[4] = ZwRenameKeyIndex;
  filter_function_table[5] = ZwReplaceKeyIndex;
  filter_function_table[6] = ZwRestoreKeyIndex;
  filter_function_table[7] = ZwSetValueKeyIndex;
  filter_function_table[8] = ZwCreateFileIndex;
  filter_function_table[9] = ZwFsControlFileIndex;
  filter_function_table[0xA] = ZwSetInformationFileIndex;
  filter_function_table[0xB] = ZwWriteFileIndex;
  filter_function_table[0xD] = ZwCreateProcessIndex;
  filter_function_table[0xE] = ZwCreateProcessExIndex;
  filter_function_table[0xF] = ZwCreateUserProcessIndex;
  filter_function_table[0x10] = ZwCreateThreadIndex;
  filter_function_table[0x11] = ZwOpenThreadIndex;
  filter_function_table[0x12] = ZwDeleteFileIndex;
  filter_function_table[0x13] = ZwOpenFileIndex;
  filter_function_table[0x14] = ZwReadVirtualMemoryIndex;
  filter_function_table[0x15] = ZwTerminateProcessIndex;
  filter_function_table[0x16] = ZwQueueApcThreadIndex;
  filter_function_table[0x17] = ZwSetContextThreadIndex;
  filter_function_table[0x18] = ZwSetInformationThreadIndex;
  filter_function_table[0x19] = ZwProtectVirtualMemoryIndex;
  filter_function_table[0x1A] = ZwWriteVirtualMemoryIndex;
  filter_function_table[0x1B] = ZwAdjustGroupsTokenIndex;
```

4、申请一块缓冲区专门存放过滤函数开关、代理函数等地址，大小是0x7D10



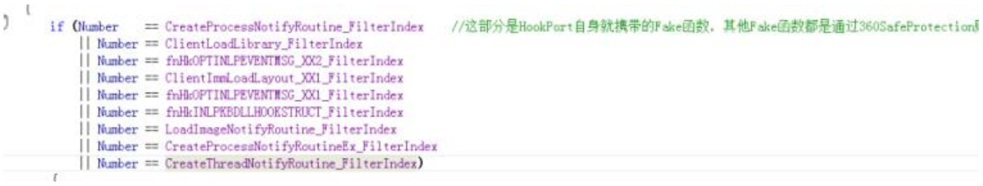
```
#define g_SSDTServiceLimit 2000

typedef struct _SYSTEM_SERVICE_FILTER_TABLE{
    PULONG ProxySSDTServiceAddress[g_SSDTServiceLimit + 1];          //起始偏移0000*4,保存被Hook的SSDT函数对应的代理函数的地址
    PULONG ProxyShadowSSDTServiceAddress[g_SSDTServiceLimit + 1];    //起始偏移2001*4,保存被Hook的ShadowSSDT函数对应的代理函数的地址
    ULONG SwitchTableForSSDT[g_SSDTServiceLimit + 1];               //起始偏移4002*4,保存SSDT Hook开关,决定该函数是否会被Hook
    ULONG SwitchTableForShadowSSDT[g_SSDTServiceLimit + 1];         //起始偏移6003*4,保存ShadowSSDT Hook开关,决定该函数是否会被Hook
