

56-INTRODUCCION AL REVERSING CON IDA PRO - KERNEL EXPLOITATION.

Contents

56-INTRODUCCION AL REVERSING CON IDA PRO - KERNEL EXPLOITATION.....	1
HACKSYS VULNERABLE DRIVER.....	1
Vulnerabilities Implemented	1

HACKSYS VULNERABLE DRIVER.

Vamos a ver ahora un driver que esta programado con diferentes vulnerabilidades para entender como explotarlas, como siempre por ahora usaremos un Windows 7 sp1 sin ningún parche de seguridad, sabemos que allí funcionara todo, luego iremos viendo que cambios hay más adelante y que otras posibilidades existen en los nuevos sistemas, pero vamos paso a paso.

Tenemos un driver compilado con los símbolos por ahora que tiene la posibilidad de explotarlo de casi todas las formas posibles, está hecho para practicar.

<https://github.com/hacksys/HackSysExtremeVulnerableDriver>

Vulnerabilities Implemented

- Double Fetch
- Pool Overflow
- Use After Free
- Type Confusion
- Stack Overflow
- Integer Overflow
- Stack Overflow GS
- Arbitrary Overwrite
- Null Pointer Dereference
- Uninitialized Heap Variable
- Uninitialized Stack Variable

- Insecure Kernel Resource Access

Empezaremos poco a poco primero con el análisis del stack overflow.

Por supuesto hay que copiar el driver a la maquina target y cargarlo con el OSR DRIVER LOADER.

Lo copiamos con su idb a una carpeta local y lo abrimos en IDA para ir analizando.

Bueno como ya sabemos acá tenemos símbolos, eso nos facilita mucho las cosas, pero igual lo primero que debemos buscar y que casi siempre es reconocido con o sin símbolos es la estructura `_DRIVER_OBJECT` que se pasa como argumento al `DriverEntry`.

En este caso no hay demasiado problema

```

00016006 ; Attributes: bp-based frame
00016006
00016006 ; int __stdcall DriverEntry(_DRIVER_OBJECT *DriverObject, _UNICODE_STRING *RegistryPath)
00016006 _DriverEntry@08 proc near
00016006
00016006 DeviceName = _UNICODE_STRING ptr -14h
00016006 DosDeviceName = _UNICODE_STRING ptr -0Ch
00016006 DeviceObject = dword ptr -4
00016006 DriverObject = dword ptr 8
00016006 RegistryPath = dword ptr 0Ch
00016006
00016006 mov     edi, edi
00016008 push    ebp
00016009 mov     ebp, esp
0001600B sub     esp, 14h
0001600E and     [ebp+DeviceObject], 0
00016012 push    esi
00016013 mov     esi, ds: __imp__RtlInitUnicodeString@0 ; RtlInitUnicodeString(x,x)
00016019 push    edi
0001601A xor     eax, eax
0001601C mov     [ebp+DosDeviceName.Length], ax
00016020 lea     edi, [ebp+DosDeviceName.MaximumLength]
00016023 stosd
00016024 stosw
00016026 push    offset aDeviceHacksys ; "\\Device\\HackSysExtremeVulnerableDrive"...
0001602B lea     eax, [ebp+DeviceName]
0001602E push    eax ; DestinationString
0001602F call    esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
00016031 push    offset aDosDevicesHack_0 ; "\\DosDevices\\HackSysExtremeVulnerable"...
00016036 lea     eax, [ebp+DosDeviceName]
00016039 push    eax ; DestinationString
0001603A call    esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)

```

Acá está bien a la vista el punto de entrada y sus argumentos están bien detectados.

Vemos que usa como en los ejemplos anteriores la api `RtlInitUnicodeString` para inicializar las estructuras

RtlInitUnicodeString routine

For more information, see the [WdmlibRtlInitUnicodeStringEx](#) function.

Syntax

```
C++
VOID RtlInitUnicodeString(
    _Out_ PUNICODE_STRING DestinationString,
    _In_opt_ PCWSTR SourceString
);
```

Parameters

DestinationString [out]

For more information, see the [WdmlibRtlInitUnicodeStringEx](#) function.

SourceString [in, optional]

For more information, see the [WdmlibRtlInitUnicodeStringEx](#) function.

Return value

Recordemos que el primer argumento era un puntero a la estructura UNICODE_STRING, allí vemos PUNICODE_STRING o sea puntero a estructura UNICODE_STRING.

UNICODE_STRING structure

The **UNICODE_STRING** structure is used by various [Local Security Authority](#) (LSA) functions to specify a *Unicode* string.

Syntax

```
C++
typedef struct _LSA_UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR Buffer;
} LSA_UNICODE_STRING, *PLSA_UNICODE_STRING, UNICODE_STRING, *PUNICODE_STRING;
```

Members

Length

Specifies the length, in bytes, of the string pointed to by the **Buffer** member, not including the terminating **NULL** character, if any.

Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP: When the **Length** structure member is zero and the **MaximumLength** structure member is 1, the **Buffer** structure member can be an empty string or contain solely a null character. This behavior changed beginning with Windows Server 2008 R2 and Windows 7 with SP1.

MaximumLength

Specifies the total size, in bytes, of memory allocated for **Buffer**. Up to **MaximumLength** bytes may be written into the buffer without trampling memor

Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP: When the **Length** structure member is zero and the **MaximumLength** structure member is 1, the **Buffer** structure member can be an empty string or contain solely a null character. This behavior changed beginning with Windows Server 2008 R2 and Windows 7 with SP1.

Buffer

Pointer to a wide-character string. Note that the strings returned by the various LSA functions might not be null-terminated.

O sea, es un buffer donde guardara el largo en un word, el máximo largo en otro campo del tipo word y copiara el puntero a la string unicode que le pasamos como source a continuación en el tercer campo.

```

00000000
00000000 _UNICODE_STRING struc ; (sizeof=0x8, align=0x4, copyof_116)
00000000 ; XREF: .data:WdfDriverStubRegistryPath/r
00000000 ; _DriverEntry@8/r ...
00000000 Length dw ? ; XREF: _DriverEntry@8_0+22/w
00000002 MaximumLength dw ? ; XREF: _DriverEntry@8_0+39/w
00000004 Buffer dd ? ; XREF: _DriverEntry@8_0+3F/w ; offset
00000008 _UNICODE_STRING ends
00000000

```

Primero inicializa a cero la variable DosDeviceName que también es del tipo UNICODE_STRING.

```

00016019 push    edi
0001601A xor     eax, eax
0001601C mov     [ebp+DosDeviceName.Length], ax
00016020 lea     edi, [ebp+DosDeviceName.MaximumLength]
00016023 stosd
00016024 stosw
00016026 push    offset aDeviceHackSysExtremeVulnerableDrive...

```

Pone a cero el campo Length moviendo AX que vale 0 allí, y luego STOSD copia el valor de EAX o sea pone a cero la direccion donde apunta EDI o sea en el campo MaximumLength y luego otro STOSW copia AX o sea cero en los dos bytes siguientes o sea poniendo 6 bytes a cero inicializa los dos campos restantes de la estructura que ocupan 6 bytes (1 WORD y un DWORD)

El compilador solo inicializa la variable DosDeviceName la otra que se llama DeviceName no la pone a cero, la usa directamente.

```

0001601C mov     [ebp+DosDeviceName.Length], ax
00016020 lea     edi, [ebp+DosDeviceName.MaximumLength]
00016023 stosd
00016024 stosw
00016026 push    offset aDeviceHackSysExtremeVulnerableDrive...
0001602B lea     eax, [ebp+DeviceName]
0001602E push    eax ; DestinationString
0001602F call    esi ; RtlInitUnicodeString(x,x)

```

O sea que DeviceName es la string esa convertida a tipo estructura UNICODE_STRING, o sea que en los tres campos estará el largo, el máximo largo y el puntero que le pasamos a la string source se copiara al tercer campo.

```

INIT:00016370 ; const word _t_aDeviceHackSys
INIT:00016398 aDeviceHackSys: ; DATA XREF: DriverEntry(x,x)+20fo
INIT:00016398 unicode 0, <\Device\HackSysExtremeVulnerableDriver>,0
INIT:000163E6 align 4
INIT:000163E8 __IMPORT_DESCRIPTOR_ntoskrnl dd rva off_16410 ; Import Name Table
INIT:000163EC dd 0 ; Time stamp
INIT:000163F0 dd 0 ; Forwarder Chain
INIT:000163F4 dd rva aNtoskrnl_exe ; DLL Name
INIT:000163F8 dd rva __imp__DbgPrint ; Import Address Table
INIT:000163FC null import descriptor dd 5 dup(0)

```

En mi maquina esta en 0x0016938, ese offset lo copiara al tercer campo de la estructura.

```

0001602F call esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
00016031 push offset aDosdevicesHa_0 ; "\\DosDevices\HackSysExtremeVulnerableD"...
00016036 lea eax, [ebp+DosDeviceName]
00016039 push eax ; DestinationString
0001603A call esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)

```

En DosDeviceName armara la otra UNICODE_STRING usando como source la esa otra string.

Despues viene la llamada a IoCreateDevice, recordemos que había que crear un Device Object para poder comunicarse desde los programas en modo user.

```

0001601C mov [ebp+DosDeviceName.Length], ax
00016020 lea edi, [ebp+DosDeviceName.MaximumLength]
00016023 stosd
00016024 stosw
00016026 push offset aDeviceHackSys ; "\\Device\HackSysExtremeVulnerableDrive"...
00016028 lea eax, [ebp+DeviceName]
0001602E push eax ; DestinationString
0001602F call esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
00016031 push offset aDosdevicesHa_0 ; "\\DosDevices\HackSysExtremeVulnerableD"...
00016036 lea eax, [ebp+DosDeviceName]
00016039 push eax ; DestinationString
0001603A call esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
0001603C mov esi, [ebp+DriverObject]
0001603F lea eax, [ebp+DeviceObject]
00016042 push eax ; DeviceObject
00016043 push 0 ; Exclusive
00016045 push 100h ; DeviceCharacteristics
0001604A push 22h ; DeviceType
0001604C lea eax, [ebp+DeviceName]
0001604F push eax ; DeviceName
00016050 push 0 ; DeviceExtensionSize
00016052 push esi ; DriverObject
00016053 call ds: __imp__IoCreateDevice024 ; IoCreateDevice(x,x,x,x,x,x,x)
00016059 mov edi, eax
0001605B test edi, edi
0001605D jge short loc_1607C

```

IoCreateDevice routine

The **IoCreateDevice** routine creates a device object for use by a driver.

Syntax

```
C++  
  
NTSTATUS IoCreateDevice(  
    _In_ PDRIVER_OBJECT DriverObject,  
    _In_ ULONG DeviceExtensionSize,  
    _In_opt_ PUNICODE_STRING DeviceName,  
    _In_ DEVICE_TYPE DeviceType,  
    _In_ ULONG DeviceCharacteristics,  
    _In_ BOOLEAN Exclusive,  
    _Out_ PDEVICE_OBJECT *DeviceObject  
);
```

Parameters

DriverObject [in]

Pointer to the driver object for the caller. Each driver receives a pointer to its driver object in a parameter to its **DriverEntry** routine. WDM function and filter drivers also receive a driver object pointer in their **AddDevice** routines.

DeviceExtensionSize [in]

Specifies the driver-determined number of bytes to be allocated for the **device extension** of the device object. The internal structure of the device extension is driver-defined.

DeviceName [in, optional]

Esto estará casi siempre cerca del punto de entrada en la mayoría de los drivers que interactúan con programas en modo user.

Specifying Exclusive Access to Device Objects.

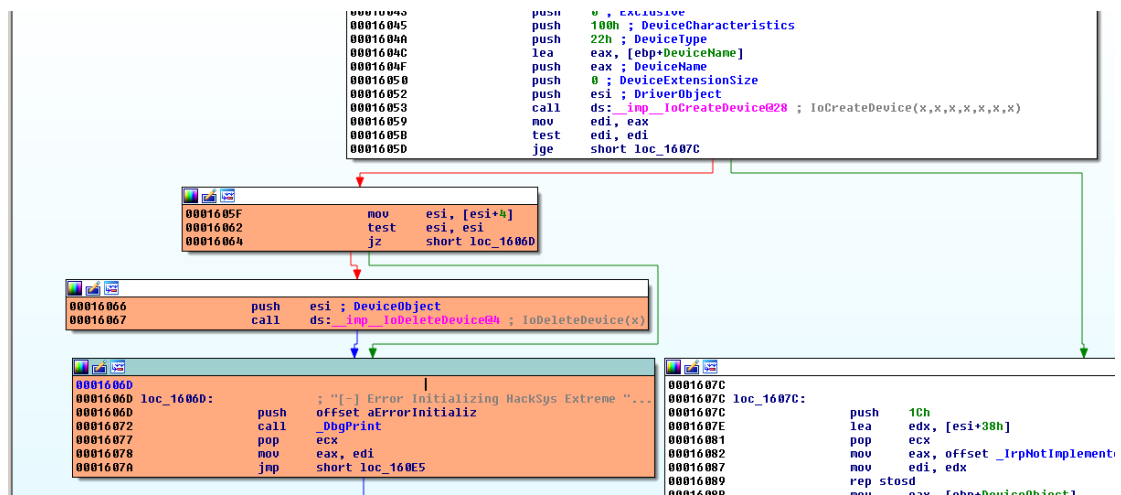
DeviceObject [out]

Pointer to a variable that receives a pointer to the newly created **DEVICE_OBJECT** structure. The **DEVICE_OBJECT** structure is allocated from nonpaged pool.