    PULONG SavedSSDTServiceAddress[g_SSDTServiceLimit + 1];          //起始偏移8004*4,保存被Hook的原始SSDT函数的地址          作废
    PULONG SavedShadowSSDTServiceAddress[g_SSDTServiceLimit + 1];    //起始偏移A005*4,保存被Hook的原始ShadowSSDT函数的地址    作废
}SYSTEM_SERVICE_FILTER_TABLE, *PSYSTEM_SERVICE_FILTER_TABLE;

PSYSTEM_SERVICE_FILTER_TABLE    g_SS_Filter_Table;                          //Hook框架的结构体
```

1、接下来执行HookPort_InitProxyAddress函数填充该结构，感兴趣的自己单独写一个代理函数，不感兴趣的直接通用模板处理即可。



```
    {
        goto Next;
    }
    if (Number == ZwWriteFile_FilterIndex)      //filter_function_table[11] = ZwWriteFileIndex;
    {
        g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwWriteFileIndex] = Filter_ZwWriteFile;
        g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwWriteFileGatherIndex] = Filter_ZwWriteFileGather;
        goto Next;
    }
    if (Number == ZwCreateThread_FilterIndex)       //filter_function_table[16] = ZwCreateThreadIndex;
    {
        g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwCreateThreadIndex] = Filter_ZwCreateThread;
        g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwCreateThreadExIndex] = Filter_ZwCreateThreadEx;
        goto Next;
    }
    if (Number == ZwLoad_Un_Driver_FilterIndex)         // filter_function_table[34] = ZwLoadDriverIndex;
    {
        g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwLoadDriverIndex] = Filter_ZwLoadDriver;
        g_SS_Filter_Table->ProxySSDTServiceAddress[g_SSDT_Func_Index_Data.ZwUnloadDriverIndex] = Filter_ZwUnloadDriver;
        goto Next;
    }
    if (Number == ZwOpenFile_FilterIndex)       // filter_function_table[19] = ZwOpenFileIndex;
```

2、注意我们要过滤掉以下几个函数，因为他们由HookPort实现Fake函数

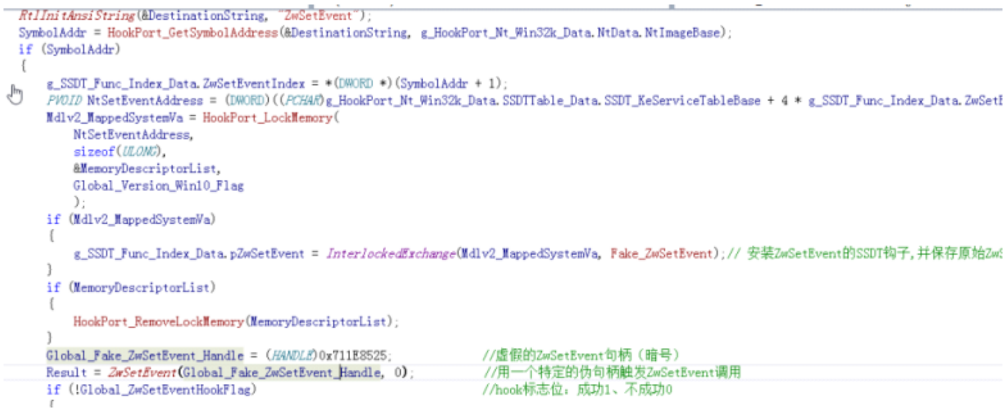


```
    {
    if (Number    == CreateProcessNotifyRoutine_FilterIndex    //这部分是HookPort自身就携带的Fake函数，其他Fake函数都是通过360SafeProtection
        || Number == ClientLoadLibrary_FilterIndex
        || Number == fnHkOPTINLPEVENTMSG_XX2_FilterIndex
        || Number == ClientImmLoadLayout_XX1_FilterIndex
        || Number == fnHkOPTINLPEVENTMSG_XX1_FilterIndex
        || Number == fnHkINLPKBDLLHOOKSTRUCT_FilterIndex
        || Number == LoadImageNotifyRoutine_FilterIndex
        || Number == CreateProcessNotifyRoutineEx_FilterIndex
        || Number == CreateThreadNotifyRoutine_FilterIndex)
    {
```

5、KiFastCallEntry处理

1、通过修改SSDT表的ZwSetEvent来触发安装钩子

设置一个虚假的ZwSetEvent句柄来触发，注意保存原始函数地址后面要恢复



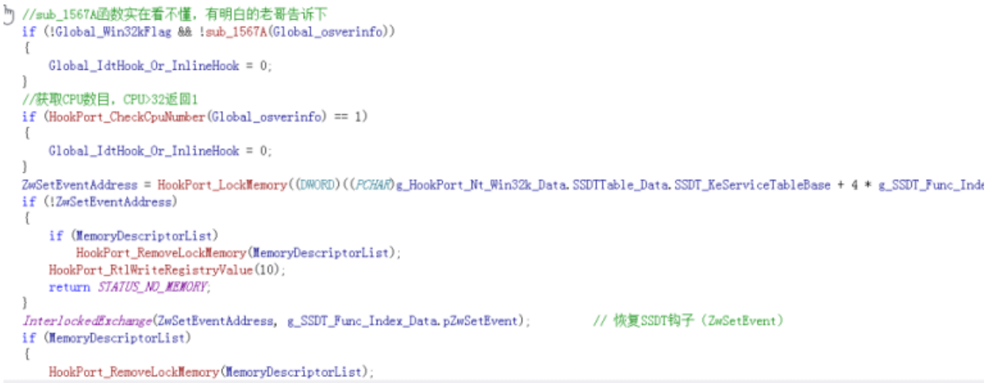
```
RtlInitAnsiString(&DestinationString, "ZwSetEvent");
SymbolAddr = HookPort_GetSymbolAddress(&DestinationString, g_HookPort_Nt_Win32k_Data.NtData.NtImageBase);
if (SymbolAddr)
{
    g_SSDT_Func_Index_Data.ZwSetEventIndex = *(DWORD *)(SymbolAddr + 1);
    PVOID NtSetEventAddress = (DWORD)((PCHAR)g_HookPort_Nt_Win32k_Data.SSDTTable_Data.SSDT_KeServiceTableBase + 4 * g_SSDT_Func_Index_Data.ZwSetE
    Mdlv2_MappedSystemVa = HookPort_LockMemory(
        NtSetEventAddress,
        sizeof(ULONG),
        &MemoryDescriptorList,
        Global_Version_Win10_Flag
        );
    if (Mdlv2_MappedSystemVa)
    {
        g_SSDT_Func_Index_Data.pZwSetEvent = InterlockedExchange(Mdlv2_MappedSystemVa, Fake_ZwSetEvent);// 安装ZwSetEvent的SSDT钩子,并保存原始Zw
    }
    if (MemoryDescriptorList)
    {
        HookPort_RemoveLockMemory(MemoryDescriptorList);
    }
    Global_Fake_ZwSetEvent_Handle = (HANDLE)0x711E8525;             //虚假的ZwSetEvent句柄（暗号）
    Result = ZwSetEvent(Global_Fake_ZwSetEvent_Handle, 0);          //用一个特定的伪句柄触发ZwSetEvent调用
    if (!Global_ZwSetEventHookFlag)                                 //hook标志位：成功1、不成功0
    {
```

5、1 Fake_ZwSetEvent处理部分

1、判断Hook方式，默认Global_IdtHook_Or_InlineHook置1，并且恢复SSDT钩子（ZwSetEvent）

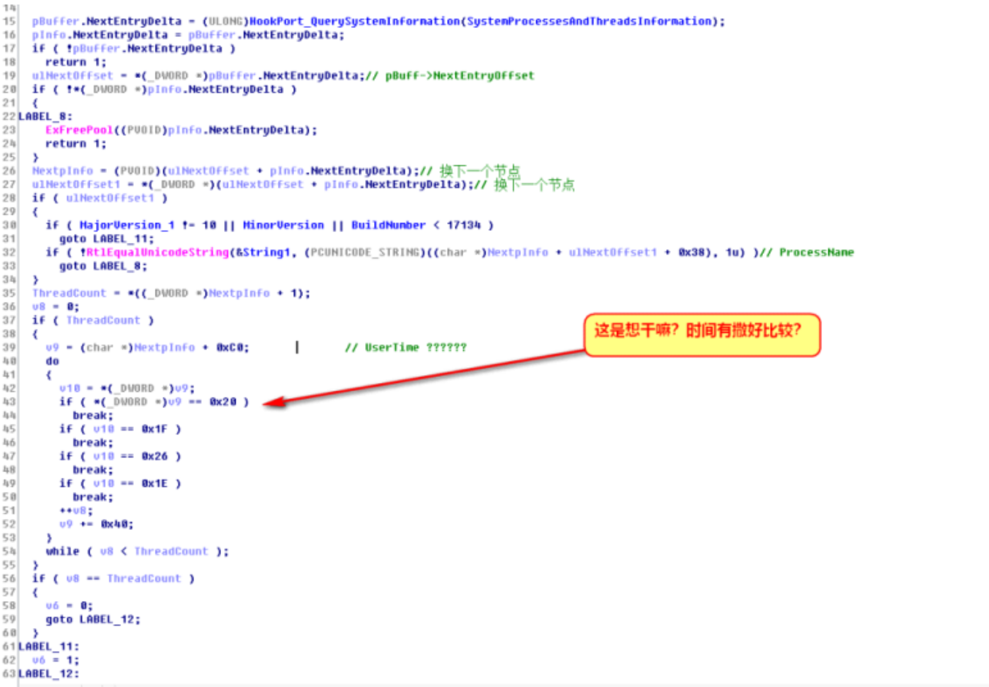
修改：Jmpxxx　　Global_IdtHook_Or_InlineHook == 0