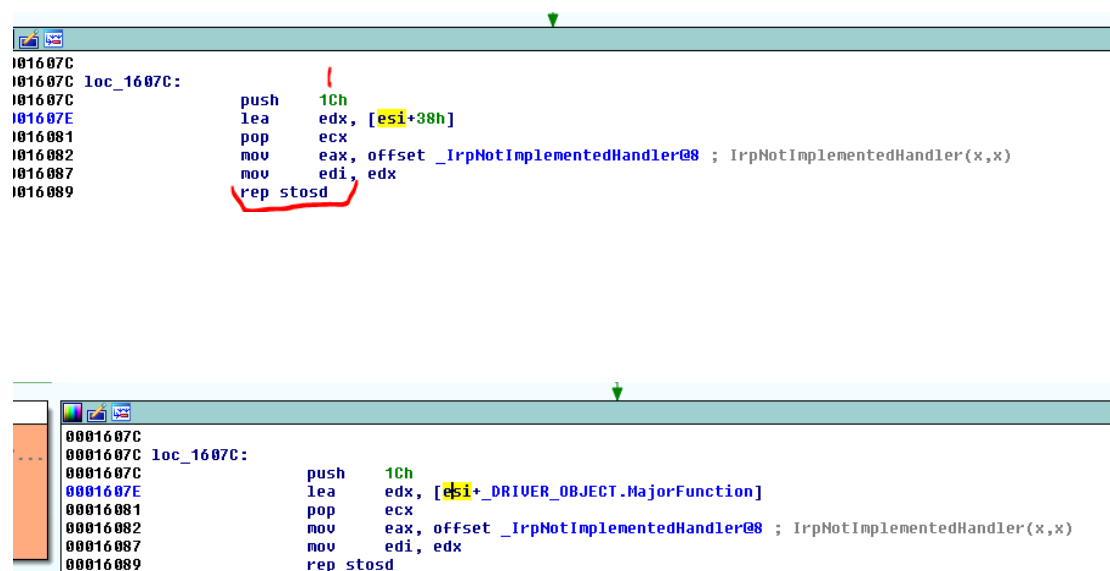
El último argumento es el puntero a la nueva estructura creada **DEVICE_OBJECT**.

```
00016024 stosw  
00016026 push offset aDeviceHacksyse ; "\\Device\\HackSysExtreme!  
0001602B lea eax, [ebp+DeviceName]  
0001602E push eax ; DestinationString  
0001602F call esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeSt  
00016031 push offset aDosdevicesHa_0 ; "\\DosDevices\\HackSysExt  
00016036 lea eax, [ebp+DosDeviceName]  
00016039 push eax ; DestinationString  
0001603A call esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeSt  
0001603C mov esi, [ebp+DriverObject]  
0001603F lea eax, [ebp+DeviceObject]  
00016042 push eax ; DeviceObject  
00016043 push 0 ; Exclusive  
00016045 push 100h ; DeviceCharacteristics  
0001604A push 22h ; DeviceType  
0001604C lea eax, [ebp+DeviceName]  
0001604F push eax ; DeviceName  
00016050 push 0 ; DeviceExtensionSize  
00016052 push esi ; DriverObject  
00016053 call ds:imp__IoCreateDevice@28 ; IoCreateDevice(x,x,x  
00016059 mov edi, eax  
0001605B test edi, edi  
0001605D jge short loc_1607C
```

Vemos que si el resultado de crear el Device es negativo lo cual se chequea en ese salto condicional signed, va a los bloques naranjas de error y se borra el Device Object con `IoDeleteDevice`.



Luego va a inicializar a partir de ESI + 38 como ESI apunta a DriverObject apretando T, puedo ver que campo es (sino esta DRIVER_OBJECT ir a LOCAL TYPES y sincronizarlo)

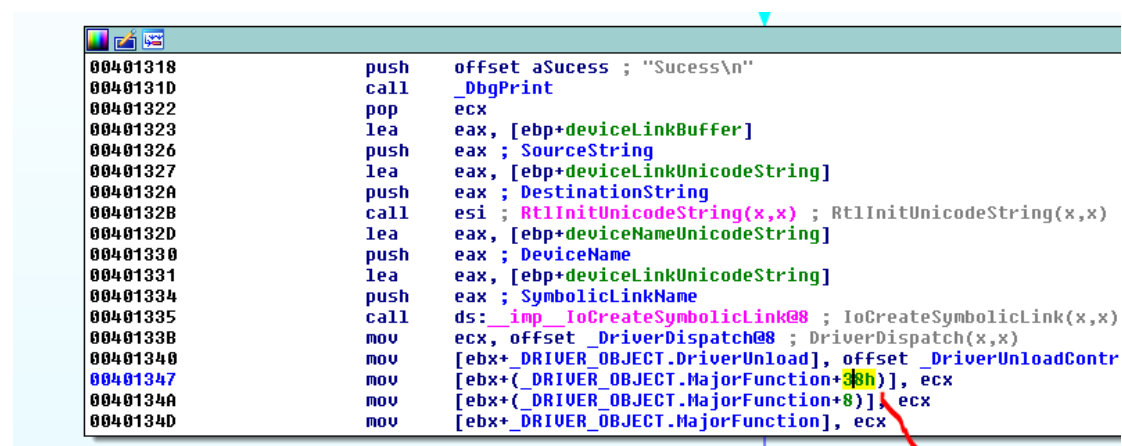


O sea es el puntero a la estructura MajorFunction que recordamos es una tablita de punteros que según la posición, me llevarán a diferentes funciones según el caso, recordemos que por ejemplo.

Below, the elements of the MajorFunction[] array are defined (also from *ntddk.h*):

```
[..]
#define IRP_MJ_CREATE                0x00
#define IRP_MJ_CREATE_NAMED_PIPE   0x01
#define IRP_MJ_CLOSE                 0x02
#define IRP_MJ_READ                  0x03
#define IRP_MJ_WRITE                 0x04
#define IRP_MJ_QUERY_INFORMATION     0x05
#define IRP_MJ_SET_INFORMATION      0x06
#define IRP_MJ_QUERY_EA              0x07
#define IRP_MJ_SET_EA                0x08
#define IRP_MJ_FLUSH_BUFFERS         0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION 0x0b
#define IRP_MJ_DIRECTORY_CONTROL    0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL  0x0d
#define IRP_MJ_DEVICE_CONTROL        0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
#define IRP_MJ_SHUTDOWN              0x10
#define IRP_MJ_LOCK_CONTROL          0x11
#define IRP_MJ_CLEANUP               0x12
```

El primer puntero o sea el que está en la posición 0 es IRP_MJ_CREATE y será donde saltará cuando utilice CreateFile para abrir el handle del Device, el segundo puntero o sea el valor 0x1 estará en la posición 4 ya que son DWORDs y así sucesivamente, quiere decir que inversamente si yo tengo un campo de esta estructura por su offset para saber que puntero es deberé dividir por cuatro, en el ejemplo que usábamos en los drivers anteriores recordamos que



```

00401318      push    offset aSucess ; "Sucess\n"
0040131D      call     _DbgPrint
00401322      pop      ecx
00401323      lea      eax, [ebp+deviceLinkBuffer]
00401326      push     eax ; SourceString
00401327      lea      eax, [ebp+deviceLinkUnicodeString]
0040132A      push     eax ; DestinationString
0040132B      call     esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
0040132D      lea      eax, [ebp+deviceNameUnicodeString]
00401330      push     eax ; DeviceName
00401331      lea      eax, [ebp+deviceLinkUnicodeString]
00401334      push     eax ; SymbolicLinkName
00401335      call     ds:imp__IoCreateSymbolicLink@8 ; IoCreateSymbolicLink(x,x)
0040133B      mov      ecx, offset _DriverDispatch@8 ; DriverDispatch(x,x)
00401340      mov      [ebx+_DRIVER_OBJECT.DriverUnload], offset _DriverUnloadContr
00401347      mov      [ebx+(_DRIVER_OBJECT.MajorFunction+38h)], ecx
0040134A      mov      [ebx+(_DRIVER_OBJECT.MajorFunction+8)], ecx
0040134D      mov      [ebx+_DRIVER_OBJECT.MajorFunction], ecx