修改：Int 4　　Global_IdtHook_Or_InlineHook == 1



```
//sub_1567A函数实在看不懂，有明白的老哥告诉下
if (!Global_Win32kFlag && !sub_1567A(Global_osverinfo))
{
    Global_IdtHook_Or_InlineHook = 0;
}
//获取CPU数目，CPU>32返回1
if (HookPort_CheckCpuNumber(Global_osverinfo) == 1)
{
    Global_IdtHook_Or_InlineHook = 0;
}
ZwSetEventAddress = HookPort_LockMemory((DWORD)((PCHAR)g_HookPort_Nt_Win32k_Data.SSDTTable_Data.SSDT_KeServiceTableBase + 4 * g_SSDT_Func_Inde
if (!ZwSetEventAddress)
{
    if (MemoryDescriptorList)
        HookPort_RemoveLockMemory(MemoryDescriptorList);
    HookPort_RtlWriteRegistryValue(10);
    return STATUS_NO_MEMORY;
}
InterlockedExchange(ZwSetEventAddress, g_SSDT_Func_Index_Data.pZwSetEvent);         // 恢复SSDT钩子（ZwSetEvent)
if (MemoryDescriptorList)
{
    HookPort_RemoveLockMemory(MemoryDescriptorList);
```

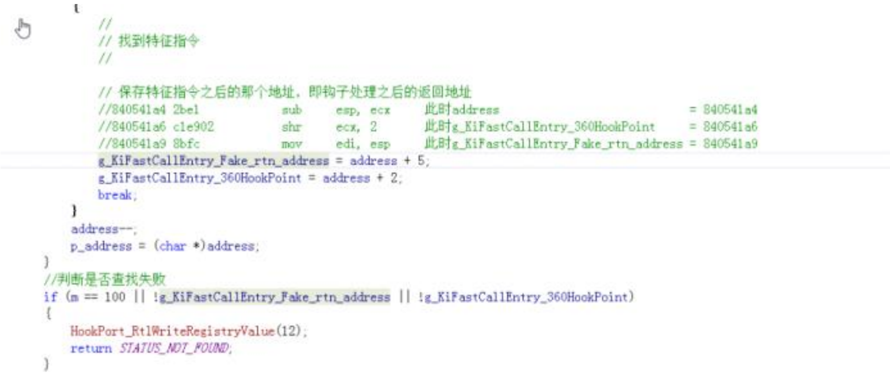
2、sub_1567A实在没看懂，有明白的老哥告诉下



```
  pBuffer.NextEntryDelta = (ULONG)HookPort_QuerySystemInformation(SystemProcessesAndThreadsInformation);
  pInfo.NextEntryDelta = pBuffer.NextEntryDelta;
  if ( !pBuffer.NextEntryDelta )
    return 1;
  ulNextOffset = *(_DWORD *)pBuffer.NextEntryDelta;// pBuff->NextEntryOffset
  if ( !*(_DWORD *)pInfo.NextEntryDelta )
  {
LABEL_8:
    ExFreePool((PVOID)pInfo.NextEntryDelta);
    return 1;
  }
  NextpInfo = (PVOID)(ulNextOffset + pInfo.NextEntryDelta);// 换下一个节点
  ulNextOffset1 = *(_DWORD *)(ulNextOffset + pInfo.NextEntryDelta);// 换下一个节点
  if ( ulNextOffset1 )
  {
    if ( MajorVersion_1 != 10 || MinorVersion || BuildNumber < 17134 )
      goto LABEL_11;
    if ( !RtlEqualUnicodeString(&String1, (PCUNICODE_STRING)((char *)NextpInfo + ulNextOffset1 + 0x38), 1u) )// ProcessName
      goto LABEL_8;
  }
  ThreadCount = *((_DWORD *)NextpInfo + 1);
  v8 = 0;
  if ( ThreadCount )
  {
    v9 = (char *)NextpInfo + 0xC0;        |        // UserTime ??????
    do
    {
      v10 = *(_DWORD *)v9;
      if ( *(_DWORD *)v9 == 0x20 )
        break;
      if ( v10 == 0x1F )
        break;
      if ( v10 == 0x26 )
        break;
      if ( v10 == 0x1E )
        break;
      ++v8;
      v9 += 0x40;
    }
    while ( v8 < ThreadCount );
  }
  if ( v8 == ThreadCount )
  {
    v6 = 0;
    goto LABEL_12;
  }
LABEL_11:
  v6 = 1;
LABEL_12:
```

这是想干嘛？时间有撒好比较？

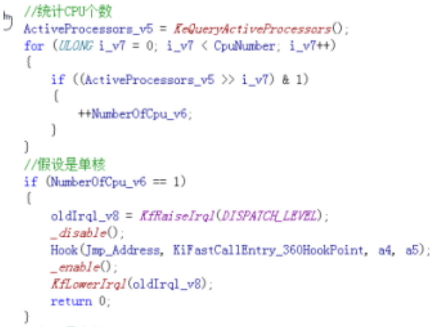
3、栈回溯获取返回地址[EBP+4]（这里指的是正常调用时返回到KiFastCallEntry中的地址），找到hook点



```
        {
            //
            // 找到特征指令
            //

            // 保存特征指令之后的那个地址，即钩子处理之后的返回地址
            //840541a4 2be1            sub     esp, ecx      此时address                            = 840541a4
            //840541a6 c1e902          shr     ecx, 2        此时g_KiFastCallEntry_360HookPoint      = 840541a6
            //840541a9 8bfc            mov     edi, esp      此时g_KiFastCallEntry_Fake_rtn_address = 840541a9
            g_KiFastCallEntry_Fake_rtn_address = address + 5;
            g_KiFastCallEntry_360HookPoint = address + 2;
            break;
        }
        address--;
        p_address = (char *)address;
    }
    //判断是否查找失败
    if (m == 100 || !g_KiFastCallEntry_Fake_rtn_address || !g_KiFastCallEntry_360HookPoint)
    {
        HookPort_RtlWriteRegistryValue(12);
        return STATUS_NOT_FOUND;
    }
```

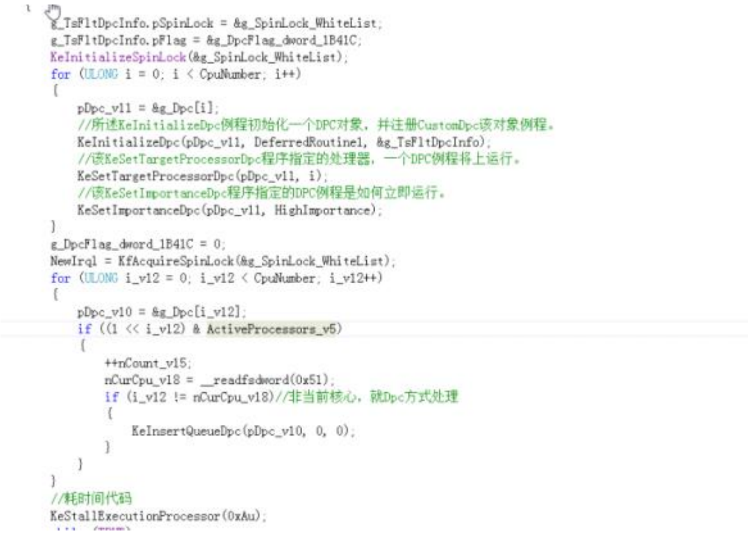
4、多核Hook方法

假设是单核直接替换即可



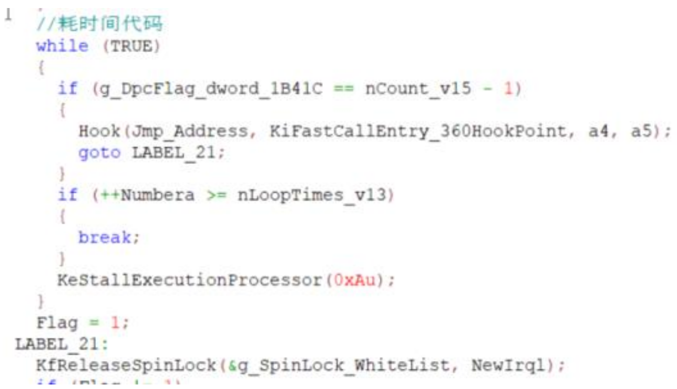
```
    //统计CPU个数
    ActiveProcessors_v5 = KeQueryActiveProcessors();
    for (ULONG i_v7 = 0; i_v7 < CpuNumber; i_v7++)
    {
        if ((ActiveProcessors_v5 >> i_v7) & 1)
        {
            ++NumberOfCpu_v6;
        }
    }
    //假设是单核
    if (NumberOfCpu_v6 == 1)
    {
        oldIrql_v8 = KfRaiseIrql(DISPATCH_LEVEL);
        _disable();
        Hook(Jmp_Address, KiFastCallEntry_360HookPoint, a4, a5);