```

Eso corresponde a 0x38/4 o sea

Python>hex(0x38/4)

0xe

```
#define IRP_MJ_FLUSH_BUFFERS 0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION 0x0b
#define IRP_MJ_DIRECTORY_CONTROL 0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL 0x0d
#define IRP_MJ_DEVICE_CONTROL 0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
#define IRP_MJ_SHUTDOWN 0x10
#define IRP_MJ_LOCK_CONTROL 0x11
#define IRP_MJ_CLEANUP 0x12
```

Que este 0xe era IRP_MJ_DEVICE_CONTROL cuando le pasábamos un IOCTL desde user, ese puntero lo pisábamos con un Dispatch para que según que IOCTL sea, se ejecute diferentes acciones mediante un switch por ejemplo.

En el caso actual vemos que inicializa a partir de un puntero al inicio de la tablita MajorFunction, copia el valor de EAX al cual le mueve un offset de una función que se llama _Irp_NotImplementedHandlers, y eso lo copia 0x1c veces que pasa a ECX, que es la cantidad de punteros a inicializar.

```
0001607C
0001607C loc_1607C:
0001607C
0001607E      push    1Ch
0001607E      lea     edx, [esi+_DRIVER_OBJECT.MajorFunction]
00016081      pop     ecx
00016082      mov     eax, offset _IrpNotImplementedHandler@8 ; IrpNotImplementedHandler(x,x)
00016087      mov     edi, edx
00016089      rep stosd
```

O sea que al principio toda la tablita la inicializa con este puntero que aparentemente no haría nada ya veremos, aparenta ser como un caso por default.

```
0001607C
0001607C loc_1607C:
0001607C      push    1Ch
0001607E      lea     edx, [esi+_DRIVER_OBJECT.MajorFunction]
00016081      pop     ecx
00016082      mov     eax, offset _IrpNotImplementedHandler@8 ; IrpNotImplementedHandler(x,x)
00016087      mov     edi, edx
00016089      rep stosd
0001608B      mov     eax, [ebp+DeviceObject]
0001608E      mov     dword ptr [edx], offset _IrpCreateHandler@8 ; IrpCreateHandler(x,x)
00016094      mov     [esi+(_DRIVER_OBJECT.MajorFunction+8)], offset _IrpCreateHandler@8 ; IrpCreateHandler(x,x)
0001609B      mov     [esi+(_DRIVER_OBJECT.MajorFunction+38h)], offset _IrpDeviceIoCtlHandler@8 ; IrpDeviceIoCtlHandler(x,x)
000160A2      mov     [esi+_DRIVER_OBJECT.DriverUnload], offset _IrpUnloadHandler@4 ; IrpUnloadHandler(x)
```

Como EDX tenía el puntero al inicio de MajorFunction su contenido es la posición 0 o sea

```
#define IRP_MJ_CREATE 0x00
```

```
#define IRP_MJ_CREATE_NAMED_PIPE 0x01
```

```
+#define IRP_MJ_CLOSE 0x02

+#define IRP_MJ_READ 0x03

+#define IRP_MJ_WRITE 0x04

+#define IRP_MJ_QUERY_INFORMATION 0x05

+#define IRP_MJ_SET_INFORMATION 0x06

+#define IRP_MJ_QUERY_EA 0x07

+#define IRP_MJ_SET_EA 0x08

+#define IRP_MJ_FLUSH_BUFFERS 0x09

+#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a

+#define IRP_MJ_SET_VOLUME_INFORMATION 0x0b

+#define IRP_MJ_DIRECTORY_CONTROL 0x0c

+#define IRP_MJ_FILE_SYSTEM_CONTROL 0x0d

+#define IRP_MJ_DEVICE_CONTROL 0x0e

+#define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f

+#define IRP_MJ SCSI 0x0f

+#define IRP_MJ_SHUTDOWN 0x10

+#define IRP_MJ_LOCK_CONTROL 0x11

+#define IRP_MJ_CLEANUP 0x12

+#define IRP_MJ_CREATE_MAILSLOT 0x13

+#define IRP_MJ_QUERY_SECURITY 0x14

+#define IRP_MJ_SET_SECURITY 0x15

+#define IRP_MJ_POWER 0x16

+#define IRP_MJ_SYSTEM_CONTROL 0x17

+#define IRP_MJ_DEVICE_CHANGE 0x18

+#define IRP_MJ_QUERY_QUOTA 0x19

+#define IRP_MJ_SET_QUOTA 0x1a

+#define IRP_MJ_PNP 0x1b

+#define IRP_MJ_PNP_POWER 0x1b

+#define IRP_MJ_MAXIMUM_FUNCTION 0x1b
```

Crearemos una estructura MajorFunction

```
struct __MajorFunction{
unsigned int _MJ_CREATE;
unsigned int _MJ_CREATE_NAMED_PIPE;
unsigned int _MJ_CLOSE;
unsigned int _MJ_READ;
unsigned int _MJ_WRITE;
unsigned int _MJ_QUERY_INFORMATION;
unsigned int _MJ_SET_INFORMATION;
unsigned int _MJ_QUERY_EA;
unsigned int _MJ_SET_EA;
unsigned int _MJ_FLUSH_BUFFERS;
unsigned int _MJ_QUERY_VOLUME_INFORMATION;
unsigned int _MJ_SET_VOLUME_INFORMATION;
unsigned int _MJ_DIRECTORY_CONTROL;
unsigned int _MJ_FILE_SYSTEM_CONTROL;
unsigned int _MJ_DEVICE_CONTROL;
unsigned int _MJ_INTERNAL_DEVICE_CONTROL;
unsigned int _MJ_SCSI;
unsigned int _MJ_SHUTDOWN;
unsigned int _MJ_LOCK_CONTROL;
unsigned int _MJ_CLEANUP;
unsigned int _MJ_CREATE_MAILSLLOT;
unsigned int _MJ_QUERY_SECURITY;
unsigned int _MJ_SET_SECURITY;
unsigned int _MJ_POWER;
unsigned int _MJ_SYSTEM_CONTROL;
unsigned int _MJ_DEVICE_CHANGE;
unsigned int _MJ_QUERY_QUOTA;
unsigned int _MJ_SET_QUOTA;
unsigned int _MJ_PNP;
unsigned int _MJ_PNP_POWER;
unsigned int _MJ_MAXIMUM_FUNCTION;
};
```

Se que son punteros pero para nuestro uso unsigned int funcionara, el tema es que el local types, usando INSERT no me la toma, asi que ahí mismo exporte , le agregue la estructura y la volví a cargar con FILE-LOAD FILE-PARSE C HEADER FILE y asi la tomo.

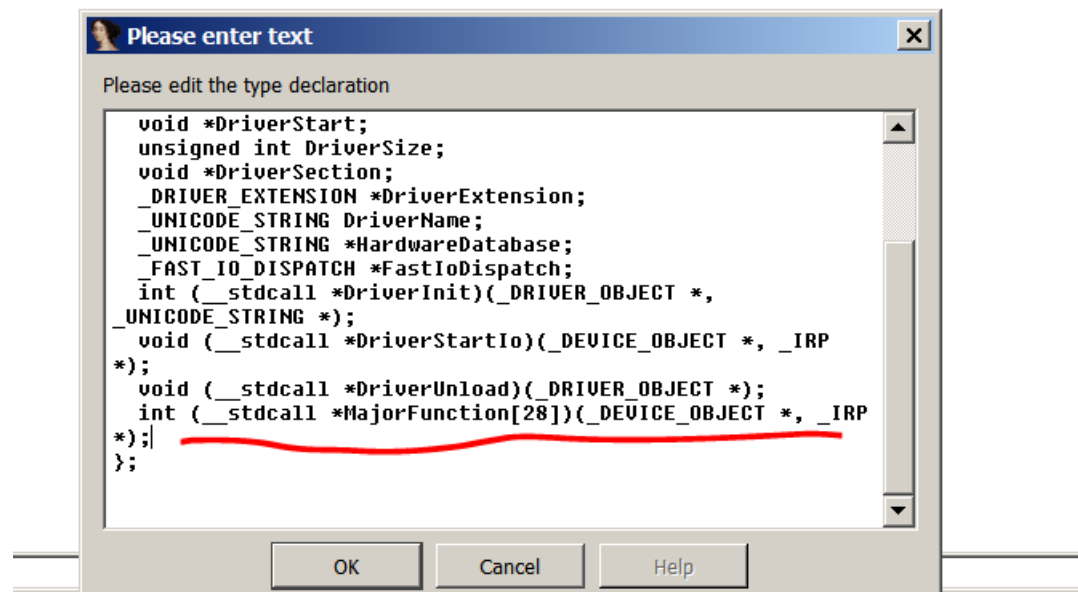
Index	Name	Value	Type	Structure
16	\$658AD709826129357C30...	00000004		struct {unsigned __int32 IoPriorityBoosted : 1;unsig
17	\$DE020C52BDF7FF700B105...	00000004		union {\$658AD709826129357C3064DC5428D5B8
18	_OWNER_ENTRY	00000008		struct {unsigned int OwnerThread;\$DE020C52BDF
19	_DEVICE_OBJECT	000000B8	Auto	struct { __int16 Type;unsigned __int16 Size;int Refe
20	_DRIVER_OBJECT	000000A8	Auto	stru
21	_UNICODE_STRING	00000008	Auto	stru
22	_DRIVER_EXTENSION	00000014	Auto	stru
23	_FILE_OBJECT	00000080		stru
24	\$FAF74743FBE1C8632047C...	00000008		stru

Line 19 of 496

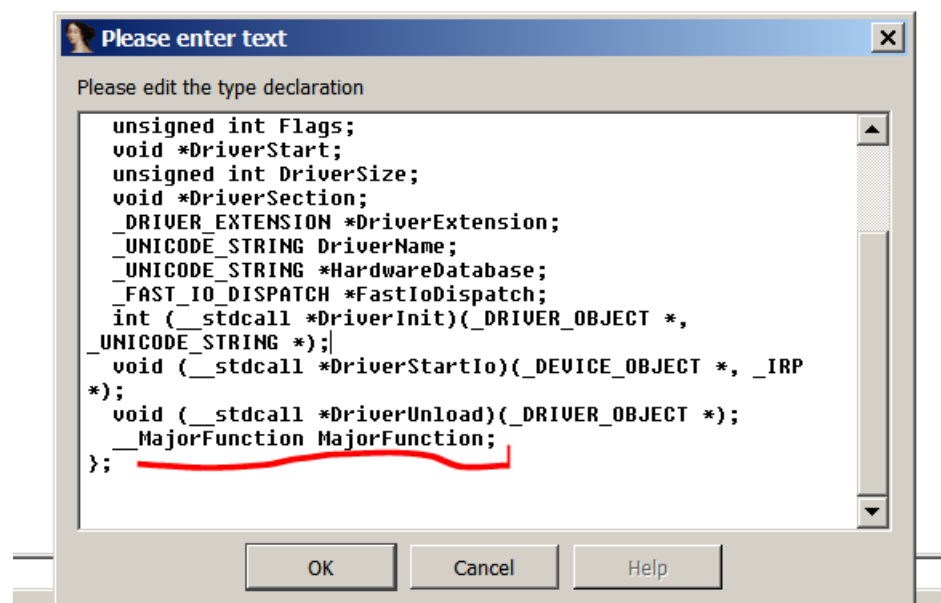
```

erable\i386\HEVD.h,2013: Syntax error near: _ACTIVATION_C
erable\i386\HEVD.h,2014: Syntax error near: _TP_CALLBACK_
erable\i386\HEVD.h,2018: Syntax error near: }
erable\i386\HEVD.h,2178: Syntax error near: _TP_CALLBACK_
erable\i386\HEVD.h,2179: Syntax error near: _TP_POOL
erable\i386\HEVD.h,2180: Syntax error near: }
erable\i386\HEVD.h,2228: Syntax error near: _TP_CALLBACK_INSTANCE
erable\i386\HEVD.h,2231: Syntax error near: }
erable\i386\HEVD.h,2237: Syntax error near: _TP_CALLBACK_INSTANCE
  
```

La agregue en el .h



Vemos que le cambie la definición del campo MajorFunction, dentro de la estructura DRIVER_OBJECT para que sea del tipo __MajorFunction que definí.



```

00010081      pop     ecx
00010082      mov     eax, offset _IrpNotImplementedHandler@8 ; IrpNotImplementedHandler(x,x)
00010087      mov     edi, edx
00010089      rep stosd
0001008B      mov     eax, [ebp+DeviceObject]
0001008E      mov     dword ptr [edx], offset _IrpCreateHandler@8 ; IrpCreateHandler(x,x)
00010094      mov     [esi+_DRIVER_OBJECT.MajorFunction.MJ_CLOSE], offset _IrpCreateHandler@8 ; IrpCreateHandler(x
0001009B      mov     [esi+_DRIVER_OBJECT.MajorFunction.MJ_DEVICE_CONTROL], offset _IrpDeviceIoCtlHandler@8 ; IrpD
000100A2      mov     [esi+_DRIVER_OBJECT.DriverUnload], offset _IrpUnloadHandler@4 ; IrpUnloadHandler(x)

```

Vemos que ahora si quedan definidos los campos con sus nombres de cada puntero.