        _enable();
        KfLowerIrql(oldIrql_v8);
        return 0;
    }
```

假设是多核就采用DPC方式处理即可



```
    {
        g_TsFltDpcInfo.pSpinLock = &g_SpinLock_WhiteList;
        g_TsFltDpcInfo.pFlag = &g_DpcFlag_dword_1B41C;
        KeInitializeSpinLock(&g_SpinLock_WhiteList);
        for (ULONG i = 0; i < CpuNumber; i++)
        {
            pDpc_v11 = &g_Dpc[i];
            //所述KeInitializeDpc例程初始化一个DPC对象，并注册CustomDpc该对象例程。
            KeInitializeDpc(pDpc_v11, DeferredRoutine1, &g_TsFltDpcInfo);
            //该KeSetTargetProcessorDpc程序指定的处理器，一个DPC例程将上运行。
            KeSetTargetProcessorDpc(pDpc_v11, i);
            //该KeSetImportanceDpc程序指定的DPC例程是如何立即运行。
            KeSetImportanceDpc(pDpc_v11, HighImportance);
        }
        g_DpcFlag_dword_1B41C = 0;
        NewIrql = KfAcquireSpinLock(&g_SpinLock_WhiteList);
        for (ULONG i_v12 = 0; i_v12 < CpuNumber; i_v12++)
        {
            pDpc_v10 = &g_Dpc[i_v12];
            if ((1 << i_v12) & ActiveProcessors_v5)
            {
                ++nCount_v15;
                nCurCpu_v18 = __readfsdword(0x51);
                if (i_v12 != nCurCpu_v18)//非当前核心，就Dpc方式处理
                {
                    KeInsertQueueDpc(pDpc_v10, 0, 0);
                }
            }
        }
        //耗时间代码
        KeStallExecutionProcessor(0xAu);
```

暂停N一段时间处理hook



```
  //耗时间代码
  while (TRUE)
  {
    if (g_DpcFlag_dword_1B41C == nCount_v15 - 1)
    {
      Hook(Jmp_Address, KiFastCallEntry_360HookPoint, a4, a5);
      goto LABEL_21;
    }
    if (++Numbera >= nLoopTimes_v13)
    {
      break;
    }
    KeStallExecutionProcessor(0xAu);
  }
  Flag = 1;
LABEL_21:
  KfReleaseSpinLock(&g_SpinLock_WhiteList, NewIrql);
```

6、创建进程、线程、模块回调和IAT方式挂钩KeUserModeCallback

ClientLoadLibrary加载模块相关

ClientImmLoadLayout加载模块相关

fnHkOPTINLPEVENTMSG未知

fnHkINLPKBDLLHOOKSTRUCT拦截键盘消息