Cuando hacemos T no se puede elegir la estructura Driver_Object porque EDX apunta a MajorFunction asi que elijo esta ultima.

The screenshot shows a debugger window with assembly code on the left and a 'Choose a structure for offset' dialog on the right. The assembly code is for a function named 'loc_1607C'. The dialog lists various structures and their offsets, with 'MajorFunction.MJ_CREATE' selected at the bottom.

Assembly Code (Left Panel):

```

0001607C      loc_1607C:
0001607C      push     1Ch
0001607C      lea     edx, [esi+_DRIVER_OB
00016081      pop     ecx
00016082      mov     eax, offset _IrpNotIm
00016087      mov     edi, edx
00016089      rep stosd
0001608B      mov     eax, [ebp+DeviceObject]
0001608E      mov     dword ptr [edx], offs
00016094      mov     [esi+_DRIVER_OBJECT.M
0001609B      mov     [esi+_DRIVER_OBJECT.M
000160A2      mov     [esi+_DRIVER_OBJECT.M
000160A9      or       dword ptr [eax+1Ch],
000160AD      mov     eax, [ebp+DeviceObject]
000160B0      and     dword ptr [eax+1Ch],
000160B7      lea     eax, [ebp+DeviceName]
000160BA      push    eax ; DeviceName
000160BB      lea     eax, [ebp+DosDeviceNa
000160BE      push    eax ; SymbolicLinkNa
000160BF      call    ds: _imp__IoCreateSym
000160C5      push    offset asc_1616C ; "
000160CA      push    offset aS ; "%S"
000160CF      mov     esi, eax
000160D1      call    _DbgPrint
000160D6      push    offset aHacksysExtr

```

Choose a structure for offset (Right Panel):

Operand representation
_EH4_SCOPETABLE.GSCookieOffset
_EH4_SCOPETABLE_RECORD.EndosingLevel
_DEVICE_OBJECT.Type
_LIST_ENTRY.Flink
_WAIT_CONTEXT_BLOCK.WaitQueueEntry.DeviceListEntry.Flink
_KDEVICE_QUEUE_ENTRY.DeviceListEntry.Flink
_KDEVICE_QUEUE.Type
_KDPC.Type
_KEVENT.Header.__u0.__s0.Type
_KEVENT.Header.__u0.Lock
_KEVENT.Header.__u0.__s2.gap0
_KEVENT.Header.__u0.__s3.gap0
_KEVENT.Header.__u0.__s4.gap0
_KEVENT.Header.__u0.__s5.gap0
_KEVENT.Header.__u0
_DISPATCHER_HEADER.__u0.__s0.Type
_DISPATCHER_HEADER.__u0.Lock
_DISPATCHER_HEADER.__u0.__s2.gap0
_DISPATCHER_HEADER.__u0.__s3.gap0
_DISPATCHER_HEADER.__u0.__s4.gap0
_DISPATCHER_HEADER.__u0.__s5.gap0
_DISPATCHER_HEADER.__u0
_DRIVER_OBJECT.Type
_DRIVER_EXTENSION.DriverObject
MajorFunction.MJ_CREATE

```

0001607C
0001607C loc_1607C:
0001607C      push    1Ch
0001607E      lea     edx, [esi+_DRIVER_OBJECT.MajorFunction]
00016081      pop     ecx
00016082      mov     eax, offset _IrpNotImplementedHandler@8 ; IrpNotImplementedHandler(x,x)
00016087      mov     edi, edx
00016089      rep stqsd
0001608B      mov     eax, [ebp+DeviceObject]
0001608E      mov     [edx+_MajorFunction.MJ_CREATE], offset _IrpCreateHandler@8 ; IrpCreateHandler(x,x)
00016094      mov     [esi+_DRIVER_OBJECT.MajorFunction.MJ_CLOSE], offset _IrpCreateHandler@8 ; IrpCreateHandler(x,x)
0001609B      mov     [esi+_DRIVER_OBJECT.MajorFunction.MJ_DEVICE_CONTROL], offset _IrpDeviceIoCtlHandler@8
000160A2      mov     [esi+_DRIVER_OBJECT.DriverUnload], offset _IrpUnloadHandler@4 ; IrpUnloadHandler(x)
000160A9      or      dword ptr [eax+1Ch], 10h
000160AD      mov     eax, [ebp+DeviceObject]
000160B0      and     dword ptr [eax+1Ch], 0FFFFFF7h

```

Ahora quedo mas lindo, esta bien determinado las funciones que usaran _MJ_CREATE, _MJ_CLOSE, _MJ_DEVICE_CONTROL y la que se llamara al detener el driver DriverUnload.

Obviamente cuando desde user hagamos CreateFile llamara a la función que pisa el campo _MJ_CREATE cuando pasemos un IOCTL a DeviceIoControl, llamara a _MJ_DEVICE_CONTROL, cuando se llame a CloseHandle, llamara a la que pisa _MJ_CLOSE y cuando se detenga la que pisa DriverUnload.

Miremos un poco la función que se llamara cuando se le pasen los IOCTL.

Sincronizamos la estructura IRP desde LOCAL TYPES

Como vimos en la parte 53 el campo 60 de IRP apunta a una estructura _IO_STACK_LOCATION que si figura en IDA aquí pasa lo mismo.

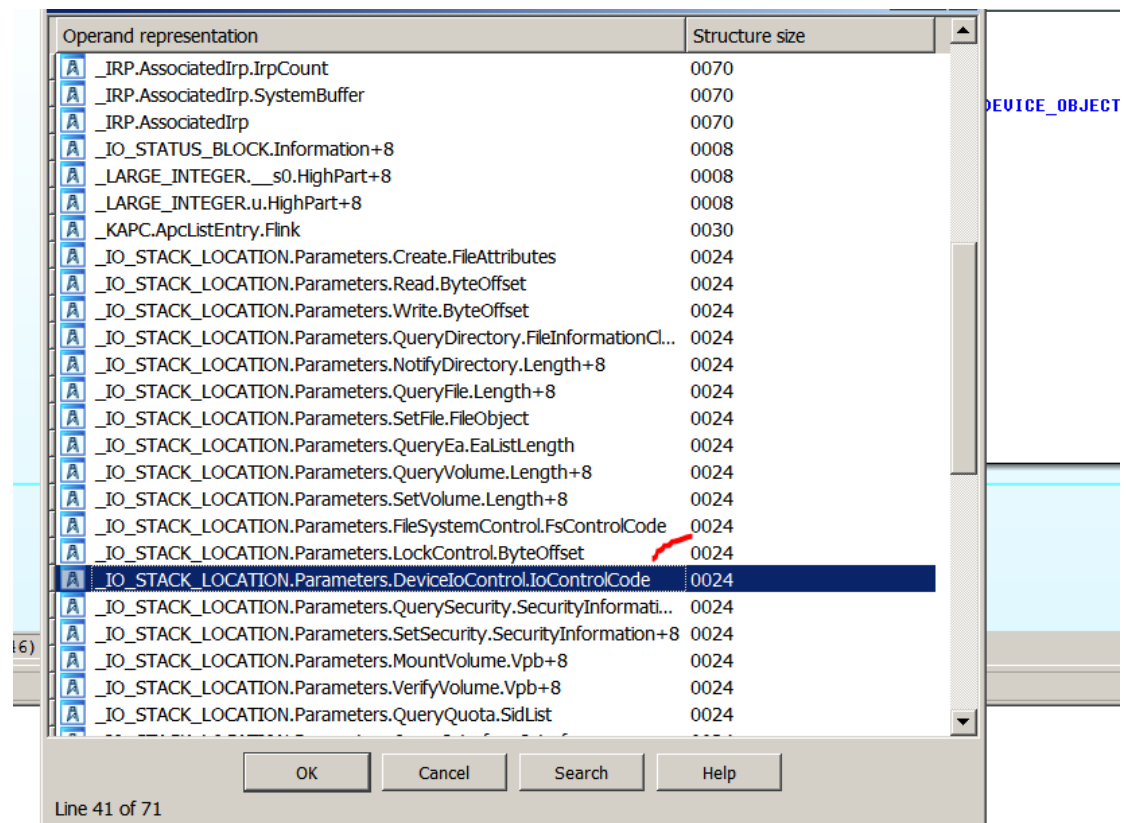

```

0001508E
0001508E
0001508E ; Attributes: bp-based frame
0001508E ; int __stdcall IrpDeviceIoCtlHandler(_DEVICE_OBJECT *DeviceObject, _IRP
0001508E _IrpDeviceIoCtlHandler@8 proc near
0001508E
0001508E DeviceObject = dword ptr 8
0001508E Irp = dword ptr 0Ch
0001508E
0001508E mov edi, edi
00015090 push ebp
00015091 mov ebp, esp
00015093 push ebx
00015094 push esi
00015095 push edi
00015096 mov edi, [ebp+Irp]
00015099 mov esi, [edi+60h]
0001509C mov edx, [esi+0Ch]
0001509F mov eax, 22201Fh
000150A4 cmp edx, eax
000150A6 ja loc_151A2

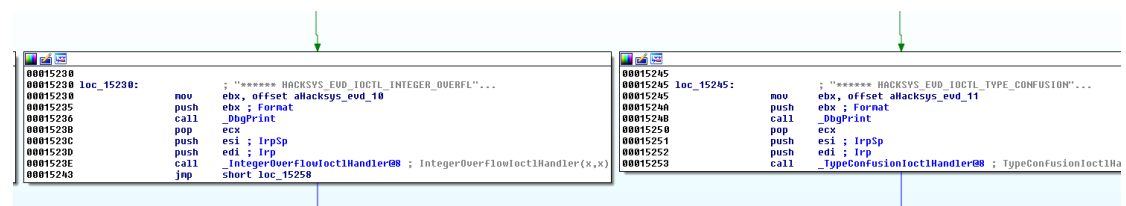
```

ESI aquí apunta a `_IO_STACK_LOCATION`, así que todo lo que sea `ESI+xxx` será un campo de dicha estructura, después de sincronizarla desde `LOCAL TYPES`.

Recordemos que `_IO_STACK_LOCATION` tiene varias opciones elegiré la correspondiente a `IoControlCode`.



Vemos que según el IOCTL el switch nos envía a diferentes bloques, y que los mismos están marcados con el tipo de vulnerabilidad que tiene cada camino.



```

00015172 loc_15172: ; "***** HACKSYS_EVD_STACKOVERFLOW *****"...
00015172 mov     ebx, offset aHacksys_evd_st
00015177 push   ebx ; Format
00015178 call   _DbgPrint
0001517D pop     ecx
0001517E push   esi ; IrpSp
0001517F push   edi ; Irp
00015180 call   _StackOverflowIoctlHandler@8 ; StackOverflowIoctlHandler(x,x)
00015185 jmp     loc_15258

```

Alli hay una que dice STACKOVERFLOW, asi que no hay que matarse mucho jeje.

```

00015172 loc_15172: ; "***** HACKSYS_EVD_STACKOVERFLOW *****"...
00015172 mov     ebx, offset aHacksys_evd_st
00015177 push   ebx ; Format
00015178 call   _DbgPrint
0001517D pop     ecx
0001517E push   esi ; IrpSp
0001517F push   edi ; Irp
00015180 call   _StackOverflowIoctlHandler@8 ; StackOverflowIoctlHandler(x,x)
00015185 jmp     loc_15258

```

Vemos que los dos argumentos que le pasa en EDI la estructura IRP y en ESI ahí dice IRPSP que es el nombre de la variable del tipo `_IO_STACK_LOCATION` que estaba en ESI.

```

000146FA ; Attributes: bp-based frame
000146FA ; int __stdcall StackOverflowIoctlHandler(_IRP *Irp, _IO_STACK_LOCATION *IrpSp)
000146FA _StackOverflowIoctlHandler@8 proc near
000146FA Irp           = dword ptr 8
000146FA IrpSp         = dword ptr 0Ch
000146FA mov     edi, edi
000146FC push   ebp
000146FD mov     ebp, esp
000146FF mov     ecx, [ebp+IrpSp]
00014702 mov     edx, [ecx+_IO_STACK_LOCATION.Parameters.DeviceIoControl.Type3InputBuffer]
00014705 mov     ecx, [ecx+8]
00014708 mov     eax, 0C0000001h
0001470D test    edx, edx
0001470F jz      short loc_14718

```

```

00000000 ,-----
00000000
00000000 $343716E60DEC8CEA3C236115305CA9A5 struct ; (sizeof=0x10, align=0x4, copyof_68)
00000000 ; XREF: $A9814862756C69703A76C8F506771A8D/r
00000000 OutputBufferLength dd ?,
00000004 InputBufferLength dd ?
00000008 IoControlCode dd ? ; XREF: IrpDeviceIoctlHandler(x,x)+E/r
0000000C Type3InputBuffer dd ? ; XREF: StackOverflowIoctlHandler(x,x)+8/r ; offset
00000010 $343716E60DEC8CEA3C236115305CA9A5 ends
00000010

```

Es un puntero a un buffer de entrada, también en la misma subestructura esta el IoControlCode y el largo del Buffer de entrada y de salida, supuestamente estos valores se los pasare yo, veamos que hace con ellos.

```

000146FA ; int __stdcall StackOverflowIoctlHandler(_IRP *Irp, _IO_STACK_LOCATION *IrpSp)
000146FA _StackOverflowIoctlHandler08 proc near
000146FA
000146FA Irp          = dword ptr 8
000146FA IrpSp        = dword ptr 0Ch
000146FA
000146FA mov     edi, edi
000146FA push   ebp
000146FA mov     ebp, esp
000146FD mov     ecx, [ebp+IrpSp]
000146FF mov     edx, [ecx+IO_STACK_LOCATION.Parameters.DeviceIoControl.Type3InputBuffer]
00014702 mov     ecx, [ecx+IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength]
00014705 mov     eax, 0C0000001h
00014708 test    edx, edx
0001470D jz      short loc_14718
0001470F
00014711 push    ecx ; Size
00014712 push    edx ; UserBuffer
00014713 call    _TriggerStackOverflow08 ; TriggerStackOverflow(x,x)
00014718
00014718 loc_14718:
00014718
00014718 non     phn

```

0, -21 (45,88) 00001705 00014705: StackOverflowIoctlHandler(x,x)+B (Synchronized with Hex View-1)

Vemos que ese size y ese buffer los pasa a la función _TriggerStackOverflow.

```

00014634
00014634 call    __SEH_prolog4
00014639 xor     esi, esi
0001463B xor     edi, edi
0001463D mov     [ebp+KernelBuffer], esi
00014643 push    7FCh ; size_t
00014648 push    esi ; int
00014649 lea     eax, [ebp+KernelBuffer+4]
0001464F push    eax ; void *
00014650 call    _memset

```

Vemos que pone a cero con ESI el primer DWORD del buffer KernelBuffer y luego con memset pone a cero desde el siguiente DWORD ya que le suma 4 a KernelBuffer, el size 0x7fc.

```

-0000001E          db ? ; undefined
-0000001D          db ? ; undefined
-0000001C  KernelBuffer dd 512 dup(?)
-0000001C  var_1C      dd ?
-00000018  ms_exc     CPPEH_RECORD ?
+00000000          db 4 dup(?)
+00000004          db 4 dup(?)
+00000008  UserBuffer dd ?
+0000000C  Size       dd ?
+00000010          ; 0
+00000010 ; end of stack variables

```

Dicho buffer tiene de largo 512 decimal por 4 ya que es un array de DWORD (dd) así que el largo total en decimal es

$512 * 4$

Out[64]: 2048

En HEXA es

hex(2048)

Out[65]: '0x800'

Por eso al poner el primer DWORD a cero y luego los 0x7fc restantes, realmente pone todo el buffer de 0x800 a cero. ($0x7fc + 4 = 0x800$)

Luego llama a ProbeForRead que chequea si el buffer de entrada en user está alineado y está en el espacio de user.

```

00014650          mov     [ebp+ms_exc.registration.ignore], esi
00014658          push    4 ; Alignment
0001465D          mov     esi, 800h
00014662          push    esi ; Length
00014663          push    [ebp+UserBuffer] ; Address
00014666          call    ds:__imp__ProbeForRead@12 ; ProbeForRead(x,x,x)

```

ProbeForRead routine

The **ProbeForRead** routine checks that a user-mode buffer actually resides in the user portion of the address space, and is correctly aligned.

Syntax

```
C++  
  
VOID ProbeForRead(  
    _In_ PVOID Address,  
    _In_ SIZE_T Length,  
    _In_ ULONG Alignment  
);
```

Parameters

Address [in]

Specifies the beginning of the user-mode buffer.

Length [in]

Specifies the length, in bytes, of the user-mode buffer.

Alignment [in]

Specifies the required alignment, in bytes, of the beginning of the user-mode buffer.

Luego imprime los punteros de los buffers y sus sizes.

```
00014662      push     esi ; Length  
00014663      push     [ebp+UserBuffer] ; Address  
00014666      call     ds:._imp_ProbeForRead@12 ; ProbeForRead(x,x,x)  
0001466C      push     [ebp+UserBuffer]  
0001466F      push     offset aUserBuffer0xP ; "[+] UserBuffer: 0x%p\n"  
00014674      call     _DbgPrint  
00014679      push     [ebp+Size]  
0001467C      push     offset aUserBufferSize ; "[+] UserBuffer Size: 0x%X\n"  
00014681      call     _DbgPrint  
00014686      lea     eax, [ebp+KernelBuffer]  
0001468C      push     eax  
0001468D      push     offset aKernelBuffer0x ; "[+] KernelBuffer: 0x%p\n"  
00014692      call     _DbgPrint  
00014697      push     esi  
00014698      push     offset aKernelbufferSi ; "[+] KernelBuffer Size: 0x%X\n"  
0001469D      call     _DbgPrint  
000146A2      push     offset aTriggeringSt_0 ; "[+] Triggering Stack Overflow\n"  
000146A7      call     _DbgPrint
```

Aquí vemos claramente el stack overflow, ya que usa el size que yo le paso como dato para copiar desde el buffer de entrada en user, al buffer en kernel que es el destination.

```
0001468D      push     offset aKernelBuffer0x ; "[+] KernelBuffer: 0x%p\n"  
00014692      call     _DbgPrint  
00014697      push     esi  
00014698      push     offset aKernelbufferSi ; "[+] KernelBuffer Size: 0x%X\n"  
0001469D      call     _DbgPrint  
000146A2      push     offset aTriggeringSt_0 ; "[+] Triggering Stack Overflow\n"  
000146A7      call     _DbgPrint  
000146AC      push     [ebp+Size] ; size_t  
000146AF      push     [ebp+UserBuffer] ; void *  
000146B2      lea     eax, [ebp+KernelBuffer]  
000146B8      push     eax ; void *  
000146B9      call     _memcpy  
000146BE      add     esp, 30h  
000146C1      jmp     short loc_146E4
```

000146E4
000146E4 loc_146E4:
000146E4

mov

```
0001465D      mov     esi, 800h
00014662      push    esi, Length
00014663      push    [ebp+UserBuffer] ; Address
00014666      call    ds:_imp_ProbeForRead@12 ; ProbeForRead(x,x,x)
0001466C      push    [ebp+UserBuffer]
0001466F      push    offset aUserbuffer0xP ; "[+] UserBuffer: 0x%p\n"
00014674      call    _DbgPrint
00014679      push    [ebp+Size]
0001467C      push    offset aUserbufferSize ; "[+] UserBuffer Size: 0x%X\n"
00014681      call    _DbgPrint
00014686      lea     eax, [ebp+KernelBuffer]
0001468C      push    eax
0001468D      push    offset aKernelbuffer0x ; "[+] KernelBuffer: 0x%p\n"
00014692      call    _DbgPrint
00014697      push    esi
00014698      push    offset aKernelbufferSi ; "[+] KernelBuffer Size: 0x%X\n"
0001469D      call    _DbgPrint
000146A2      push    offset aTriggeringSt_0 ; "[+] Triggering Stack Overflow\n"
000146A7      call    _DbgPrint
000146AC      push    [ebp+Size], size_t
000146AF      push    [ebp+UserBuffer] ; void *
000146B2      lea     eax, [ebp+KernelBuffer]
000146B8      push    eax ; void *
000146B9      call    _memcpy
000146BE      add     esp, 30h
000146C1      jmp     short loc_146E4
```

Ahí vemos que al imprimir el size del buffer de kernel, usa el que esta en ESI que es la constante 0x800, pero al hacer el memcpy, usa el argumento size que le paso yo, sin ningún tipo de chequeo lo cual producirá un stack overflow y como aquí no se ve cookie ni nada se podrá desbordar fácilmente.

En la parte siguiente haremos el script con la explotación aquí terminamos el análisis.

Hasta la parte siguiente

Ricardo Narvaja